Packages

```
#loading dataset
import pandas as pd
import numpy as np
#visualisation
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#EDA
from collections import Counter
import pandas_profiling as pp
# data preprocessing
from sklearn.preprocessing import StandardScaler
# data splitting
from sklearn.model_selection import train_test_split
# data modeling
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classification_repor
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
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
#ensembling
import six
import sys
sys.modules['sklearn.externals.six'] = six
from sklearn.ensemble import StackingClassifier
```

```
data = pd.read_csv('heart.csv')
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	th
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	

Check Missing Values

```
# get the number of missing data points per column
```

```
missvalcount = (data.isnull().sum())
print(missvalcount[missvalcount > 0])
# percent of data that is missing
t_cell = np.product(data.shape)
t_missing = missvalcount.sum()
print('Total number of our cells is:', t_cell)
print('Total number of our missing value is:', t_missing)
```

```
Series([], dtype: int64)
Total number of our cells is: 4242
Total number of our missing value is: 0
```

Data Description

Description: age - Age in Years

sex - Sex of the Patient

```
1 = Male
0 = Female
```

cp - Chest Pain Type

```
1 = Typical angina
2 = Atypical angina
3 = Non-anginal pain
```

trestbps - Resting Blood Pressure (in mm Hg on admission to the hospital)

chol - Serum Cholesterol in mg/dl

fbs - Fasting Blood sugar > 120 mg/dl

```
0 = False
1 = True
```

restecg - Resting Electrocardiographic Results

```
0 = Hypertrophy
1 = Having ST-T wave abnormality
2 = Showing probable or definite left ventricular hypertrophy by Estes' criteria
```

thalach - Maximum Heart Rate Achieved

exang - Exercise Induced Angina

```
0 = No
1 = Yes
```

oldpeak - ST Depression Induced by Exercise Relative to Rest

slope - The Slope of the Peak Exercise ST Segment

```
0 = Downsloping
1 = Flat
2 = Upsloping
```

ca - Number of Major Vessels (0-3) Colored by Flourosopy

thal - Thallium Stress Test Result

```
0 = Null
1 = Fixed defect
2 = Normal
3 = Reversible defect
```

target - Diagnosis of Heart Disease

```
0 = < 50% Diameter Narrowing (Heart Attack= No )
1 = > 50% Diameter Narrowing (Heart Attack= Yes)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

EDA

```
#!pip uninstall -y pandas-profiling
```

```
#!pip install pandas-profiling
```

```
import sys
import six
import pandas_profiling as pp
pp.ProfileReport(data)
```

Summarize dataset: 48/48 [00:04<00:00, 10.73it/s, 100% Completed]

Generate report structure: 1/1 [00:04<00:00, 4.22s/it] 100%

Render HTML: 100% 1/1 [00:00<00:00, 1.11it/s]

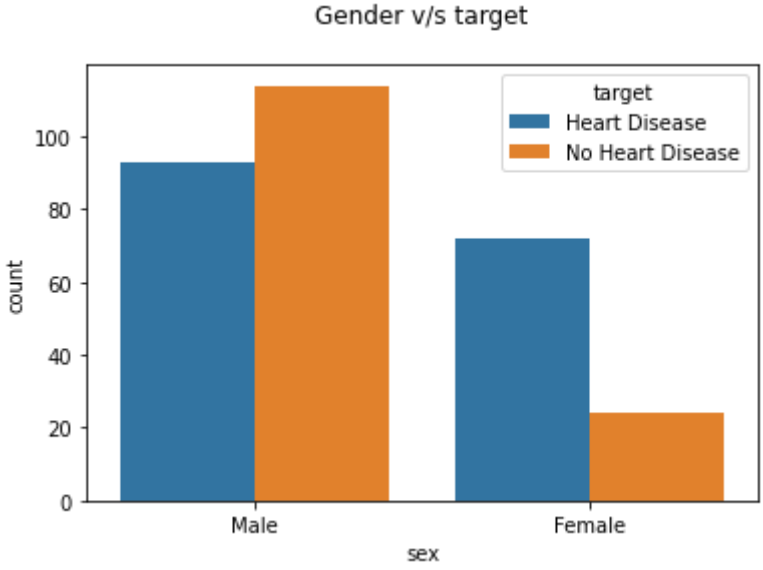
Data Visualization: Age Group & Male Female Segregation

`thal` is highly overall correlated with `target` | **High correlation** |

```
data_vis = data.copy()
def chng(sex):
    if sex == 0:
        return 'Female'
    else:
        return 'Male'
data_vis['sex'] = data_vis['sex'].apply(chng)
def chng2(prob):
    if prob == 1:
        return 'Heart Disease'
    else:
        return 'No Heart Disease'
data_vis['target'] = data['target'].apply(chng2)
```

```
sns.countplot(data= data_vis, x='sex',hue='target')
plt.title('Gender v/s target\n')
```

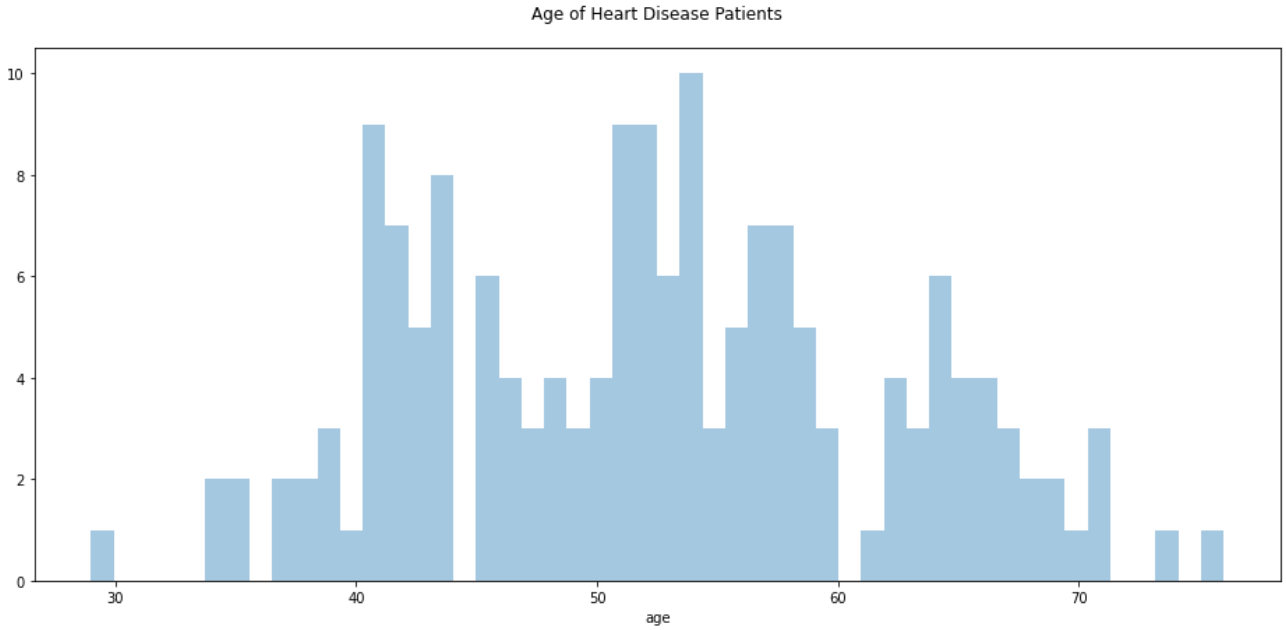
```
Text(0.5, 1.0, 'Gender v/s target\n')
```



Distinct 41

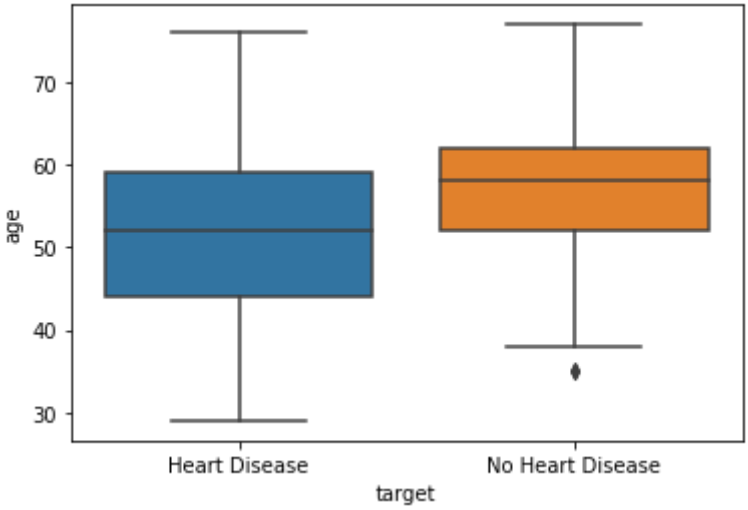
```
plt.figure(figsize=(16,7))
sns.distplot(data[data['target']==1]['age'],kde=False,bins=50)
plt.title('Age of Heart Disease Patients\n')
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
Text(0.5, 1.0, 'Age of Heart Disease Patients\n')
```




```
sns.boxplot(data=data_vis,x='target',y='age')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd547718fa0>
```



Model prepration

```
y = data["target"]
X = data.drop('target',axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 0
```

```
X_dataVis=X_test.copy();
```

```
X_datavis=X_test.copy();


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


print(y_test.unique())
Counter(y_train)
```

```
[0 1]
Counter({1: 131, 0: 111})
```

ML models

Here I take different machine learning algorithm and try to find which algorithm can predict accurately.

1. Logistic Regression
2. Naive Bayes
3. Random Forest Classifier
4. Extreme Gradient Boost
5. K-Nearest Neighbour
6. Decision Tree
7. Support Vector Machine

Logistic Regression

```
m1 = '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("confussion matrix")
print(lr_conf_matrix)
print("\n")
print("Accuracy of Logistic Regression:",lr_acc_score*100,'\n')
print(classification_report(y_test,lr_predict))
```

```
confussion matrix
[[21  6]
 [ 3 31]]


Accuracy of Logistic Regression: 85.24590163934425

              precision    recall  f1-score   support
```

```
              0       0.88      0.78      0.82        27
              1       0.84      0.91      0.87        34

       accuracy                           0.85        61
      macro avg       0.86      0.84      0.85        61
   weighted avg       0.85      0.85      0.85        61
```

Naive Bayes

```
m2 = '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("confussion matrix")
print(nb_conf_matrix)
print("\n")
print("Accuracy of Naive Bayes model:",nb_acc_score*100,'\n')
print(classification_report(y_test,nbpred))
```

```
confussion matrix
[[21  6]
 [ 3 31]]


Accuracy of Naive Bayes model: 85.24590163934425

              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.84      0.91      0.87        34

    accuracy                           0.85        61
   macro avg       0.86      0.84      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Decision Tree

```
m3 = '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("confussion matrix")
print(dt_conf_matrix)
print("\n")
```

```
print("Accuracy of DecisionTreeClassifier:",dt_acc_score*100,'\n')
print(classification_report(y_test,dt_predicted))
```

```
confussion matrix
[[23  4]
 [ 7 27]]


Accuracy of DecisionTreeClassifier: 81.9672131147541

              precision    recall  f1-score   support

           0       0.77      0.85      0.81        27
           1       0.87      0.79      0.83        34

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

Random Forest

```
m4 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=110, 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("confussion matrix")
print(rf_conf_matrix)
print("\n")
print("Accuracy of Random Forest:",rf_acc_score*100,'\n')
print(classification_report(y_test,rf_predicted))
```

```
confussion matrix
[[22  5]
 [ 3 31]]


Accuracy of Random Forest: 86.88524590163934

              precision    recall  f1-score   support

           0       0.88      0.81      0.85        27
           1       0.86      0.91      0.89        34

    accuracy                           0.87        61
   macro avg       0.87      0.86      0.87        61
weighted avg       0.87      0.87      0.87        61
```

K-Nearest Neighbors

```
m5 = '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("confussion matrix")
print(knn_conf_matrix)
print("\n")
print("Accuracy of K-NeighborsClassifier:",knn_acc_score*100,'\n')
print(classification_report(y_test,knn_predicted))
```

```
confussion matrix
[[24  3]
 [ 4 30]]


Accuracy of K-NeighborsClassifier: 88.52459016393442

              precision    recall  f1-score   support

           0       0.86      0.89      0.87        27
           1       0.91      0.88      0.90        34

    accuracy                           0.89        61
   macro avg       0.88      0.89      0.88        61
weighted avg       0.89      0.89      0.89        61
```

Support Vector Machines

```
m6 = '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("confussion 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))
```

```
confussion matrix
[[23  4]
 [ 3 31]]


Accuracy of Support Vector Classifier: 88.52459016393442

              precision    recall  f1-score   support

           0       0.88      0.85      0.87        27
           1       0.89      0.91      0.90        34
```

```
          accuracy                           0.89        61
         macro avg       0.89      0.88      0.88        61
      weighted avg       0.89      0.89      0.88        61
```

XGBoost

```
m7='Extreme Gradient Boosting'
xg = XGBClassifier()
xg.fit(X_train,y_train)
xg_predicted  = xg.predict(X_test)
xg_conf_matrix = confusion_matrix(y_test,xg_predicted)
xg_accuracy = accuracy_score(y_test,xg_predicted)
print("Confusion Matrix")
print(xg_conf_matrix)
print("\n")
print("Accuracy Score of XGBoost:" , xg_accuracy*100,"\n")
print(classification_report(y_test,xg_predicted))
```

```
Confusion Matrix
[[22  5]
 [ 4 30]]


Accuracy Score of XGBoost: 85.24590163934425

              precision    recall  f1-score   support

           0       0.85      0.81      0.83        27
           1       0.86      0.88      0.87        34

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Feature Importance (Contributing Features)

Statistical tests: SelectKBest class and chi-squared (chi2)

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
dataPred = data.copy()
X = dataPred.iloc[:,0:13]  #independent columns
y = dataPred.iloc[:,-1]    #target column
#apply SelectKBest class to extract top best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
```

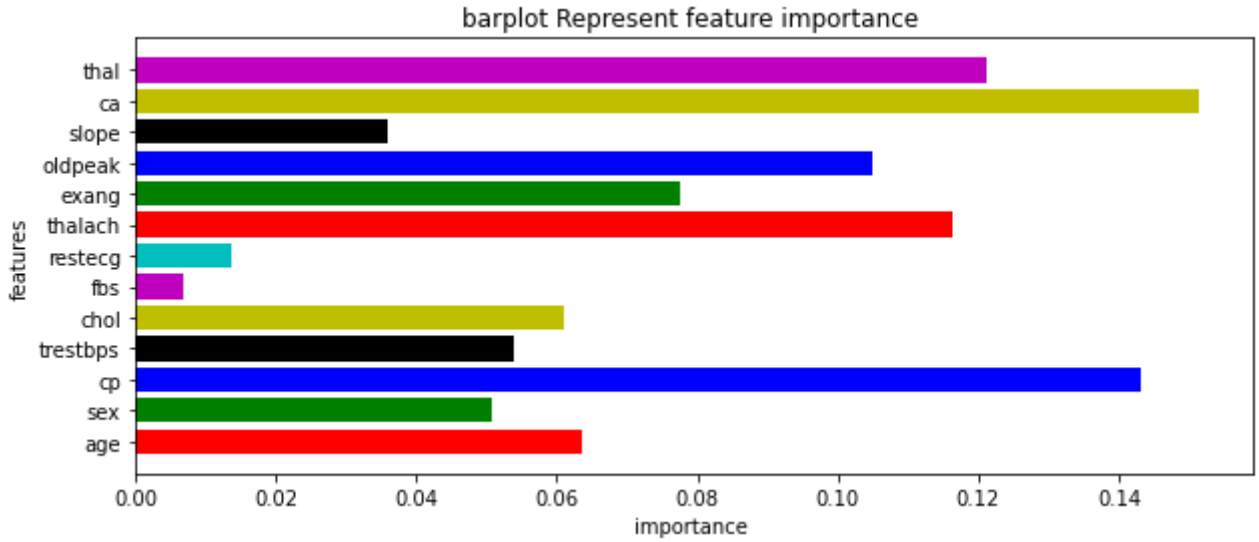
```
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(12,'Score'))  #print best features
```

```
        Specs       Score
7     thalach  188.320472
9     oldpeak   72.644253
11         ca   66.440765
2          cp   62.598098
8       exang   38.914377
4        chol   23.936394
0         age   23.286624
3    trestbps   14.823925
10      slope    9.804095
1         sex    7.576835
12       thal    5.791853
6     restecg    2.978271
```

Using Random Forest for finding feature importance

```
#Using Random Forest
imp_feature = pd.DataFrame({'Feature': ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 're
       'exang', 'oldpeak', 'slope', 'ca', 'thal'], 'Importance': rf.feature_importances_})
plt.figure(figsize=(10,4))
plt.title("barplot Represent feature importance ")
plt.xlabel("importance ")
plt.ylabel("features")
plt.barh(imp_feature['Feature'],imp_feature['Importance'],color = 'rgbkymc')
plt.show()
```

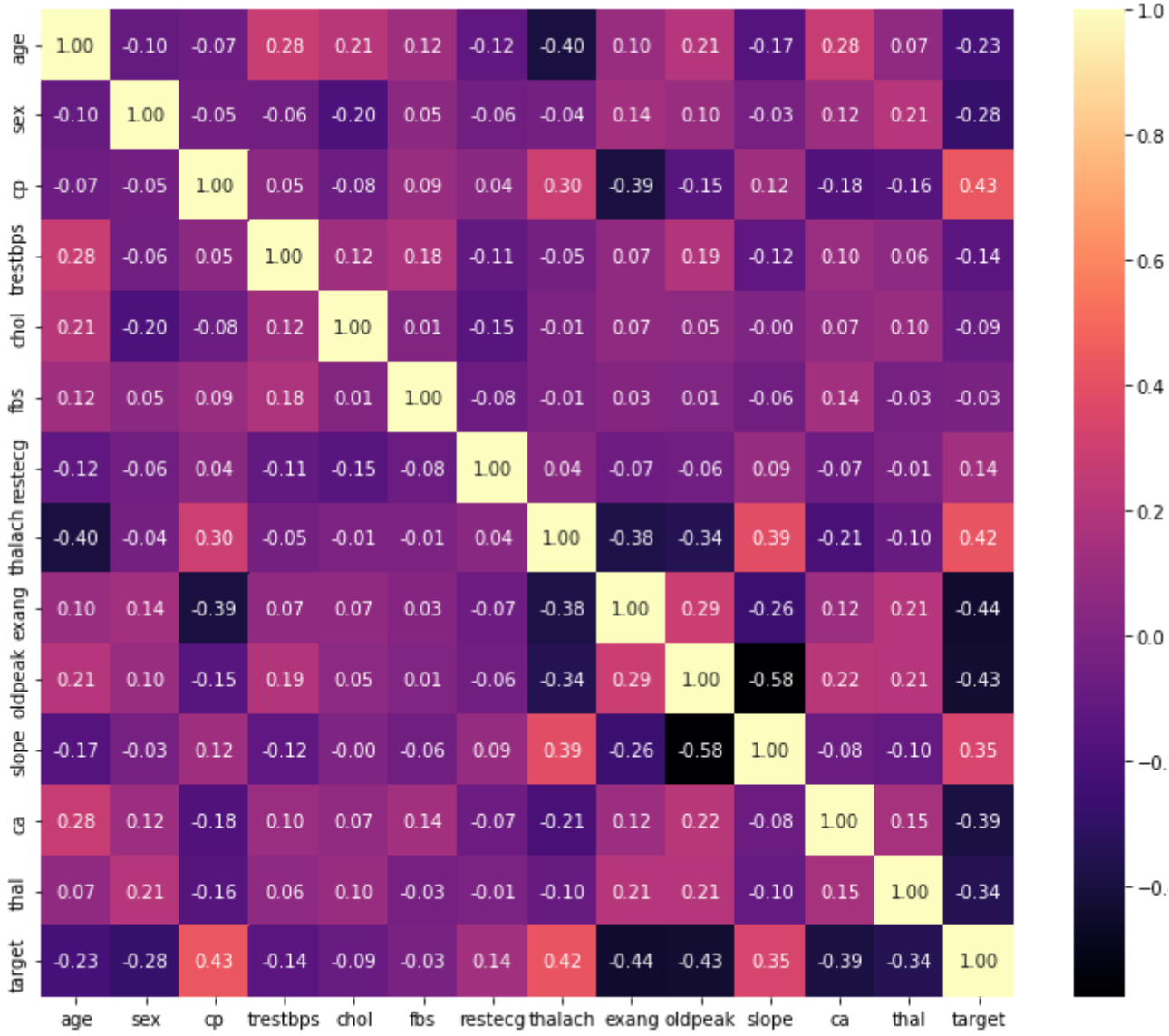
```
<ipython-input-24-45eda738f194>:8: MatplotlibDeprecationWarning: Using a string of s
  plt.barh(imp_feature['Feature'],imp_feature['Importance'],color = 'rgbkymc')
```



Using Correlation Matrix : Correlation indicates how the features are related to each other or to the target variable.

```
plt.figure(figsize=(12,10))
sns.heatmap(data.corr(),annot=True,cmap="magma",fmt='.2f')
#'cp' chest pain is highly related to the target variable
```

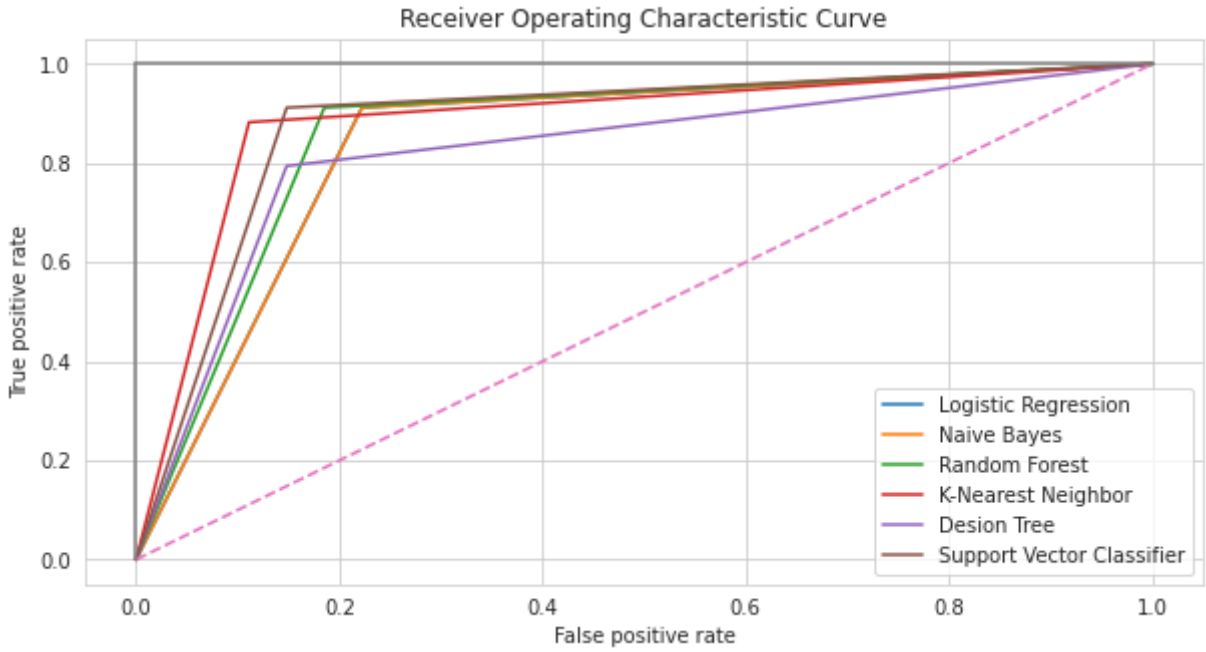
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd547646f10>
```




```
#Plotting ROC curves
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_predic
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_predic
```

```
sns.set_style('whitegrid')
plt.figure(figsize=(10,5))
plt.title('Receiver Operating Characteristic 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.show()
```



Model Evaluation

Cross-Validation

```
from sklearn import model_selection
print('10-fold cross validation:\n')

for clf, label in zip([rf, svc,knn, nb, scv],
                      ['Random Forest',
                       'Support Vector Machines',
                       'KNN',
                       'Naive Bayes',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, X, y,
                                             cv=10, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
 10-fold cross validation:

 Accuracy: 0.84 (+/- 0.05) [Random Forest]
 Accuracy: 0.67 (+/- 0.08) [Support Vector Machines]
```

```
Accuracy: 0.64 (+/- 0.09) [KNN]
Accuracy: 0.81 (+/- 0.07) [Naive Bayes]
Accuracy: 0.84 (+/- 0.05) [StackingClassifier]
```

Ensembling

> **Let's try stacking to see if we can increase accuracy.**

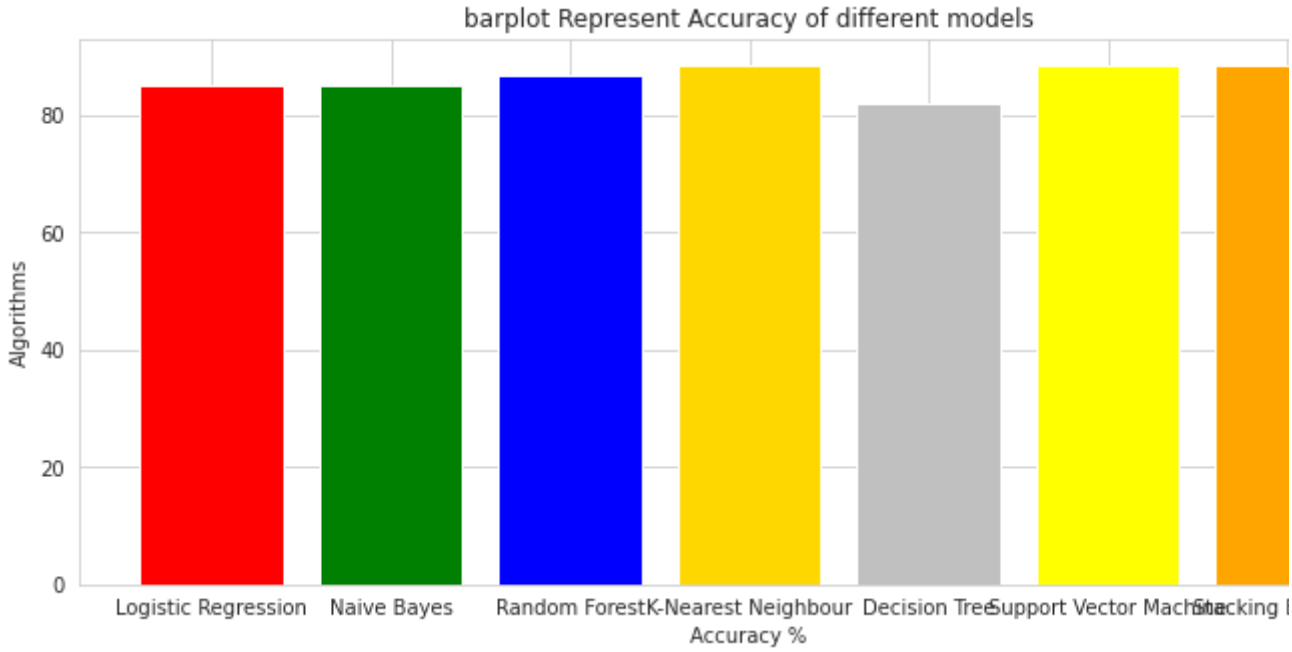
```
estimators = [
     ('rf', rf),
     ('svc', svc),
     ('knn',knn),
     ('nb',nb),

]
scv=StackingClassifier(estimators=estimators)
scv.fit(X_train,y_train)
scv_predicted = scv.predict(X_test)
scv_conf_matrix = confusion_matrix(y_test,scv_predicted)
scv_acc_score = accuracy_score(y_test,scv_predicted)
print("Confusion Matrix""\n")
print(scv_conf_matrix)
print("\n")
print("Accuracy Score for Ensemble stacking:", scv_accuracy,"\n")
print(classification_report(y_test,scv_predicted))
```

```
Confusion Matrix

[[22  5]
 [ 2 32]]


Accuracy Score for Ensemble stacking: 0.8852459016393442

              precision    recall  f1-score   support

           0       0.92      0.81      0.86        27
           1       0.86      0.94      0.90        34

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.88        61
```

```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','Random Forest',
                   'K-Nearest Neighbour','Decision Tree','Support Vector Machine','Stacki
                   nb_acc_score*100,rf_acc_score*100,knn_acc_score*100,dt_acc_score*100,s
model_ev
```

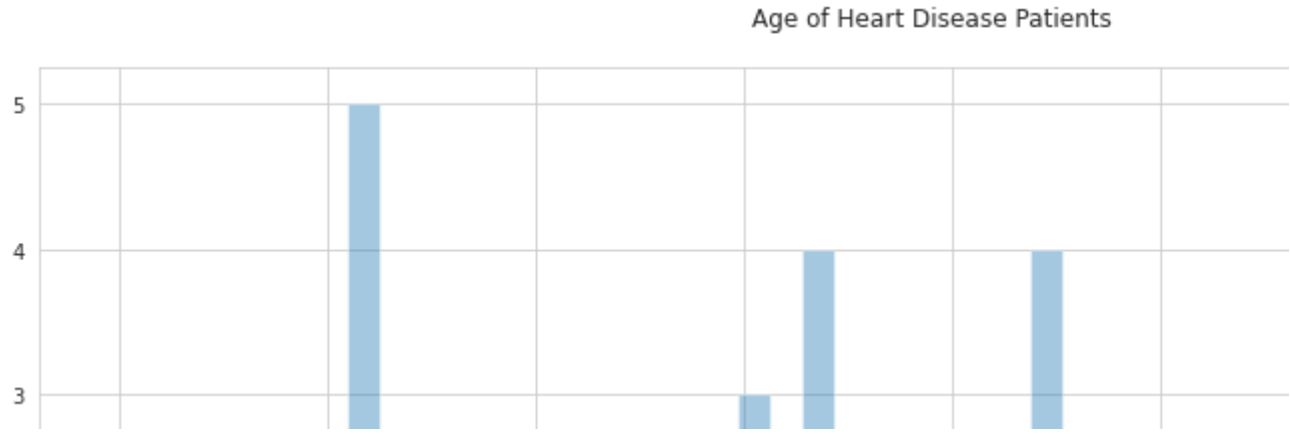
	Model	Accuracy
0	Logistic Regression	85.245902
1	Naive Bayes	85.245902
2	Random Forest	86.885246
3	K-Nearest Neighbour	88.524590
4	Decision Tree	81.967213

```
colors = ['red','green','blue','gold','silver','yellow','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()
```




```
plt.figure(figsize=(16,7))
sns.distplot(X_dataVis[scv_predicted[:]==1]['age'],kde=False,bins=50)
plt.title('Age of Heart Disease Patients\n')
```

```
Text(0.5, 1.0, 'Age of Heart Disease Patients\n')
```



Conclusion

1) SVM and KNN give the best Accuracy compared to other models.

2) thalach, oldpeak, number of blood vessels, Chest pain are major factors of heart attack.

3) Ensembling technique has no change in the model accuracy.