Heart Attack Prediction using Machine Learning

Importing Libraries

In [1]:
```python
#Data Manipulation
import pandas as pd

#Numerical Python - Arrays
import numpy as np

#Visualization Libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

#Exploratory Data Analysis (EDA)
from collections import Counter
#Data Preprocessing
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

#Data Split
from sklearn.model_selection import train_test_split
#Data Modeling
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Reading the Dataset

In [2]:
```python
df = pd.read_csv('D:\\Khushi MCA\\MCA Semester 4\\archive (1)\\heart.csv')

df.head()
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

Total Number of Rows and Columns in the Dataset

In [3]:
```python
print("Number of Rows in the Dataset: ",df.shape[0])
print("Number of Columns in the Dataset: ",df.shape[1])
```

```
Number of Rows in the Dataset:  1025
Number of Columns in the Dataset:  14
```

Description of Columns

In [4]:
```python
df.describe()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	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
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	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
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000

In [5]:
```
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
```

Searching for Null and Duplicate Values

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
```

In [7]:
```
df_dup = df.duplicated().any()
print(df_dup)
```

```
True
```

In [8]:
```
df = df.drop_duplicates()
```

In [9]:
```
print("Number of Rows in the Dataset after Dropping Duplicate Values: ",df.shape[0])
print("Number of Columns in the Dataset after Dropping Duplicate Values: ",df.shape[1])
```

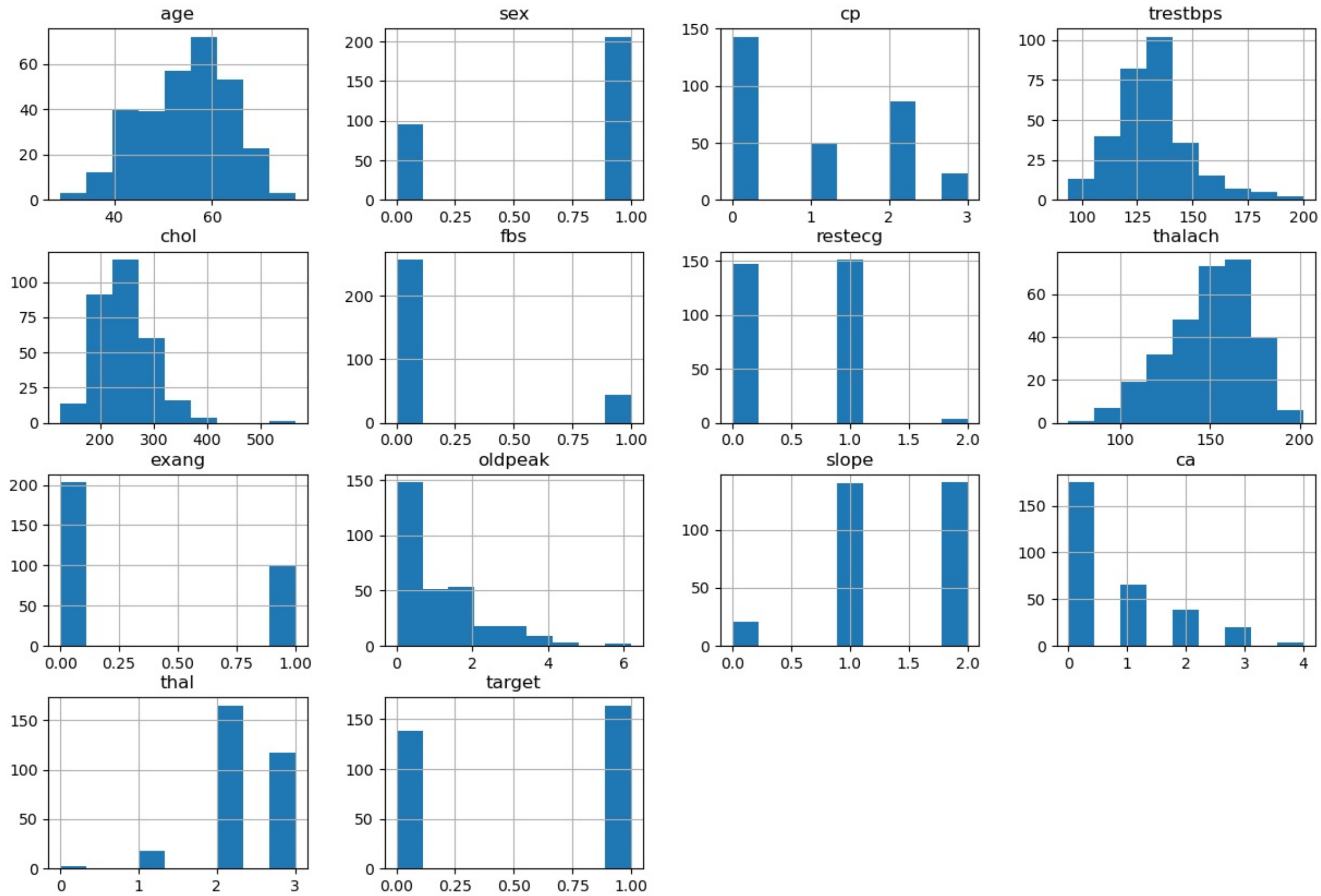
```
Number of Rows in the Dataset after Dropping Duplicate Values:  302
Number of Columns in the Dataset after Dropping Duplicate Values:  14
```

We do not have any missing values

Now lets have a look at the columns

In [10]:
```
df.hist(figsize=(15,10), bins=9)
```

Out[10]:
```
array([[<Axes: title={'center': 'age'}>, <Axes: title={'center': 'sex'}>,
        <Axes: title={'center': 'cp'}>,
        <Axes: title={'center': 'trestbps'}>],
       [<Axes: title={'center': 'chol'}>,
        <Axes: title={'center': 'fbs'}>,
        <Axes: title={'center': 'restecg'}>,
        <Axes: title={'center': 'thalach'}>],
       [<Axes: title={'center': 'exang'}>,
        <Axes: title={'center': 'oldpeak'}>,
        <Axes: title={'center': 'slope'}>,
        <Axes: title={'center': 'ca'}>],
       [<Axes: title={'center': 'thal'}>,
        <Axes: title={'center': 'target'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



In [11]:
```
info = ["age","0: female, 1: male","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain
        "serum cholestoral in mg/dl","fasting blood sugar > 120 mg/dl",
        "resting electrocardiographic results (values 0,1,2)",
        "maximum heart rate achieved","exercise induced angina","oldpeak = ST depression induced by exercise re
        "the slope of the peak exercise ST segment",
        "number of major vessels (0-3) colored by flourosopy","thal: 3 = normal; 6 = fixed defect; 7 = reversab

for i in range(len(info)):
    print(df.columns[i]+":\t\t\t"+info[i])
```

```
age:                        age
sex:                        0: female, 1: male
cp:                         chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asympto
matic
trestbps:                           resting blood pressure
chol:                       serum cholestoral in mg/dl
fbs:                        fasting blood sugar > 120 mg/dl
restecg:                            resting electrocardiographic results (values 0,1,2)
thalach:                            maximum heart rate achieved
exang:                      exercise induced angina
oldpeak:                            oldpeak = ST depression induced by exercise relative to rest
slope:                      the slope of the peak exercise ST segment
ca:                         number of major vessels (0-3) colored by flourosopy
thal:                       thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

Searching for unique values first

In [12]:
```
df['target'].value_counts()
```

Out[12]:
```
1    164
0    138
Name: target, dtype: int64
```

Exploratory Data Analysis (EDA)

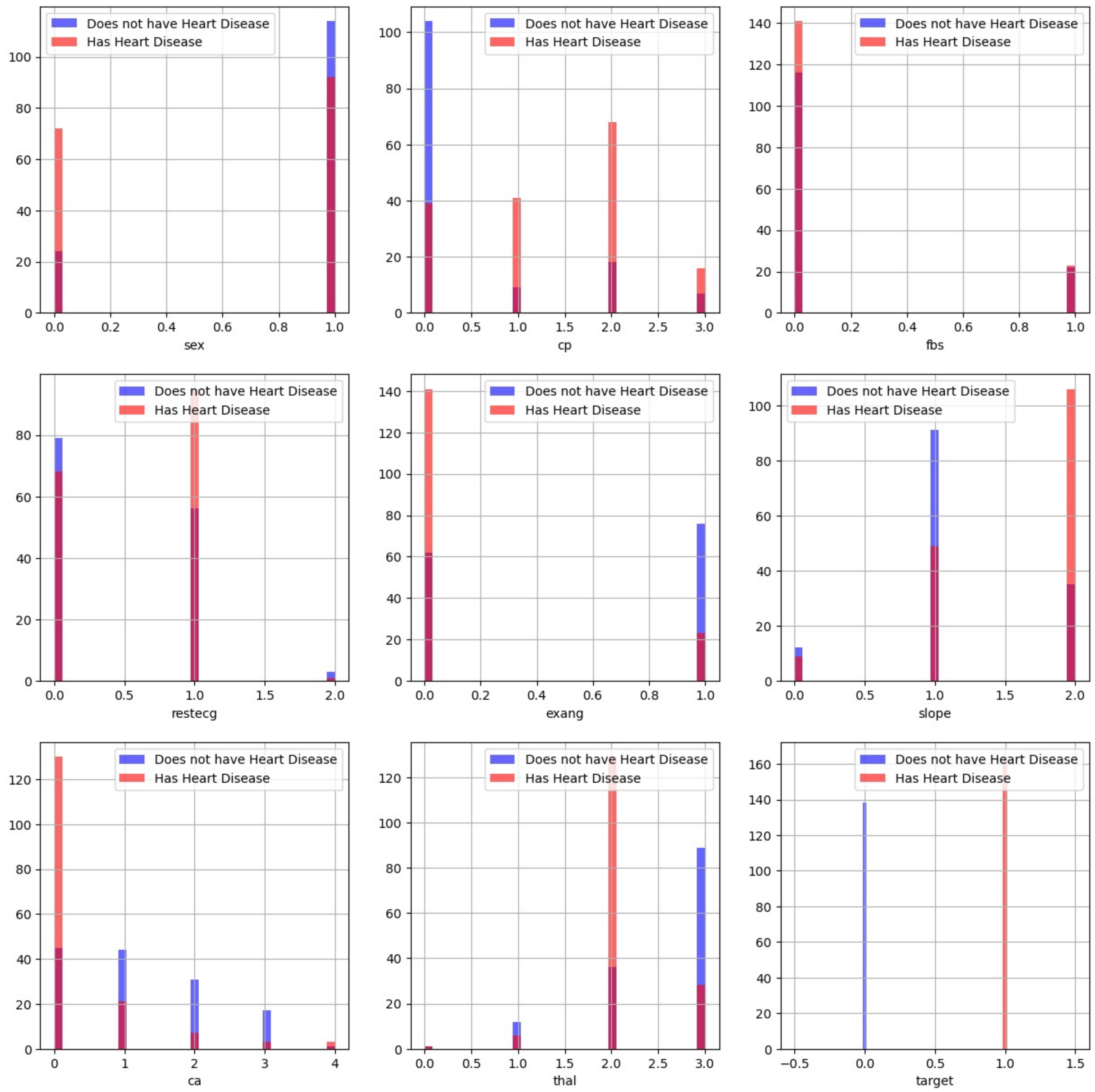
In [13]:
```
categorical_val = []
continous_val = []
for column in df.columns:
    print('==============================')
```

```
    print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```
==============================
age : [52 53 70 61 62 58 55 46 54 71 43 34 51 50 60 67 45 63 42 44 56 57 59 64
 65 41 66 38 49 48 29 37 47 68 76 40 39 77 69 35 74]
==============================
sex : [1 0]
==============================
cp : [0 1 2 3]
==============================
trestbps : [125 140 145 148 138 100 114 160 120 122 112 132 118 128 124 106 104 135
 130 136 180 129 150 178 146 117 152 154 170 134 174 144 108 123 110 142
 126 192 115  94 200 165 102 105 155 172 164 156 101]
==============================
chol : [212 203 174 294 248 318 289 249 286 149 341 210 298 204 308 266 244 211
 185 223 208 252 209 307 233 319 256 327 169 131 269 196 231 213 271 263
 229 360 258 330 342 226 228 278 230 283 241 175 188 217 193 245 232 299
 288 197 315 215 164 326 207 177 257 255 187 201 220 268 267 236 303 282
 126 309 186 275 281 206 335 218 254 295 417 260 240 302 192 225 325 235
 274 234 182 167 172 321 300 199 564 157 304 222 184 354 160 247 239 246
 409 293 180 250 221 200 227 243 311 261 242 205 306 219 353 198 394 183
 237 224 265 313 340 259 270 216 264 276 322 214 273 253 176 284 305 168
 407 290 277 262 195 166 178 141]
==============================
fbs : [0 1]
==============================
restecg : [1 0 2]
==============================
thalach : [168 155 125 161 106 122 140 145 144 116 136 192 156 142 109 162 165 148
 172 173 146 179 152 117 115 112 163 147 182 105 150 151 169 166 178 132
 160 123 139 111 180 164 202 157 159 170 138 175 158 126 143 141 167  95
 190 118 103 181 108 177 134 120 171 149 154 153  88 174 114 195 133  96
 124 131 185 194 128 127 186 184 188 130  71 137  99 121 187  97  90 129
 113]
==============================
exang : [0 1]
==============================
oldpeak : [1.  3.1 2.6 0.  1.9 4.4 0.8 3.2 1.6 3.  0.7 4.2 1.5 2.2 1.1 0.3 0.4 0.6
 3.4 2.8 1.2 2.9 3.6 1.4 0.2 2.  5.6 0.9 1.8 6.2 4.  2.5 0.5 0.1 2.1 2.4
 3.8 2.3 1.3 3.5]
==============================
slope : [2 0 1]
==============================
ca : [2 0 1 3 4]
==============================
thal : [3 2 1 0]
==============================
target : [0 1]
```

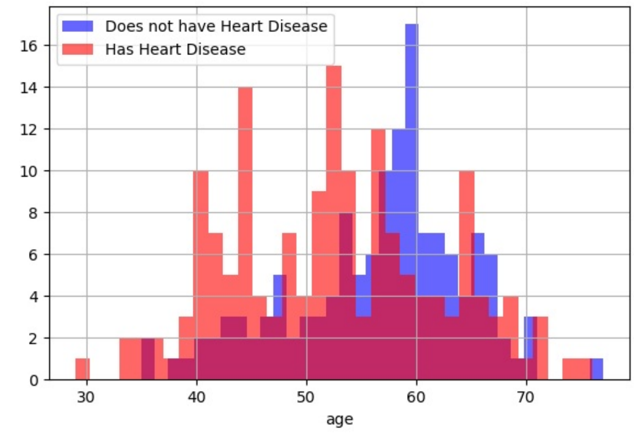
In [14]:

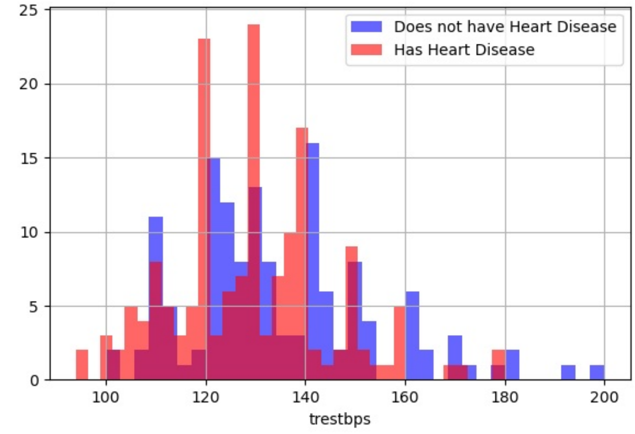
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1):
    plt.subplot(3, 3, i)
    df[df["target"] == 0][column].hist(bins=35, color='blue', label='Does not have Heart Disease', alpha=0.6)
    df[df["target"] == 1][column].hist(bins=35, color='red', label='Has Heart Disease', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
    plt.savefig('HeartDisease1.png')
```

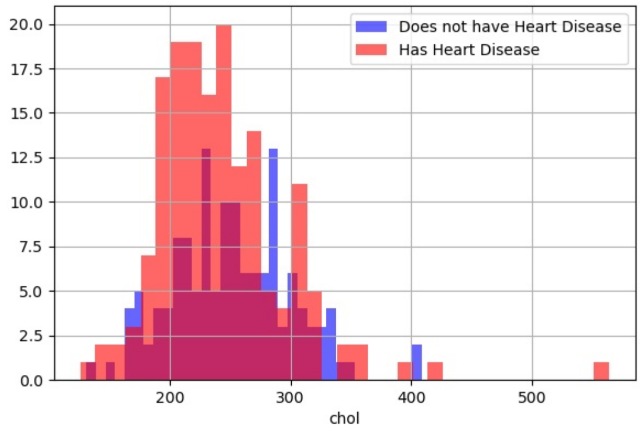


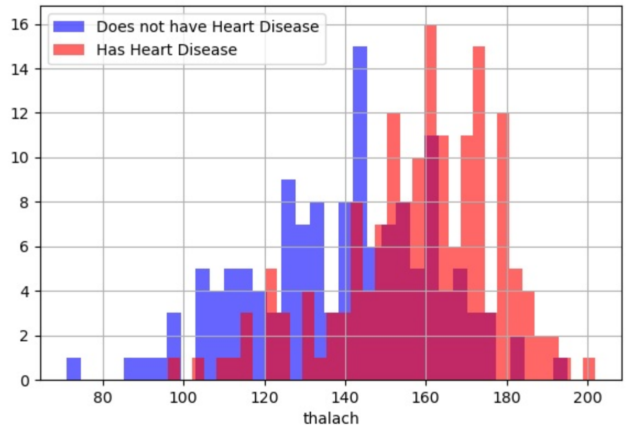
In [15]:

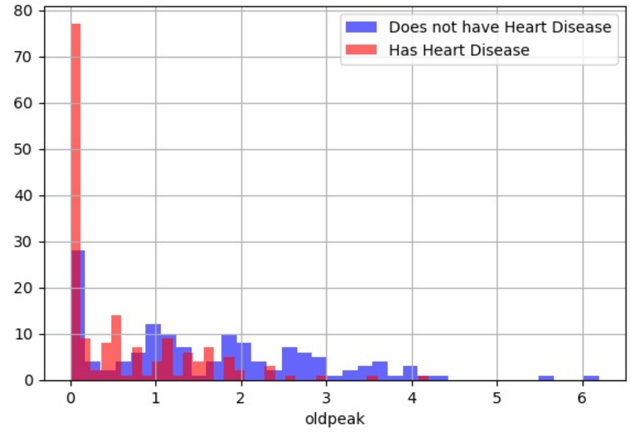
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(continous_val, 1):
    plt.subplot(3, 2, i)
    df[df["target"] == 0][column].hist(bins=35, color='blue', label='Does not have Heart Disease', alpha=0.6)
    df[df["target"] == 1][column].hist(bins=35, color='red', label='Has Heart Disease', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
    plt.savefig('HeartDisease2.png')
```











Splitting the Data into Training and Testing Dataset

In [16]:
```
y = df["target"]
X = df.drop('target', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

print("X_Train Shape is: ",X_train)
print("X_Test Shape is: ", X_test)

print("y_Train Shape is: ", y_train)
print("y_Test Shape is: ", y_test)
```

```
X_Train Shape is:       age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
81    49    1   2       118   149    0        0      126      0      0.8
193   69    1   3       160   234    1        0      131      0      0.1
70    59    1   0       170   326    0        0      140      1      3.4
719   52    1   0       108   233    1        1      147      0      0.1
628   69    0   3       140   239    0        1      151      0      1.8
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
425   51    0   0       130   305    0        1      142      1      1.2
271   44    1   1       120   263    0        1      173      0      0.0
143   34    1   3       118   182    0        0      174      0      0.0
50    58    0   3       150   283    1        0      162      0      1.0
232   60    1   0       125   258    0        0      141      1      2.8

     slope  ca  thal
81       2   3     2
193      1   1     2
70       0   0     3
719      2   3     3
628      2   2     2
..     ...  ..   ...
425      1   0     3
271      2   0     3
143      2   0     2
50       2   0     2
232      1   1     3

[241 rows x 13 columns]
X_Test Shape is:       age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
342   65    0   2       155   269    0        1      148      0      0.8
191   56    1   1       130   221    0        0      163      0      0.0
349   62    0   2       130   263    0        1       97      0      1.2
288   58    0   2       120   340    0        1      172      0      0.0
56    56    1   3       120   193    0        0      162      0      1.9
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
182   60    1   0       140   293    0        0      170      0      1.2
878   54    1   0       120   188    0        1      113      0      1.4
27    58    0   1       136   319    1        0      152      0      0.0
128   52    1   2       138   223    0        1      169      0      0.0
102   54    1   1       108   309    0        1      156      0      0.0

     slope  ca  thal
342      2   0     2
191      2   0     3
349      1   1     3
288      2   0     2
56       1   0     3
..     ...  ..   ...
182      1   2     3
878      1   1     3
27       2   2     2
128      2   4     2
102      2   0     3

[61 rows x 13 columns]
y_Train Shape is:  81     0
193    1
70     0
719    1
628    1
      ..
425    0
271    1
143    1
50     1
232    0
Name: target, Length: 241, dtype: int64
y_Test Shape is:  342    1
191    1
349    0
288    1
56     1
      ..
182    0
878    0
27     0
128    1
102    1
Name: target, Length: 61, dtype: int64
```

Checking for Unique Values

In [17]:
```
print(y_test.unique())
Counter(y_train)
```

```
[1 0]
```

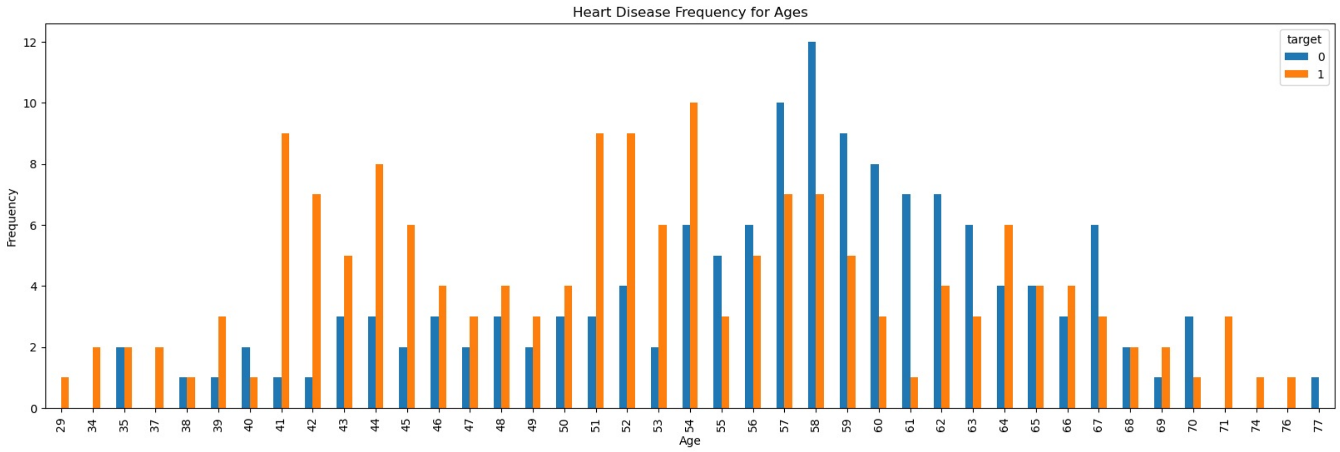
Out[17]:
```
Counter({0: 113, 1: 128})
```

Standardizing

In [18]:
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

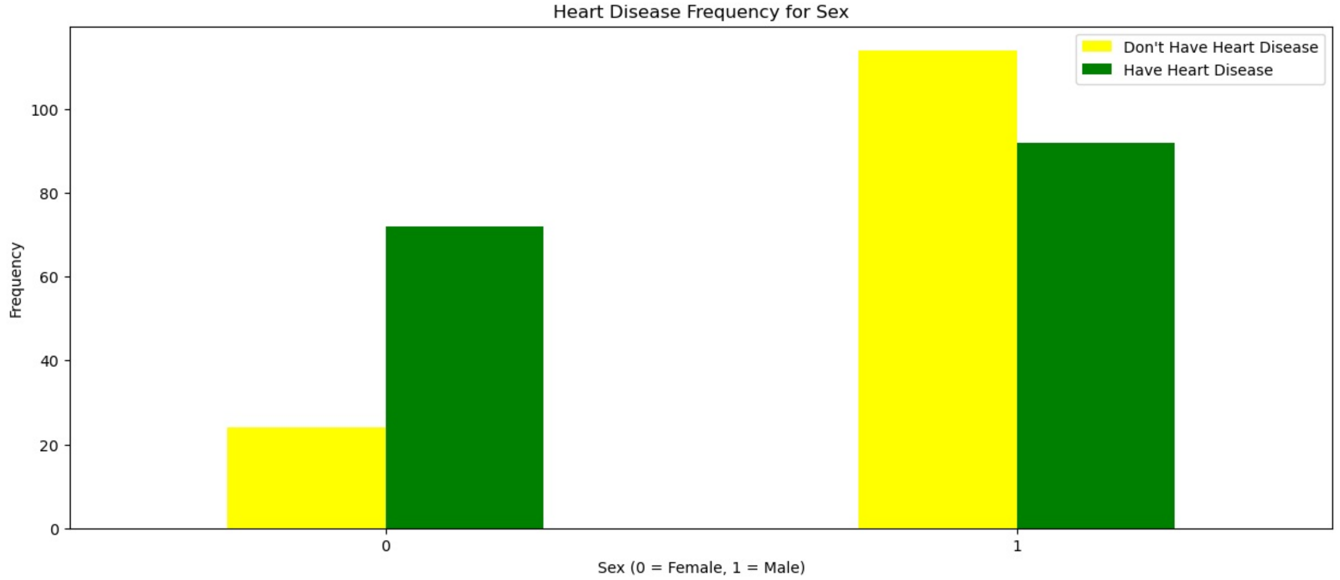
Heart Disease Frequency for Ages

In [19]:
```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```



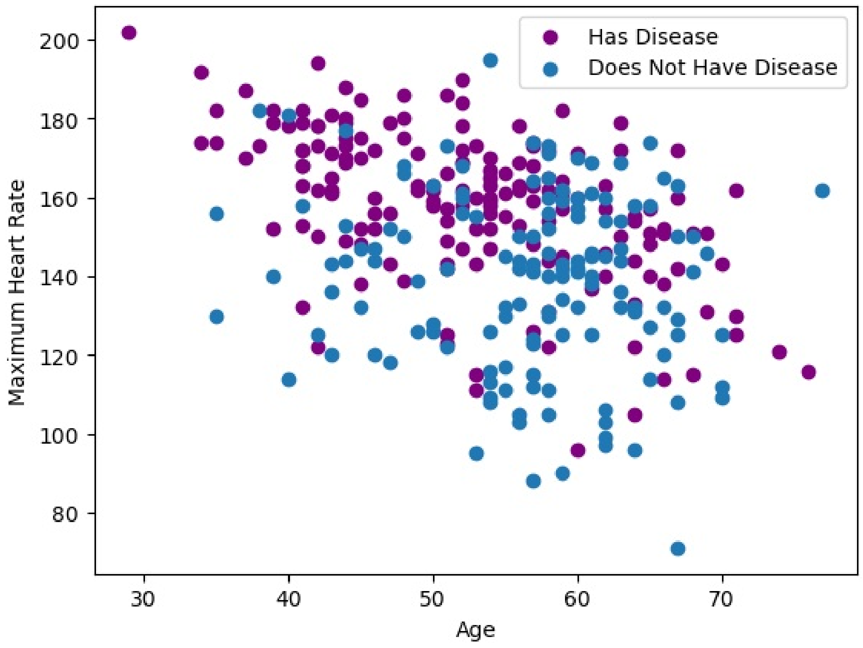
Heart Disease Frequency according to the Genders - i.e Males and Female Frequencies

In [20]:
```
pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['yellow','green' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Don't Have Heart Disease", "Have Heart Disease"])
plt.ylabel('Frequency')
plt.savefig('HeartDiseaseFrequency.png')
plt.show()
```



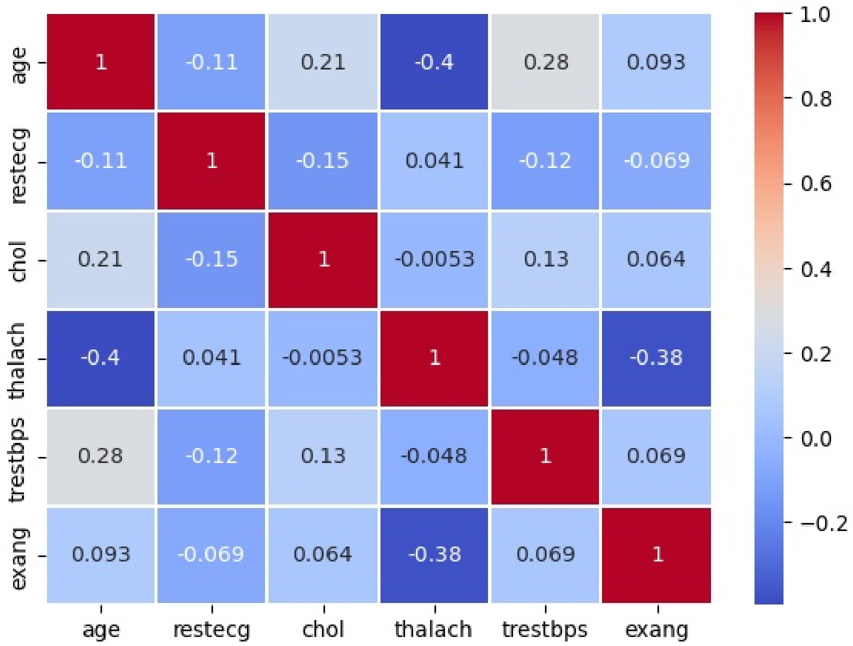
Age wise Heart Disease Rate

In [21]:
```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="purple")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Has Disease", "Does Not Have Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.savefig('HeartDiseaseRate.png')
plt.show()
```



Feature Selection

In [22]:
```
names=['age','restecg','chol','thalach','trestbps','exang']


#Set the width and height of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Correlation plot
corr = df.loc[:,names]
#Generate correlation matrix
correlation = corr.corr()

#Plot using seaborn library
sns.heatmap(correlation, annot = True, cmap='coolwarm',linewidths=.1)
plt.savefig('FeatureSelection.png')
plt.show()
```



In [23]:
```
corr
```

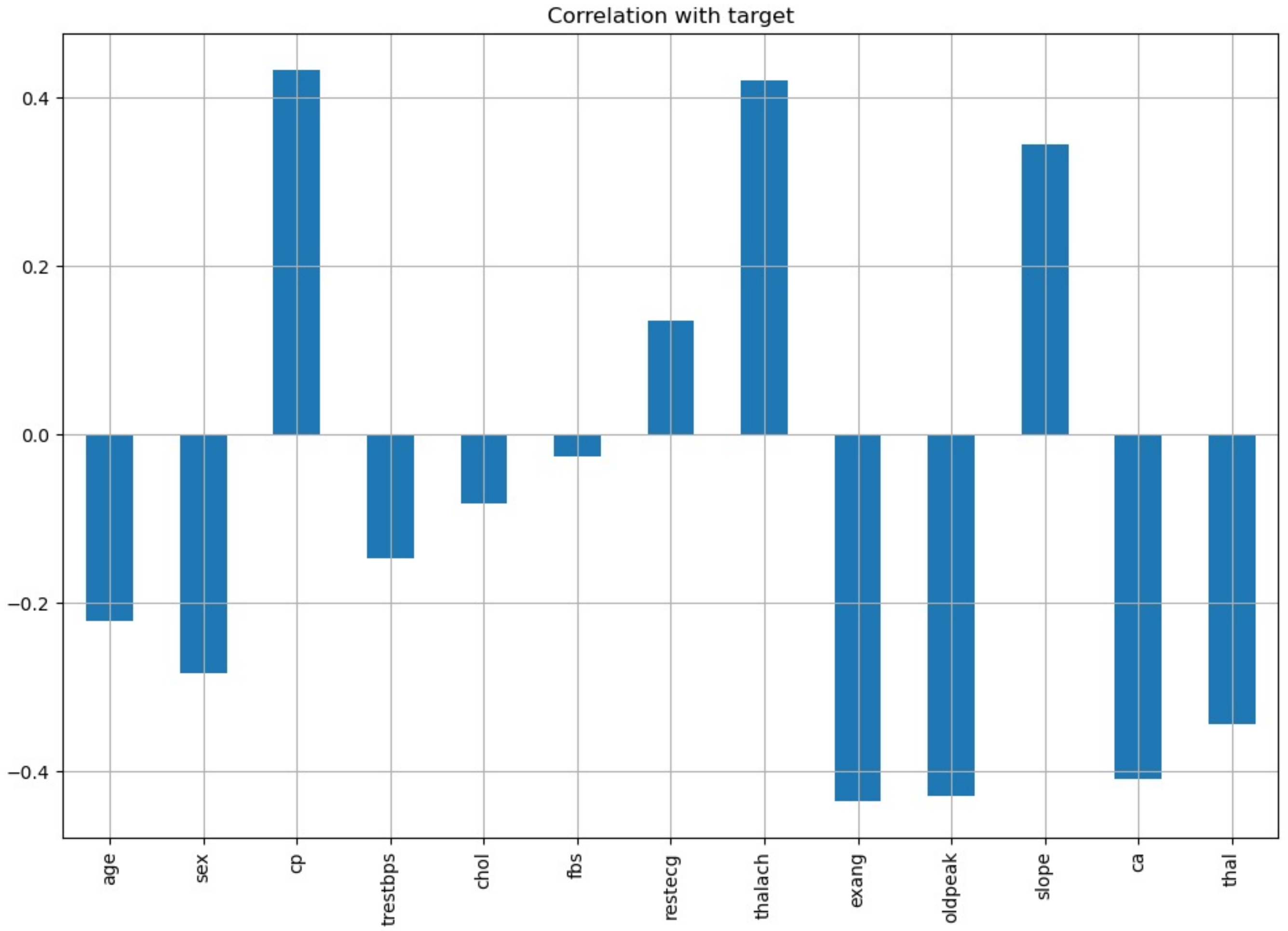
Out[23]:

	age	restecg	chol	thalach	trestbps	exang
0	52	1	212	168	125	0
1	53	0	203	155	140	1
2	70	1	174	125	145	1
3	61	1	203	161	148	0
4	62	1	294	106	138	0
...	...	...	...	...	...	...
723	68	0	211	115	120	0
733	44	1	141	175	108	0
739	52	1	255	161	128	1
843	59	0	273	125	160	0
878	54	1	188	113	120	0

302 rows × 6 columns

In [24]:

```
df.drop('target', axis=1).corrwith(df.target).plot(kind='bar', grid=True, figsize=(12, 8),
                                                   title="Correlation with target")
plt.savefig('CorrwithTarget.png')
```



Data Normalization

In [25]:

```
x= preprocessing.StandardScaler().fit(X).transform(X.astype(float))
x[0:5]
```

Out[25]:

```
array([[-0.26796589,  0.68265615, -0.93520799, -0.37655636, -0.66772815,
        -0.41844626,  0.90165655,  0.80603539, -0.69834428, -0.03712404,
         0.97951442,  1.27497996,  1.11996657],
       [-0.15726042,  0.68265615, -0.93520799,  0.47891019, -0.84191811,
         2.38979311, -1.0025412 ,  0.23749516,  1.43195847,  1.77395808,
        -2.27118179, -0.71491124,  1.11996657],
       [ 1.72473259,  0.68265615, -0.93520799,  0.76406571, -1.40319685,
        -0.41844626,  0.90165655, -1.07452077,  1.43195847,  1.34274805,
        -2.27118179, -0.71491124,  1.11996657],
       [ 0.72838335,  0.68265615, -0.93520799,  0.93515902, -0.84191811,
        -0.41844626,  0.90165655,  0.49989834, -0.69834428, -0.8995441 ,
         0.97951442,  0.28003436,  1.11996657],
       [ 0.83908882, -1.46486632, -0.93520799,  0.36484799,  0.91933586,
         2.38979311,  0.90165655, -1.90546419, -0.69834428,  0.73905401,
        -0.64583368,  2.26992556, -0.51399432]])
```

Now lets test different Machine Learning Models to Predict Heart Attack

First Model - Logistic Regression

In [26]:
```
model1 = 'Logistic Regression'
lr = LogisticRegression()
model = lr.fit(X_train, y_train)
lr_predict = lr.predict(X_test)
lr_conf_matrix = confusion_matrix(y_test, lr_predict)
lr_acc_score = accuracy_score(y_test, lr_predict)
print("Confusion Matrix")
print(lr_conf_matrix)
print("\n")
print("Accuracy of Logistic Regression:",lr_acc_score*100,'\n')
print(classification_report(y_test,lr_predict))
```

```
Confusion Matrix
[[20  5]
 [ 5 31]]


Accuracy of Logistic Regression: 83.60655737704919

              precision    recall  f1-score   support

           0       0.80      0.80      0.80        25
           1       0.86      0.86      0.86        36

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

Second Model - Naive Bayes

In [27]:
```
model2 = 'Naive Bayes'
nb = GaussianNB()
nb.fit(X_train,y_train)
nbpred = nb.predict(X_test)
nb_conf_matrix = confusion_matrix(y_test, nbpred)
nb_acc_score = accuracy_score(y_test, nbpred)
print("Confusion Matrix")
print(nb_conf_matrix)
print("\n")
print("Accuracy of Naive Bayes model:",nb_acc_score*100,'\n')
print(classification_report(y_test,nbpred))
```

```
Confusion Matrix
[[20  5]
 [ 7 29]]


Accuracy of Naive Bayes model: 80.32786885245902

              precision    recall  f1-score   support

           0       0.74      0.80      0.77        25
           1       0.85      0.81      0.83        36

    accuracy                           0.80        61
   macro avg       0.80      0.80      0.80        61
weighted avg       0.81      0.80      0.80        61
```

Third Model - Random Forest Classifier

In [28]:
```
model3 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=20, random_state=12,max_depth=5)
rf.fit(X_train,y_train)
rf_predicted = rf.predict(X_test)
rf_conf_matrix = confusion_matrix(y_test, rf_predicted)
rf_acc_score = accuracy_score(y_test, rf_predicted)
print("Confusion Matrix")
print(rf_conf_matrix)
print("\n")
print("Accuracy of Random Forest:",rf_acc_score*100,'\n')
print(classification_report(y_test,rf_predicted))
```

```
Confusion Matrix
[[18  7]
 [ 5 31]]


Accuracy of Random Forest: 80.32786885245902

              precision    recall  f1-score   support

           0       0.78      0.72      0.75        25
           1       0.82      0.86      0.84        36

    accuracy                           0.80        61
   macro avg       0.80      0.79      0.79        61
weighted avg       0.80      0.80      0.80        61
```

Fourth Model - K-Neighbors Classifier

In [29]:

```python
model4 = 'K-NeighborsClassifier'
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
knn_predicted = knn.predict(X_test)
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
knn_acc_score = accuracy_score(y_test, knn_predicted)
print("Confusion Matrix")
print(knn_conf_matrix)
print("\n")
print("Accuracy of K-NeighborsClassifier:",knn_acc_score*100,'\n')
print(classification_report(y_test,knn_predicted))
```

```
Confusion Matrix
[[19  6]
 [ 5 31]]


Accuracy of K-NeighborsClassifier: 81.9672131147541

              precision    recall  f1-score   support

           0       0.79      0.76      0.78        25
           1       0.84      0.86      0.85        36

    accuracy                           0.82        61
   macro avg       0.81      0.81      0.81        61
weighted avg       0.82      0.82      0.82        61
```

Fifth Model - Decision Tree Classifier

In [30]:

```python
model5 = 'DecisionTreeClassifier'
dt = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 6)
dt.fit(X_train, y_train)
dt_predicted = dt.predict(X_test)
dt_conf_matrix = confusion_matrix(y_test, dt_predicted)
dt_acc_score = accuracy_score(y_test, dt_predicted)
print("Confusion Matrix")
print(dt_conf_matrix)
print("\n")
print("Accuracy of DecisionTreeClassifier:",dt_acc_score*100,'\n')
print(classification_report(y_test,dt_predicted))
```

```
Confusion Matrix
[[18  7]
 [ 6 30]]


Accuracy of DecisionTreeClassifier: 78.68852459016394

              precision    recall  f1-score   support

           0       0.75      0.72      0.73        25
           1       0.81      0.83      0.82        36

    accuracy                           0.79        61
   macro avg       0.78      0.78      0.78        61
weighted avg       0.79      0.79      0.79        61
```

Sixth Model - Support Vector Classifier

In [31]:

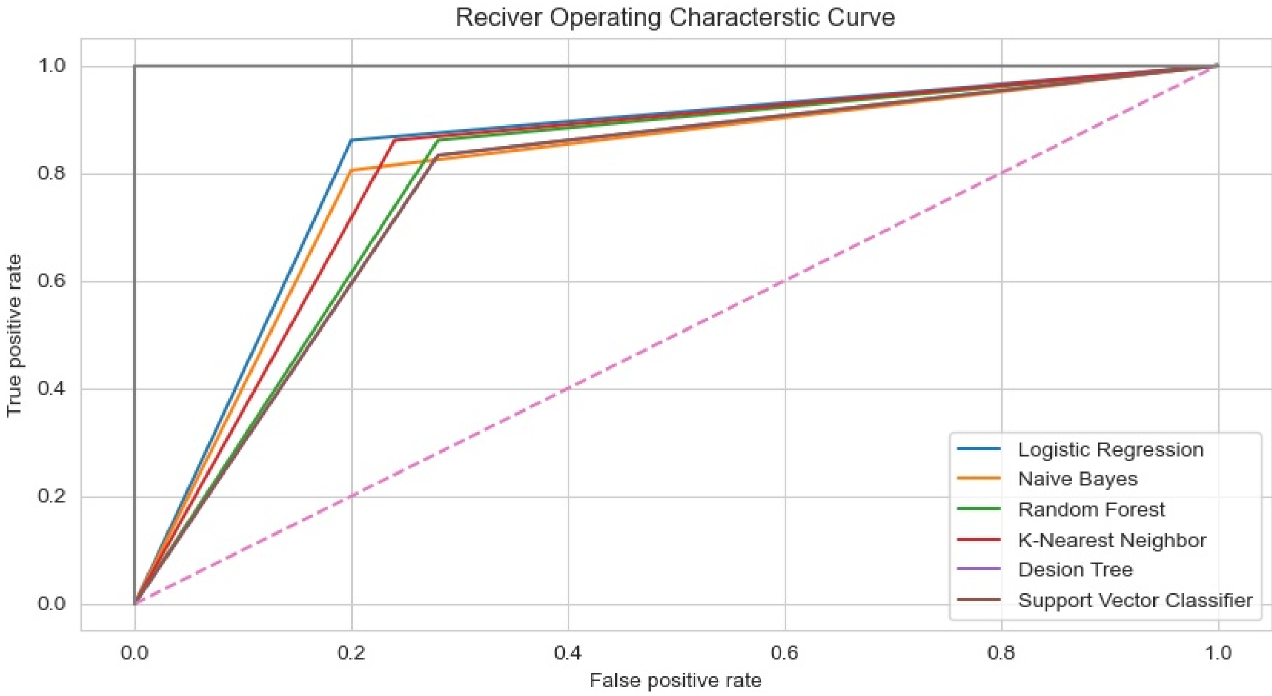
```python
model6 = 'Support Vector Classifier'
svc =  SVC(kernel='rbf', C=2)
svc.fit(X_train, y_train)
svc_predicted = svc.predict(X_test)
svc_conf_matrix = confusion_matrix(y_test, svc_predicted)
svc_acc_score = accuracy_score(y_test, svc_predicted)
```

```
print("Confusion matrix")
print(svc_conf_matrix)
print("\n")
print("Accuracy of Support Vector Classifier:",svc_acc_score*100,'\n')
print(classification_report(y_test,svc_predicted))
```

```
Confusion matrix
[[18  7]
 [ 6 30]]


Accuracy of Support Vector Classifier: 78.68852459016394

              precision    recall  f1-score   support

           0       0.75      0.72      0.73        25
           1       0.81      0.83      0.82        36

    accuracy                           0.79        61
   macro avg       0.78      0.78      0.78        61
weighted avg       0.79      0.79      0.79        61
```

In [32]:

```
lr_false_positive_rate,lr_true_positive_rate,lr_threshold = roc_curve(y_test,lr_predict)
nb_false_positive_rate,nb_true_positive_rate,nb_threshold = roc_curve(y_test,nbpred)
rf_false_positive_rate,rf_true_positive_rate,rf_threshold = roc_curve(y_test,rf_predicted)
knn_false_positive_rate,knn_true_positive_rate,knn_threshold = roc_curve(y_test,knn_predicted)
dt_false_positive_rate,dt_true_positive_rate,dt_threshold = roc_curve(y_test,dt_predicted)
svc_false_positive_rate,svc_true_positive_rate,svc_threshold = roc_curve(y_test,svc_predicted)


sns.set_style('whitegrid')
plt.figure(figsize=(10,5))
plt.title('Reciver Operating Characterstic Curve')
plt.plot(lr_false_positive_rate,lr_true_positive_rate,label='Logistic Regression')
plt.plot(nb_false_positive_rate,nb_true_positive_rate,label='Naive Bayes')
plt.plot(rf_false_positive_rate,rf_true_positive_rate,label='Random Forest')
plt.plot(knn_false_positive_rate,knn_true_positive_rate,label='K-Nearest Neighbor')
plt.plot(dt_false_positive_rate,dt_true_positive_rate,label='Desion Tree')
plt.plot(svc_false_positive_rate,svc_true_positive_rate,label='Support Vector Classifier')
plt.plot([0,1],ls='--')
plt.plot([0,0],[1,0],c='.5')
plt.plot([1,1],c='.5')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.legend()
plt.savefig('Output1.png')
plt.show()
```



Model Summary

In [33]:

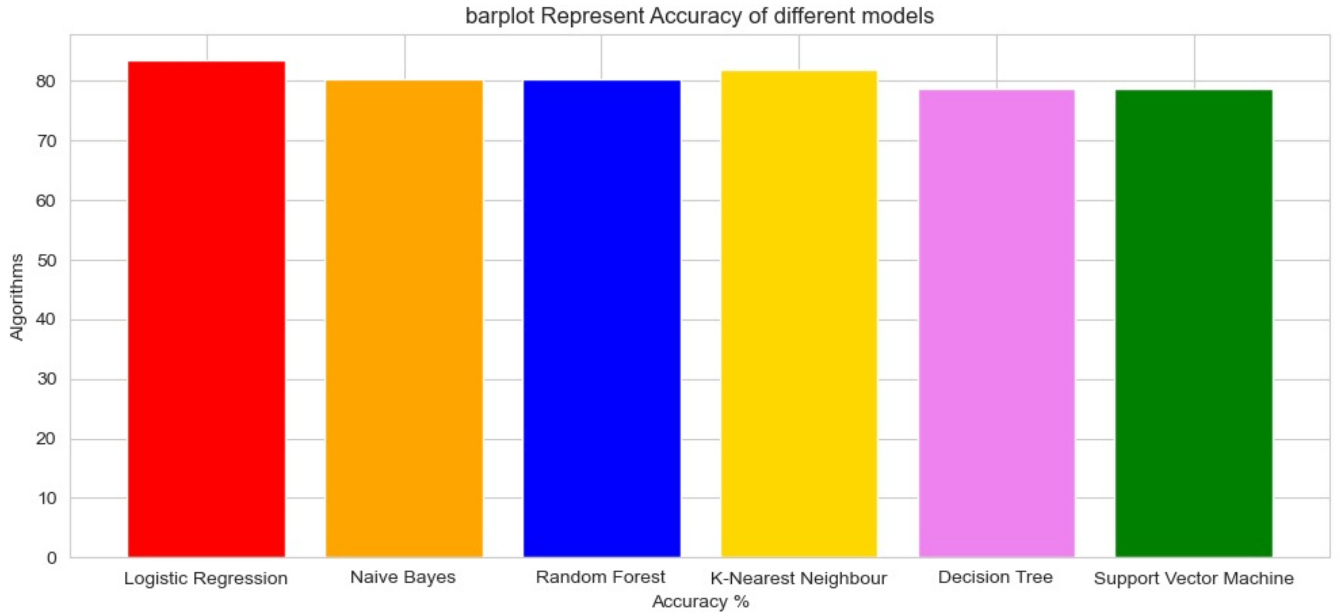
```
model_summary = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','Random Forest',
                    'K-Nearest Neighbour','Decision Tree','Support Vector Machine'], 'Accuracy': [lr_acc_score*
                    nb_acc_score*100,rf_acc_score*100,knn_acc_score*100,dt_acc_score*100,svc_acc_score*100]})
model_summary
```

Out[33]:

	Model	Accuracy
0	Logistic Regression	83.606557
1	Naive Bayes	80.327869
2	Random Forest	80.327869
3	K-Nearest Neighbour	81.967213
4	Decision Tree	78.688525
5	Support Vector Machine	78.688525

In [34]:

```
colors = ['red','orange','blue','gold','violet','green',]
plt.figure(figsize=(12,5))
plt.title("barplot Represent Accuracy of different models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_summary['Model'],model_summary['Accuracy'],color = colors)
plt.savefig('Output2.png')
plt.show()
```



We can observe that Support Vector Machine (SVM) and Decision Tree Classifier provide the most accurate results while Random Forest Classifier is the third best Machine Learning Algorithm for Heart Attack Prediction.

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js