Heart disease detection

March 26, 2021

1 Importing Libraries

[1]:
```
import warnings
warnings.filterwarnings('ignore')

#data wrangling and pre-processing
import pandas as pd
import numpy as np
#data visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split

#model validation
from sklearn.metrics import␣
↪log_loss,roc_auc_score,precision_score,f1_score,recall_score,roc_curve,auc
from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score,fbeta_score,matthews_corrcoef
from sklearn import metrics

#cross validation
from sklearn.model_selection import StratifiedKFold

#machinelearning algos
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import␣
↪RandomForestClassifier,VotingClassifier,AdaBoostClassifier,GradientBoostingClassifier,Extra
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
#import XGBoost as xgb
from scipy import stats
```

[2]:
```
df = pd.read_csv("heartsdata.csv")
```

[3]:
```
df.head()
```

```
[3]:    age  sex  chest pain type  resting bp s  cholesterol  fasting blood sugar  \
     0   40    1                2           140          289                    0
     1   49    0                3           160          180                    0
     2   37    1                2           130          283                    0
     3   48    0                4           138          214                    0
     4   54    1                3           150          195                    0

        resting ecg  max heart rate  exercise angina  oldpeak  ST slope  target
     0            0             172                0      0.0         1       0
     1            0             156                0      1.0         2       1
     2            1              98                0      0.0         1       0
     3            0             108                1      1.5         2       1
     4            0             122                0      0.0         1       0
```

2 Data Cleansing & Pre-processing

First we change the col names for easy understanding and then encode the features into categorial variables

```
[4]: #renaming to propername
df.columns = ['age' , 'sex' , 'chestpaintype' , 'resting_blood_pressure' ,␣
↪'cholesterol' , 'fasting_blood_sugar' , 'rest_ecg' ,␣
↪'max_heartrate_achieved' , 'exercise_induced_angina' , 'st_depression' ,␣
↪'st_slope' , 'target']
```

```
[5]: #converting to categorial features
df['chestpaintype'][df['chestpaintype'] == 1] = 'typical angina'
df['chestpaintype'][df['chestpaintype'] == 2] = 'atypical angina'
df['chestpaintype'][df['chestpaintype'] == 3] = 'non-anginal pain'
df['chestpaintype'][df['chestpaintype'] == 4] = 'asymptomatic'

df['rest_ecg'][df['rest_ecg'] == 0] = 'normal'
df['rest_ecg'][df['rest_ecg'] == 1] = 'ST-T wave abnormality'
df['rest_ecg'][df['rest_ecg'] == 2] = 'left venticular hypertrophy'

df['st_slope'][df['st_slope'] == 1] = 'upsloping'
df['st_slope'][df['st_slope'] == 2] ='flat'
df['st_slope'][df['st_slope'] == 3] = 'downsloping'

df["sex"] = df.sex.apply(lambda x:'male' if x==1 else 'female')
```

```
[6]: df['chestpaintype'].value_counts() #to get total number for each chestpain type
```

```
[6]: asymptomatic        625
     non-anginal pain    283
     atypical angina     216
     typical angina       66
```

```
Name: chestpaintype, dtype: int64
```

[7]:
```
df['rest_ecg'].value_counts()
```

[7]:
```
normal                         684
left venticular hypertrophy    325
ST-T wave abnormality          181
Name: rest_ecg, dtype: int64
```

[8]:
```
df['st_slope'].value_counts()
```

[8]:
```
flat           582
upsloping      526
downsloping     81
0                1
Name: st_slope, dtype: int64
```

[9]:
```
#dropping row with st_slope=0
df.drop(df[df.st_slope == 0].index, inplace=True)
#checking distribution after dropping
df['st_slope'].value_counts()
```

[9]:
```
flat           582
upsloping      526
downsloping     81
Name: st_slope, dtype: int64
```

[10]:
```
df.head()
```

[10]:
```
   age     sex     chestpaintype  resting_blood_pressure  cholesterol  \
0   40    male   atypical angina                     140          289
1   49  female  non-anginal pain                     160          180
2   37    male   atypical angina                     130          283
3   48  female      asymptomatic                     138          214
4   54    male  non-anginal pain                     150          195

   fasting_blood_sugar               rest_ecg  max_heartrate_achieved  \
0                    0                 normal                     172
1                    0                 normal                     156
2                    0  ST-T wave abnormality                      98
3                    0                 normal                     108
4                    0                 normal                     122

   exercise_induced_angina  st_depression   st_slope  target
0                        0            0.0  upsloping       0
1                        0            1.0       flat       1
2                        0            0.0  upsloping       0
3                        1            1.5       flat       1
```

```
4                        0            0.0  upsloping       0
```

Now we see that the features are successfully converted to respective categories.

[11]:
```
#checking missing entries(null) in dataset colwise
df.isna().sum()  #0:no missing entries
```

[11]:
```
age                        0
sex                        0
chestpaintype              0
resting_blood_pressure     0
cholesterol                0
fasting_blood_sugar        0
rest_ecg                   0
max_heartrate_achieved     0
exercise_induced_angina    0
st_depression              0
st_slope                   0
target                     0
dtype: int64
```

3 Exploratory Data Analysis(EDA)

[12]:
```
df.shape #to get no.of rows and cols
```

[12]:
```
(1189, 12)
```

[13]:
```
#summary statistics of numerical cols
df.describe(include =[np.number]) #fbs ranges from 0 to 1
```

[13]:
```
               age  resting_blood_pressure  cholesterol  fasting_blood_sugar  \
count  1189.000000             1189.000000  1189.000000          1189.000000
mean     53.708158              132.138772   210.376787             0.212784
std       9.352961               18.369251   101.462185             0.409448
min      28.000000                0.000000     0.000000             0.000000
25%      47.000000              120.000000   188.000000             0.000000
50%      54.000000              130.000000   229.000000             0.000000
75%      60.000000              140.000000   270.000000             0.000000
max      77.000000              200.000000   603.000000             1.000000

       max_heartrate_achieved  exercise_induced_angina  st_depression  \
count             1189.000000              1189.000000    1189.000000
mean               139.739277                 0.387721       0.923549
std                 25.527386                 0.487435       1.086464
min                 60.000000                 0.000000      -2.600000
25%                121.000000                 0.000000       0.000000
50%                141.000000                 0.000000       0.600000
```

```
75%                 160.000000                   1.000000       1.600000
max                 202.000000                   1.000000       6.200000

            target
count  1189.000000
mean      0.528175
std       0.499416
min       0.000000
25%       0.000000
50%       1.000000
75%       1.000000
max       1.000000
```

From aboe we see that resting_blood_pressure and cholestrol have some outliers(extreme vals) as they have min val of 0. Cholestrol has outlier on upper side also having maximum val of 603

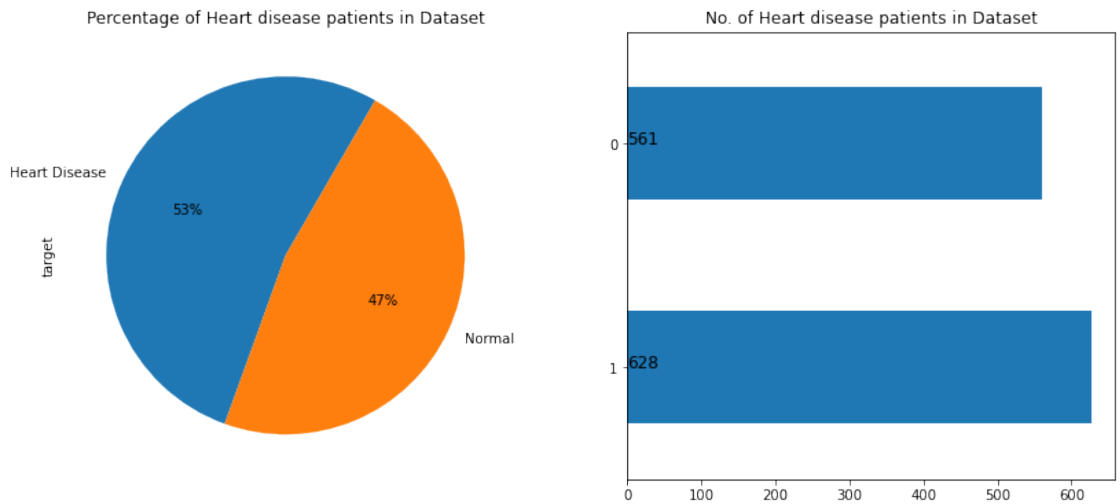
[14]:
```
#summary statistics of categorial cols
df.describe(include =[np.object])
#top gives highest occuring value
```

[14]:
```
         sex chestpaintype rest_ecg st_slope
count   1189          1189     1189     1189
unique     2             4        3        3
top     male  asymptomatic   normal     flat
freq     908           625      683      582
```

3.1 Distribution of Heart disease

Target variable

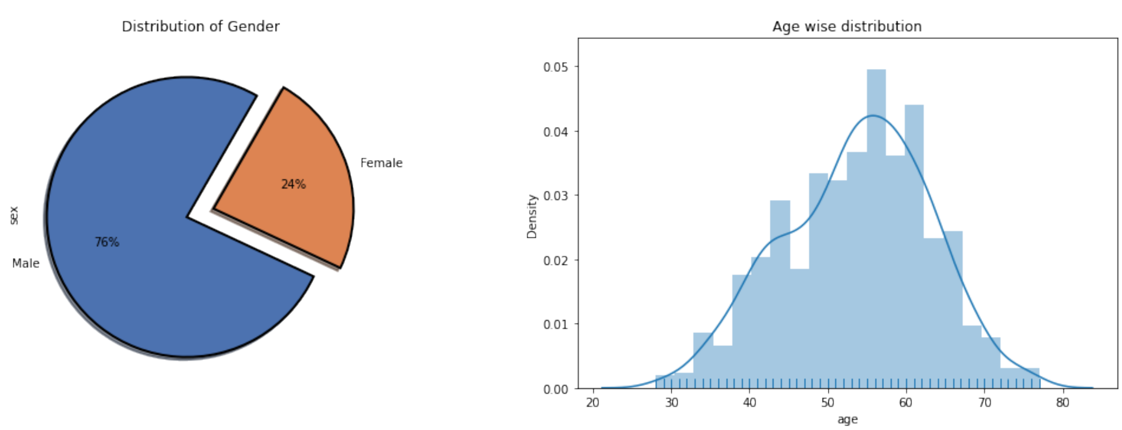
[15]:
```
# Plotting attrition of employees. to check if the data is balanced or not
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey=False, figsize=(14,6))

ax1 = df['target'].value_counts().plot.pie( x="Heart disease" ,y ='no.of␣
↪patients',
                   autopct = "%1.0f%%",labels=["Heart Disease","Normal"],␣
↪startangle = 60,ax=ax1);
ax1.set(title = 'Percentage of Heart disease patients in Dataset')

ax2 = df["target"].value_counts().plot(kind="barh" ,ax =ax2)
for i,j in enumerate(df["target"].value_counts().values):
    ax2.text(.5,i,j,fontsize=12)
ax2.set(title = 'No. of Heart disease patients in Dataset')
plt.show()
```



The dataset is balanced with 628 heart disease patients and 561 normal patients

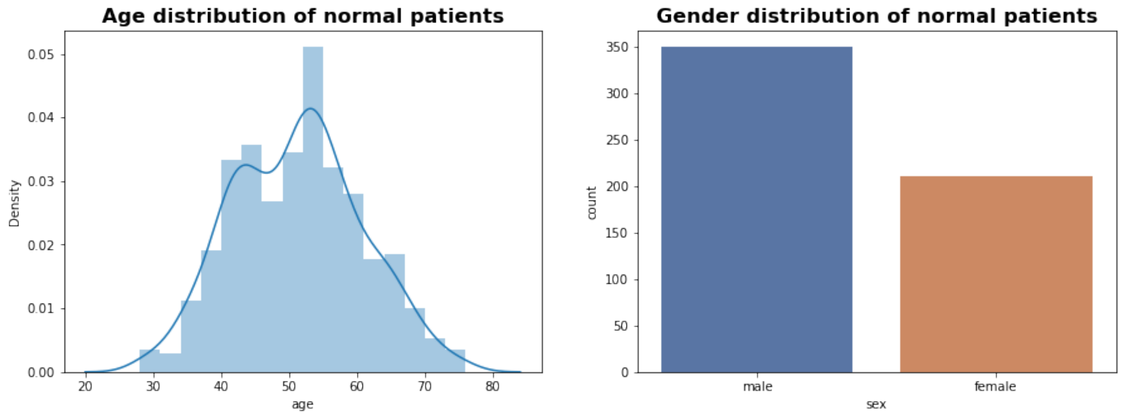
3.2 Gender & agewise distribution

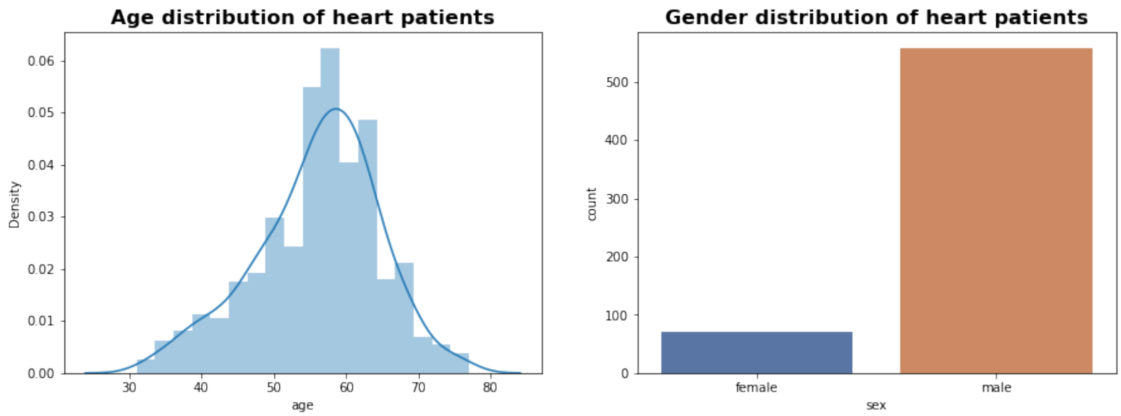
[16]:
```
plt.figure(figsize=(18,12))
plt.subplot(221)
df["sex"].value_counts().plot.pie(autopct = "%1.0f%%",colors = sns.
 ↪color_palette("deep",5),startangle = 60,labels=["Male","Female"],
wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.1,.1],shadow =True)
plt.title("Distribution of Gender")
plt.subplot(222)
ax= sns.distplot(df['age'], rug=True)
plt.title("Age wise distribution")
plt.show()
```



The percentage of males suffering from heart disease is greater than females and the average age of the patients is 55-57

[17]:
```
#we create a seperate df for normal and heart patients

att_1=df[df['target']==1]
att_0=df[df['target']==0]

#plot of normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.distplot(att_0['age'])
plt.title('Age distribution of normal patients', fontsize=16, weight='bold')

ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(att_0['sex'], palette='deep')
plt.title('Gender distribution of normal patients', fontsize=16, weight='bold')
plt.show()

#plot of heart patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.distplot(att_1['age'])
plt.title('Age distribution of heart patients', fontsize=16, weight='bold')

ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(att_1['sex'], palette='deep')
plt.title('Gender distribution of heart patients', fontsize=16, weight='bold')
plt.show()
```

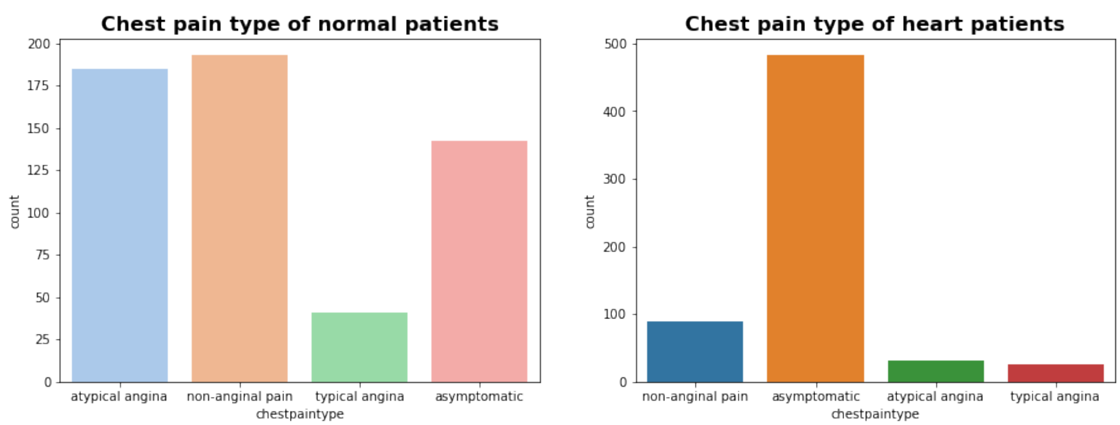




From the above plot we see that number of male heartpatients is high and the mean age is between 57-60 yrs

3.3 Distribution of Chest pain type

[18]:
```
#plt for normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.countplot(att_0['chestpaintype'], palette='pastel')
plt.title('Chest pain type of normal patients', fontsize=16, weight='bold')
#patients with chestpain type non-anginal pain and atypical angina are not muvh prone to heart disease.
#plt for heart patients
ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(att_1['chestpaintype'])
plt.title('Chest pain type of heart patients', fontsize=16, weight='bold')
#asymptomatic pain is dangerous.
plt.show()
```



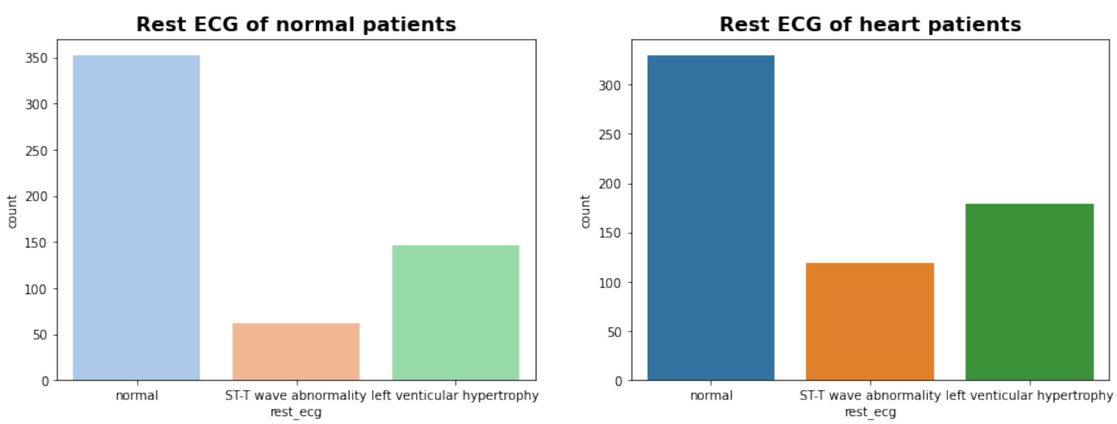
[19]:
```
#exploring heart disease patients based on chest pain type in terms of␣
 ↪percentage
plot_criteria= ['chestpaintype', 'target']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]],␣
 ↪normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

[19]: <pandas.io.formats.style.Styler at 0x2154861c10>

We find that almost 77% of chest pain type patients suffer from is asymptomatic chest pain which is also known as silent myocardial infarction (SMI). The symptoms are mild compared to actual heart attack so it is described as silent killer.

3.4 Distribution of Rest ECG

[20]:
```
#plot of normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.countplot(att_0['rest_ecg'], palette='pastel')
plt.title('Rest ECG of normal patients', fontsize=16, weight='bold')

#plot of heart patients
ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(att_1['rest_ecg'])
plt.title('Rest ECG of heart patients', fontsize=16, weight='bold')
plt.show()
```



ST-T wave abnormality and leftventicular hypertrophy is higher in heart patients compared to normal patients.

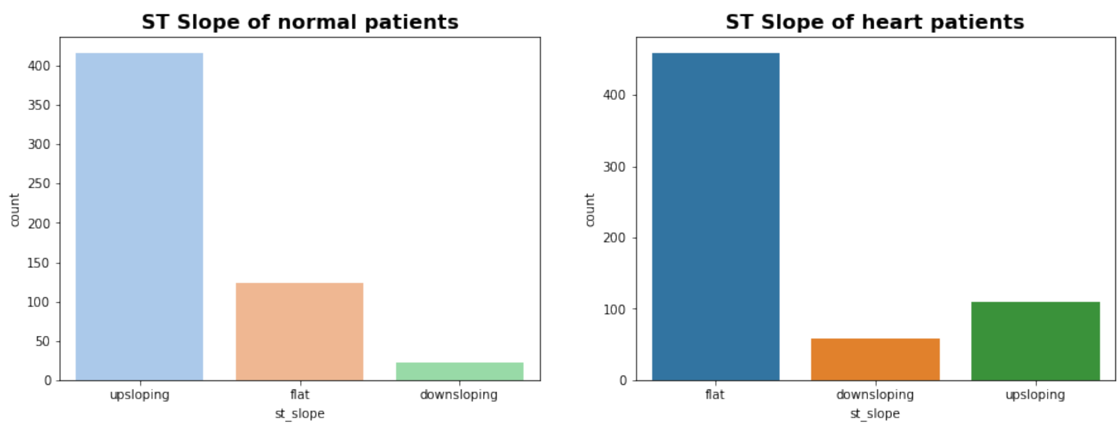
[21]:
```
#exploring heartpatients based on rest ecg
plot_criteria= ['rest_ecg', 'target']
cm = sns.light_palette("seagreen", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]],␣
↪normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

[21]: <pandas.io.formats.style.Styler at 0x2155728ee0>

ECG measures heartrate and rhythm, but doesn't show blockage in the arteries. So, around 52% of heartpatients have normal ECG.

3.5 Distribution of ST Slope

[22]:
```
#plot for normal patients
fig = plt.figure(figsize=(15,5))
ax1 = plt.subplot2grid((1,2),(0,0))
sns.countplot(att_0['st_slope'], palette='pastel')
plt.title('ST Slope of normal patients', fontsize=16, weight='bold')

#plt fro heart patients
ax1 = plt.subplot2grid((1,2),(0,1))
sns.countplot(att_1['st_slope'])
plt.title('ST Slope of heart patients', fontsize=16, weight='bold')
plt.show()
```



In case normal patients if stslope is measured after exercise it should be upsloping. But, inacse of heart patients flat is very high compared to normal patients

[23]:
```
#exploring heart patients based on at_slope
plot_criteria= ['st_slope', 'target']
cm = sns.light_palette("seagreen", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]],␣
↪normalize='columns') * 100,2)).style.background_gradient(cmap = cm)
```

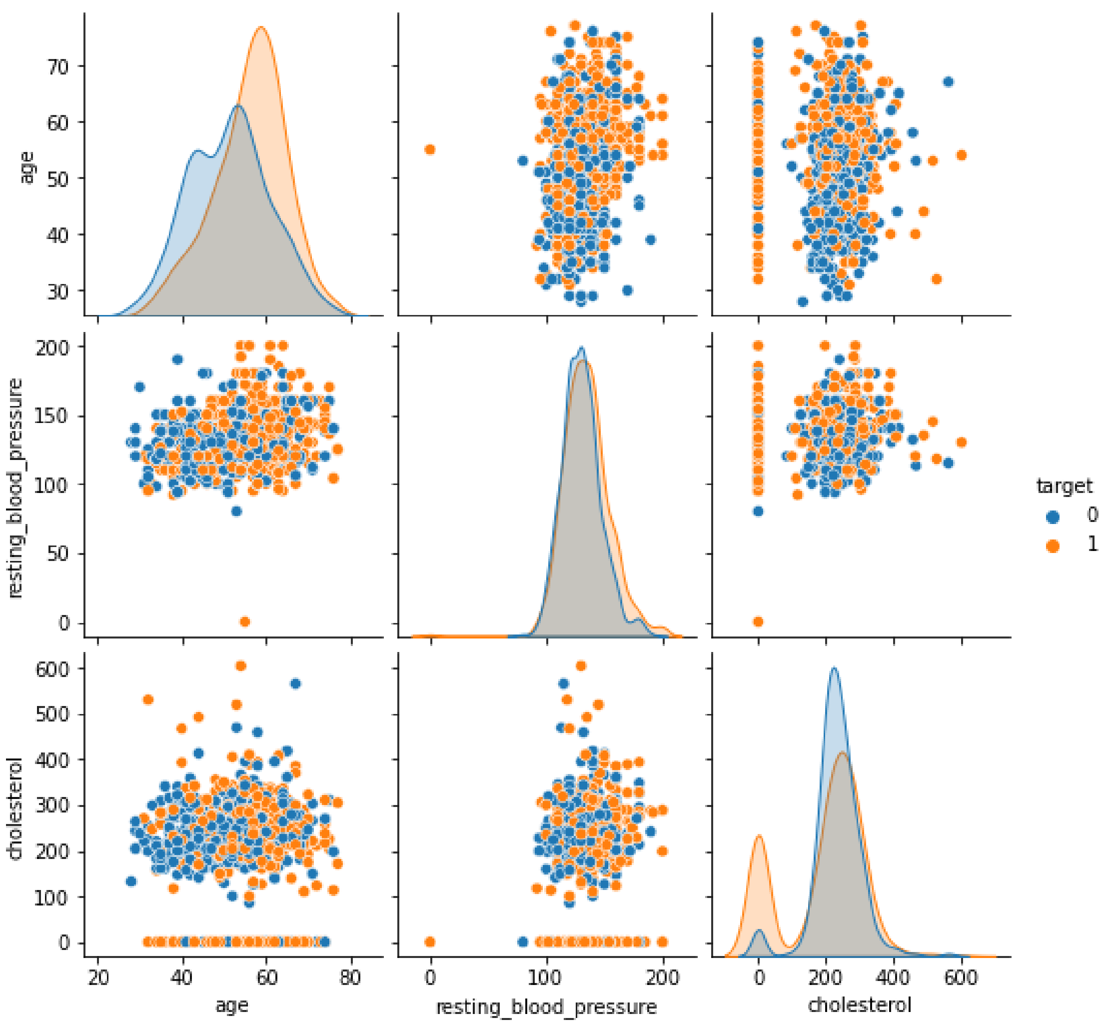
[23]: <pandas.io.formats.style.Styler at 0x215574ea60>

The above plot indicates that upslopping is a +ve sign as 74% of normal patients have upsloping while we see 73% of heart patients with flat sloping. (It is said that ST Slope is more accurate ECG to diagonise significant CAD i.e., coronary artery disease.

3.6 Distribution of numerical features

[24]:
```
sns.pairplot(df, hue = 'target', vars = ['age', 'resting_blood_pressure',␣
↪'cholesterol'])
```

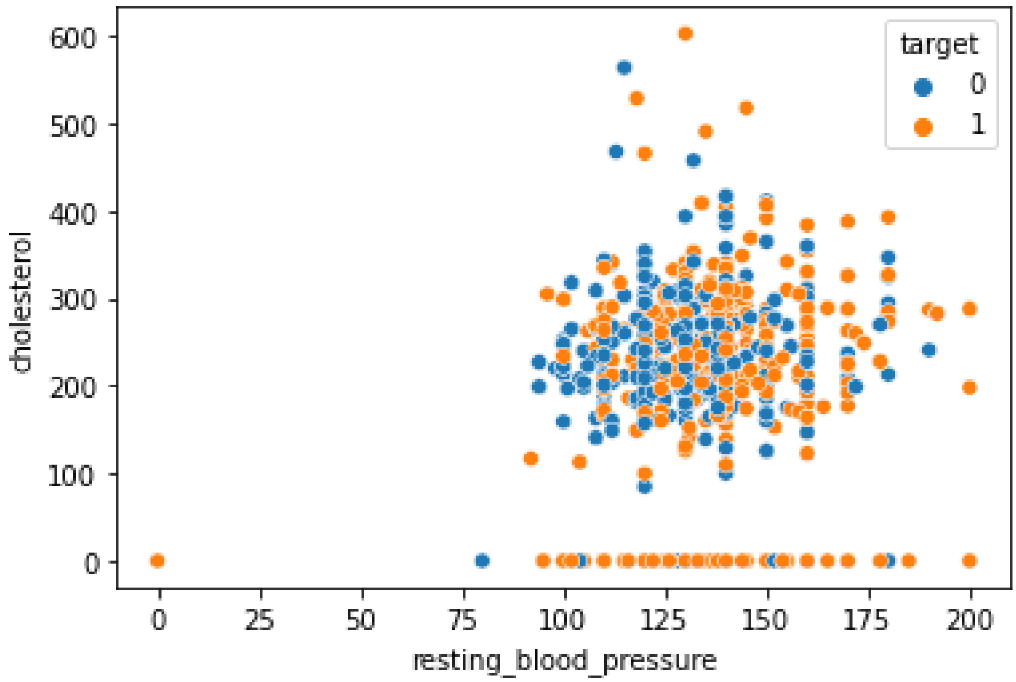
[24]: <seaborn.axisgrid.PairGrid at 0x2153fb5df0>



The plot indicates that the chance suffering heart disease is proportional to age i.e., increases with increase in the age.

[25]:
```
sns.scatterplot(x= 'resting_blood_pressure', y = 'cholesterol' , hue = 'target' , data = df)
```

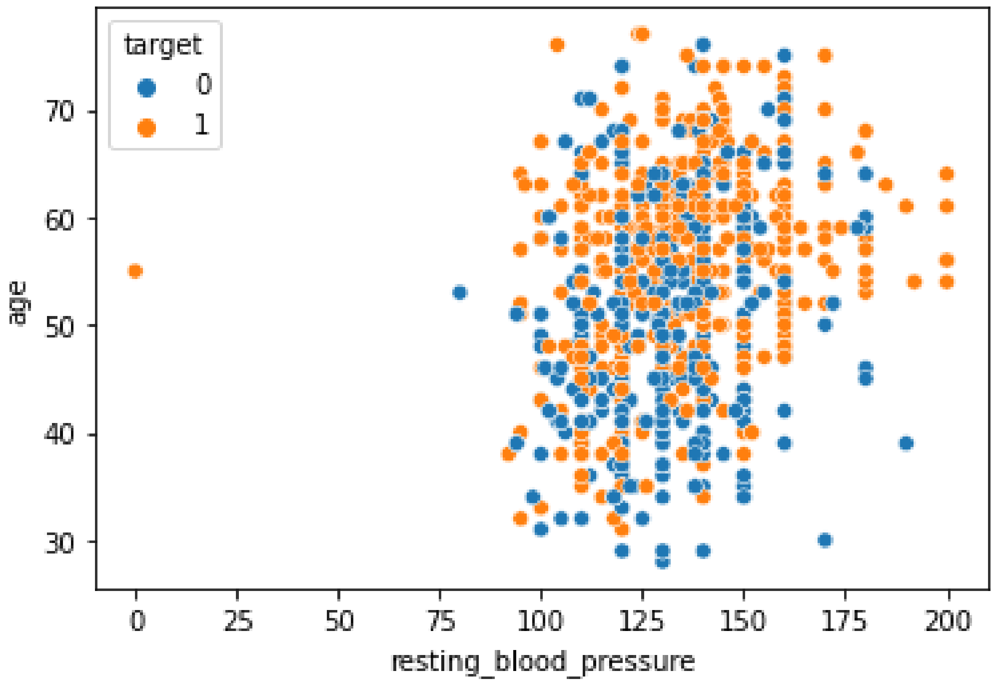
[25]:
```
<AxesSubplot:xlabel='resting_blood_pressure', ylabel='cholesterol'>
```



The plot shows the outliers in cholesterol and bp with some patients having cholesterol as zero and one patient with both cholesterol and bp as 0.

[26]:
```
sns.scatterplot(x= 'resting_blood_pressure' , y = 'age' , hue = 'target' , data= df)
```

[26]:
```
<AxesSubplot:xlabel='resting_blood_pressure', ylabel='age'>
```



The plot shows that if a person is having bp above 175 and age is above 50 the person is more likely to suffer from heart disease.

4 Outlier detection & removal

4.0.1 Detection using z-score

Outlier is extreme value it can be either large or small value in comparision with the remaining data. These may occur when there is error in data entry or it may be genuine data also. Z=(score-mean)/standard deviation

[27]:
```
#filtering the numeric features having outliers as per EDA

df_numeric =␣
↪df[['age','resting_blood_pressure','cholesterol','max_heartrate_achieved']]
```

[28]:
```
df_numeric.head()
```

[28]:

	age	resting_blood_pressure	cholesterol	max_heartrate_achieved
0	40	140	289	172
1	49	160	180	156
2	37	130	283	98
3	48	138	214	108
4	54	150	195	122

[29]:
```
#calculating the z-score for numeric columns
z = np.abs(stats.zscore(df_numeric))
print(z)
```

```
[[1.46626567 0.4281359  0.7752277  1.26430092]
 [0.5035987  1.51737007 0.29951621 0.63725935]
 [1.78715466 0.11648118 0.71606748 1.63576637]
 …
 [0.35210527 0.11648118 0.78265797 0.96953469]
 [0.35210527 0.11648118 0.2526458  1.34268112]
 [1.68019167 0.31921249 0.34881639 1.30349102]]
```

[30]:
```
#since, it is difficult to identify which points are outliers from the above␣
↪ouput.
#Defining thershold for filtering outliers.
threshold = 3
print(np.where(z > 3))
```

```
(array([  30,   76,  109,  149,  242,  366,  371,  391,  400,  450,  592,
        617,  733,  760, 1012, 1038, 1074], dtype=int64), array([2, 2, 1, 2, 1,
1, 3, 3, 1, 1, 1, 2, 1, 1, 1, 2, 1], dtype=int64))
```

The above data indicate which cell in data has z-score>3. The first array gives rownum and second arr gives resp col num i.e., z[30][2] has z-score > 3. So, we have 17 data points as outliers.

[31]:
```
#filtering outliers so we have only data points which are below thershold
df = df[(z < 3).all(axis=1)]
```

[32]:
```
df.shape
```

[32]:
```
(1172, 12)
```

We encode categorial variables as dummy variables and segregate features and target variable before splitting to train and test data.

[33]:
```
# encoding categorial variables
df = pd.get_dummies(df, drop_first=True)
#we use pandas get_dummies() to encode all the variables at a time instead of␣
↪dng them seperately.
df.head()
```

[33]:
```
   age  resting_blood_pressure  cholesterol  fasting_blood_sugar  \
0   40                     140          289                    0
1   49                     160          180                    0
2   37                     130          283                    0
3   48                     138          214                    0
4   54                     150          195                    0

   max_heartrate_achieved  exercise_induced_angina  st_depression  target  \
```

```
0                      172                        0           0.0        0
1                      156                        0           1.0        1
2                       98                        0           0.0        0
3                      108                        1           1.5        1
4                      122                        0           0.0        0

   sex_male  chestpaintype_atypical angina  chestpaintype_non-anginal pain  \
0         1                             1                               0
1         0                             0                               1
2         1                             1                               0
3         0                             0                               0
4         1                             0                               1

   chestpaintype_typical angina  rest_ecg_left venticular hypertrophy  \
0                             0                                     0
1                             0                                     0
2                             0                                     0
3                             0                                     0
4                             0                                     0

   rest_ecg_normal  st_slope_flat  st_slope_upsloping
0                1              0                   1
1                1              1                   0
2                0              0                   1
3                1              1                   0
4                1              0                   1
```

[34]:
```
df.shape
```

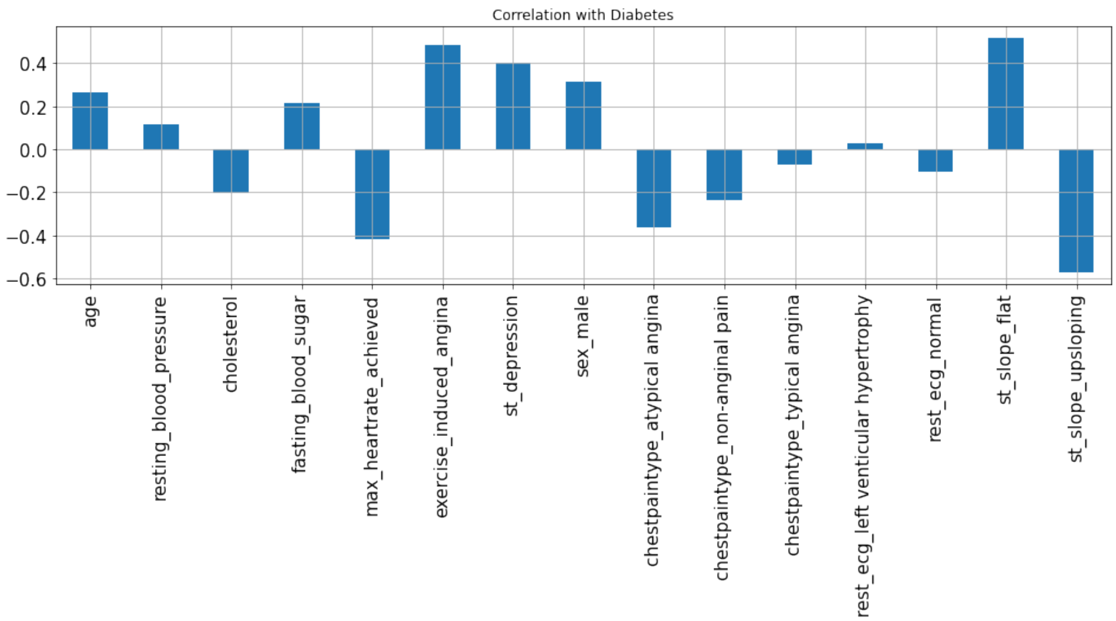
[34]:
```
(1172, 16)
```

[35]:
```
#seperating df into features (X) and target variable(y)
X = df.drop(['target'],axis=1)
y = df['target']
```

4.1 Checking correlation

[36]:
```
X.corrwith(y).plot.bar(figsize=(16,4), title = 'Correlation with Diabetes' ,␣
↪fontsize = 15, rot = 90 , grid = True)
```

[36]:
```
<AxesSubplot:title={'center':'Correlation with Diabetes'}>
```



5 Train-Test-Split

[37]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
↪test_size=0.2,shuffle=True, random_state=5)
```

[38]:
```
#checking the distribution of target var on train test split
print('Distribution of target variable in training set')
print(y_train.value_counts())

print('Distribution of target variable in test set')
print(y_test.value_counts())
```

```
Distribution of target variable in training set
1    491
0    446
Name: target, dtype: int64
Distribution of target variable in test set
1    123
0    112
Name: target, dtype: int64
```

[39]:
```
print('---Traning set shape---')
print(X_train.shape)
print(y_train.shape)

print('---Test set shape---')
```

```
print(X_test.shape)
print(y_test.shape)
```

```
---Traning set shape---
(937, 15)
(937,)
---Test set shape---
(235, 15)
(235,)
```

5.0.1 Feature normalization

Here we normalize all the numeric features in the range of 0 to 1

[40]:
```
from sklearn.preprocessing import MinMaxScaler #we use this scaler bcoz some of␣
↪our variables are in range of 0 to 1.
scaler = MinMaxScaler()
X_train[['age','resting_blood_pressure','cholesterol','max_heartrate_achieved','st_depression'
↪= scaler.
↪fit_transform(X_train[['age','resting_blood_pressure','cholesterol','max_heartrate_achieved
X_train.head()
#scale features after train test split bcoz it may cause data leakage.
```

[40]:
```
          age  resting_blood_pressure  cholesterol  fasting_blood_sugar  \
478  0.673469                0.193548     0.000000                    1
253  0.673469                0.354839     0.594705                    0
273  0.551020                0.516129     0.409369                    0
111  0.591837                0.623656     0.519348                    0
50   0.448980                0.408602     0.474542                    0

     max_heartrate_achieved  exercise_induced_angina  st_depression  sex_male  \
478                0.303704                        1       0.454545         1
253                0.355556                        1       0.194805         1
273                0.466667                        1       0.584416         1
111                0.185185                        1       0.584416         1
50                 0.400000                        1       0.454545         1

     chestpaintype_atypical angina  chestpaintype_non-anginal pain  \
478                              0                               0
253                              0                               0
273                              0                               0
111                              0                               0
50                               0                               0

     chestpaintype_typical angina  rest_ecg_left venticular hypertrophy  \
478                             0                                     0
253                             0                                     0
273                             0                                     0
```

```
111                          0                                    0
50                           0                                    0

     rest_ecg_normal  st_slope_flat  st_slope_upsloping
478                1              0                   0
253                0              0                   1
273                1              1                   0
111                1              1                   0
50                 1              1                   0
```

[41]:
```
X_test[['age','resting_blood_pressure','cholesterol','max_heartrate_achieved','st_depression']]
↪= scaler.
↪transform(X_test[['age','resting_blood_pressure','cholesterol','max_heartrate_achieved','st
X_test.head()
#we only do transform but not fit and transform on test set bcoz it learns the␣
↪pattern from previous train set and performs on test set
```

[41]:
```
           age  resting_blood_pressure  cholesterol  fasting_blood_sugar  \
1024  0.693878                0.301075     0.572301                    0
182   0.469388                0.408602     0.456212                    0
785   0.346939                0.494624     0.480652                    0
924   0.591837                0.623656     0.562118                    0
780   0.612245                0.387097     0.527495                    0

      max_heartrate_achieved  exercise_induced_angina  st_depression  \
1024                0.266667                        0       0.376623
182                 0.614815                        0       0.194805
785                 0.629630                        1       0.220779
924                 0.333333                        1       0.272727
780                 0.466667                        1       0.584416

      sex_male  chestpaintype_atypical angina  chestpaintype_non-anginal pain  \
1024         1                              1                               0
182          1                              1                               0
785          0                              0                               0
924          1                              0                               0
780          1                              0                               0

      chestpaintype_typical angina  rest_ecg_left venticular hypertrophy  \
1024                             0                                      1
182                              0                                      0
785                              0                                      1
924                              0                                      1
780                              0                                      1

      rest_ecg_normal  st_slope_flat  st_slope_upsloping
1024                0              1                   0
```

182	1	0	1
785	0	1	0
924	0	1	0
780	0	1	0

6 Cross Validation

Here we build different baseline models and perform 10-fold cross validaition to filter top performing baseline models to use in the level 0 of stacked ensemble method.

[42]:
```
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
#from sklearn.discriminate_analysis import LinearDiscriminateAnalysis
#import xgboost as xgb

# function initializing baseline machine learning models
def GetBasedModel():
    basedModels = []
    basedModels.append(('LR_L2'   , LogisticRegression(penalty='l2')))
   # basedModels.append(('LDA'  , LinearDiscriminantAnalysis()))
    basedModels.append(('KNN7'  , KNeighborsClassifier(7)))
    basedModels.append(('KNN5'  , KNeighborsClassifier(5)))
    basedModels.append(('KNN9'  , KNeighborsClassifier(9)))
    basedModels.append(('KNN11'  , KNeighborsClassifier(11)))
    basedModels.append(('CART' , DecisionTreeClassifier()))
    basedModels.append(('NB'   , GaussianNB()))
    basedModels.append(('SVM Linear'  ,␣
 ↪SVC(kernel='linear',gamma='auto',probability=True)))
    basedModels.append(('SVM RBF'  ,␣
 ↪SVC(kernel='rbf',gamma='auto',probability=True)))
    basedModels.append(('AB'   , AdaBoostClassifier()))
    basedModels.append(('GBM'  ,␣
 ↪GradientBoostingClassifier(n_estimators=100,max_features='sqrt')))
    basedModels.append(('RF_Ent100'   ,␣
 ↪RandomForestClassifier(criterion='entropy',n_estimators=100)))
    basedModels.append(('RF_Gini100'   ,␣
 ↪RandomForestClassifier(criterion='gini',n_estimators=100)))
    basedModels.append(('ET100'   , ExtraTreesClassifier(n_estimators= 100)))
    basedModels.append(('ET500'   , ExtraTreesClassifier(n_estimators= 500)))
    basedModels.append(('MLP', MLPClassifier()))
    #basedModels.append(('SGD3000', SGDClassifier(max_iter=1000, tol=1e-4)))
    basedModels.append(('ET1000'   , ExtraTreesClassifier(n_estimators= 1000)))

    return basedModels
```

```
# function for performing 10-fold cross validation of all the baseline models
def BasedLine2(X_train, y_train,models):
    # Test options and evaluation metric
    num_folds = 10
    scoring = 'accuracy'
    seed = 7
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed)
        cv_results = model_selection.cross_val_score(model, X_train, y_train,␣
 ↪cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)


    return results,msg
```

[43]:
```
models = GetBasedModel()
names,results = BasedLine2(X_train, y_train,models)
```

```
LR_L2: 0.851704 (0.051909)
KNN7: 0.851659 (0.047773)
KNN5: 0.843079 (0.043544)
KNN9: 0.857001 (0.040182)
KNN11: 0.852745 (0.039732)
CART: 0.865557 (0.019567)
NB: 0.845310 (0.048020)
SVM Linear: 0.852803 (0.052849)
SVM RBF: 0.852745 (0.044010)
AB: 0.853832 (0.028773)
GBM: 0.891169 (0.035169)
RF_Ent100: 0.930679 (0.036256)
RF_Gini100: 0.937074 (0.040172)
ET100: 0.921036 (0.026989)
ET500: 0.919973 (0.031996)
MLP: 0.868771 (0.037063)
ET1000: 0.923187 (0.033194)
```

7 Model Building

7.0.1 Random Forest Classifier (criterion = ‘entropy’)

[44]:
```
rf_ent = RandomForestClassifier(criterion = 'entropy',n_estimators=100)
rf_ent.fit(X_train, y_train)
y_pred_rfe = rf_ent.predict(X_test)
```

7.0.2 Multi Layer Perceptron

[45]:
```
mlp = MLPClassifier()
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)
```

7.0.3 K Nearest Neighbor (n=9)

[46]:
```
knn = KNeighborsClassifier(9)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

7.0.4 Extra Tree Classifier (n_estimators=500)

[47]:
```
et_500 = ExtraTreesClassifier(n_estimators= 500)
et_500.fit(X_train, y_train)
y_pred_et_500 = et_500.predict(X_test)
```

7.0.5 Support Vector Classifier (kernel=‘linear’)

[48]:
```
svc = SVC(kernel='linear', gamma='auto', probability=True)
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)
```

7.0.6 Adaboost Classifier

[49]:
```
ada = AdaBoostClassifier()
ada.fit(X_train, y_train)
y_pred_ada = ada.predict(X_test)
```

7.0.7 Decision Tree Classifier (CART)

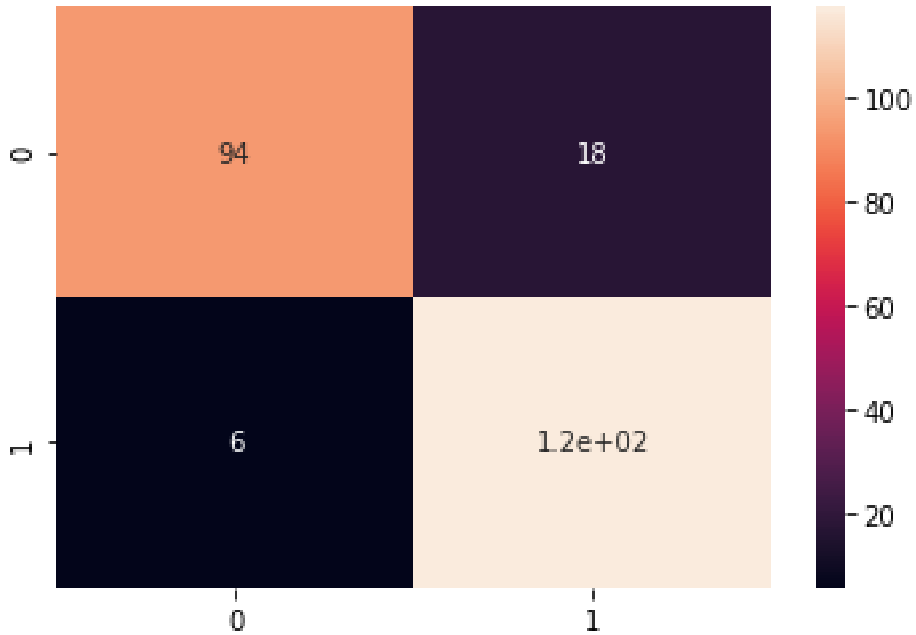
[50]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred_dtc = dtc.predict(X_test)
```

8 Model Evaluation

Here we define the evaluation metrics for our model. The most important metrics for this domain are specificity, Precision, Geometric mean, sensitivity, ROC AUC curve and matthew correlation coefficient.

[51]:
```
#F1 Score = 2(Recall Precision)/(Recall+Precision)
CM = confusion_matrix(y_test, y_pred_rfe)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_rfe)
acc = accuracy_score(y_test, y_pred_rfe)
roc = roc_auc_score(y_test, y_pred_rfe)
precision = precision_score(y_test, y_pred_rfe)
recall = recall_score(y_test, y_pred_rfe)
f1score = f1_score(y_test, y_pred_rfe)
matthew = matthews_corrcoef(y_test, y_pred_rfe) #if this val is more towards +1␣
↪then it is good model
model_results = pd.DataFrame([['Random Forest',acc, precision, recall,␣
↪specificity, f1score,roc, loss_log, matthew]], columns = ['Model',␣
↪'Accuracy' , 'Precision' , 'Sensitivity' , 'Specificity' , 'F1 Score' ,␣
↪'ROC' , 'Log_Loss' , 'matthew_corrcoef'])

model_results
```

[51]:
```
           Model  Accuracy  Precision  Sensitivity  Specificity  F1 Score  \
0  Random Forest  0.897872   0.866667      0.95122     0.839286  0.906977

        ROC  Log_Loss  matthew_corrcoef
0  0.895253  3.527426          0.798545
```



8.1 Comparison with other Models

[52]:
```
data = {    'MLP': y_pred_mlp,
            'KNN': y_pred_knn,
            'Extra tree classifier': y_pred_et_500,
            'SVC': y_pred_svc,
            'Adaboost': y_pred_ada,
            'CART': y_pred_dtc }

models = pd.DataFrame(data)

for column in models:
    CM=confusion_matrix(y_test,models[column])

    TN = CM[0][0]
    FN = CM[1][0]
    TP = CM[1][1]
    FP = CM[0][1]
    specificity = TN/(TN+FP)
    loss_log = log_loss(y_test, models[column])
    acc= accuracy_score(y_test, models[column])
    roc=roc_auc_score(y_test, models[column])
    precision = precision_score(y_test, models[column])
    recall = recall_score(y_test, models[column])
    f1 = f1_score(y_test, models[column])
```

```
    matthew = matthews_corrcoef(y_test, models[column])
    results =pd.DataFrame([[column,acc, precision,recall,specificity, f1,roc,␣
 ↪loss_log,matthew]],
                columns = ['Model', 'Accuracy','Precision',␣
 ↪'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','matthew_corrcoef'])
    model_results = model_results.append(results, ignore_index = True)

model_results
```

[52]:

```
                    Model  Accuracy  Precision  Sensitivity  Specificity  \
0           Random Forest  0.897872   0.866667     0.951220     0.839286
1                     MLP  0.829787   0.798561     0.902439     0.750000
2                     KNN  0.808511   0.786765     0.869919     0.741071
3  Extra tree classifier  0.897872   0.872180     0.943089     0.848214
4                     SVC  0.825532   0.801471     0.886179     0.758929
5                Adaboost  0.834043   0.813433     0.886179     0.776786
6                    CART  0.838298   0.829457     0.869919     0.803571

   F1 Score       ROC  Log_Loss  matthew_corrcoef
0  0.906977  0.895253  3.527426          0.798545
1  0.847328  0.826220  5.879036          0.662916
2  0.826255  0.805495  6.613907          0.618029
3  0.906250  0.895652  3.527422          0.797405
4  0.841699  0.822554  6.026006          0.652539
5  0.848249  0.831482  5.732052          0.668866
6  0.849206  0.836745  5.585068          0.675997
```

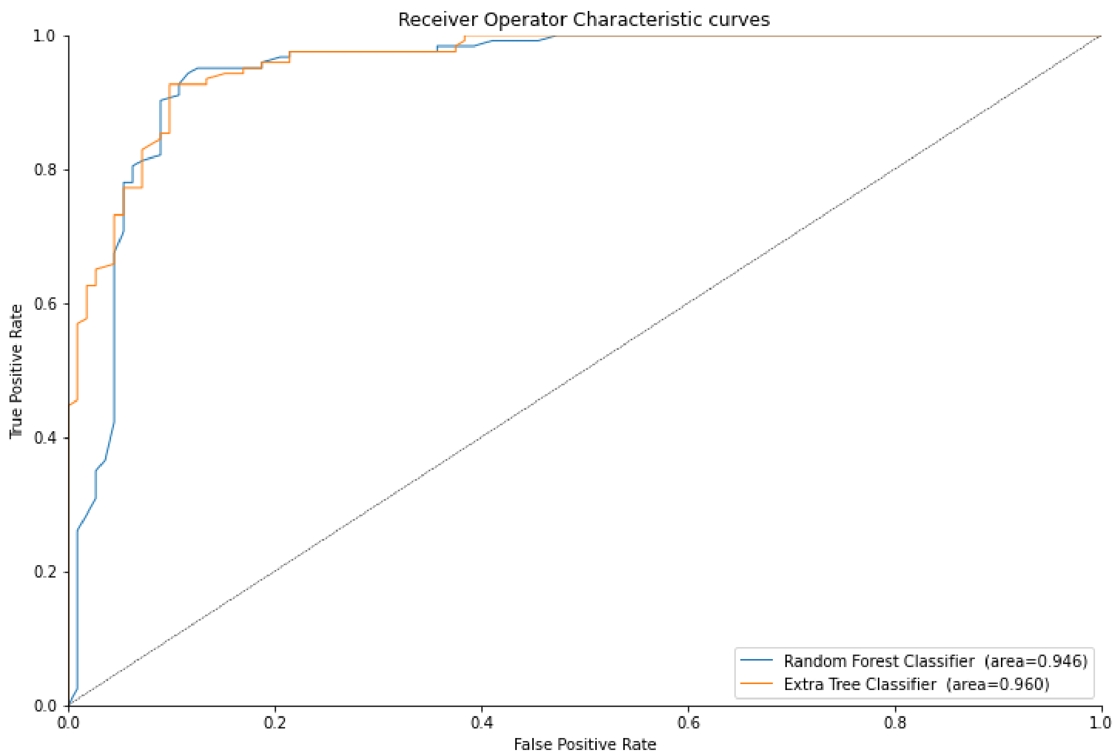
The above data frame shows that Random Forest Classifier is best with highest accuracy of 0.8978, sensitivity: 0.8392 and specificity of 0.8392 and highest f1_score of 0.9069 and lowest log_loss.

8.1.1 ROC AUC Curve

[53]:

```
def roc_auc_plot(y_true, y_proba, label=' ', l='-', lw=1.0):
    from sklearn.metrics import roc_curve, roc_auc_score
    fpr, tpr, _ = roc_curve(y_true, y_proba[:,1])
    ax.plot(fpr, tpr, linestyle=l, linewidth=lw,
            label="%s (area=%.3f)"%(label,roc_auc_score(y_true, y_proba[:,1])))

f, ax = plt.subplots(figsize=(12,8))


roc_auc_plot(y_test,rf_ent.predict_proba(X_test),label='Random Forest␣
 ↪Classifier ',l='-')
roc_auc_plot(y_test,et_500.predict_proba(X_test),label='Extra Tree Classifier␣
 ↪',l='-')
```

```
ax.plot([0,1], [0,1], color='k', linewidth=0.5, linestyle='--',
        )
ax.legend(loc="lower right")
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_title('Receiver Operator Characteristic curves')
sns.despine()
```

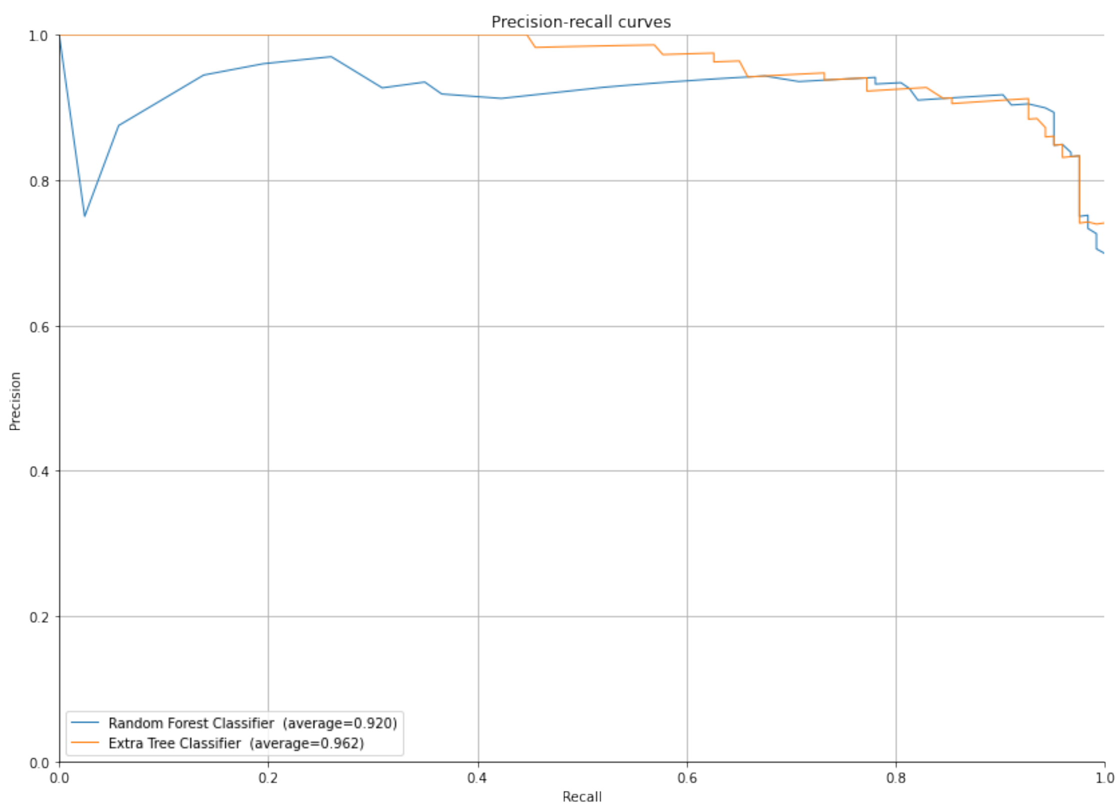


The highest avg area under the curve is attained by Extra Tree Classifier i.e., 0.960 #highest area means more generalized

8.1.2 Precision Recall Curve

[54]:
```
def precision_recall_plot(y_true, y_proba, label=' ', l='-', lw=1.0):
    from sklearn.metrics import precision_recall_curve, average_precision_score
    precision, recall, _ = precision_recall_curve(y_test,
                                                  y_proba[:,1])
    average_precision = average_precision_score(y_test, y_proba[:,1],
                                                average="micro")
```

```
    ax.plot(recall, precision, label='%s (average=%.
↪3f)'%(label,average_precision),
            linestyle=l, linewidth=lw)

f, ax = plt.subplots(figsize=(14,10))

precision_recall_plot(y_test,rf_ent.predict_proba(X_test),label='Random Forest␣
↪Classifier ',l='-')
precision_recall_plot(y_test,et_500.predict_proba(X_test),label='Extra Tree␣
↪Classifier ',l='-')

ax.set_xlabel('Recall')
ax.set_ylabel('Precision')
ax.legend(loc="lower left")
ax.grid(True)
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_title('Precision-recall curves')
sns.despine()
```



8.2 Feature Selection

[55]:
```
num_feats=11

def cor_selector(X, y,num_feats):
    cor_list = []
    feature_name = X.columns.tolist()
    # calculate the correlation with y for each feature
    for i in X.columns.tolist():
        cor = np.corrcoef(X[i], y)[0, 1]
        cor_list.append(cor)
    # replace NaN with 0
    cor_list = [0 if np.isnan(i) else i for i in cor_list]
    # feature name
    cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.
 ↪tolist()
    # feature selection? 0 for not select, 1 for select
    cor_support = [True if i in cor_feature else False for i in feature_name]
    return cor_support, cor_feature
cor_support, cor_feature = cor_selector(X, y,num_feats)
print(str(len(cor_feature)), 'selected features')
```

```
11 selected features
```

[56]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
X_norm =MinMaxScaler().fit_transform(X)
chi_selector = SelectKBest(chi2, k=num_feats)
chi_selector.fit(X_norm, y)
chi_support = chi_selector.get_support()
chi_feature = X.loc[:,chi_support].columns.tolist()
print(str(len(chi_feature)), 'selected features')
```

```
11 selected features
```

[57]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
rfe_selector = RFE(estimator=LogisticRegression(),␣
 ↪n_features_to_select=num_feats, step=10, verbose=5)
rfe_selector.fit(X_norm, y)
rfe_support = rfe_selector.get_support()
rfe_feature = X.loc[:,rfe_support].columns.tolist()
print(str(len(rfe_feature)), 'selected features')
```

```
Fitting estimator with 15 features.
11 selected features
```

[58]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
embeded_lr_selector = SelectFromModel(LogisticRegression(penalty="l2",␣
↪solver='lbfgs'), max_features=num_feats)
embeded_lr_selector.fit(X_norm, y) #penalty l2 is used for feature selection

embeded_lr_support = embeded_lr_selector.get_support()
embeded_lr_feature = X.loc[:,embeded_lr_support].columns.tolist()
print(str(len(embeded_lr_feature)), 'selected features')
```

```
7 selected features
```

[59]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100,␣
↪criterion='gini'), max_features=num_feats)
embeded_rf_selector.fit(X, y)
embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features')
```

```
8 selected features
```

[60]:
```
#putting all selections together
feature_name = X.columns
feature_selection_df = pd.DataFrame({'Feature':feature_name, 'Pearson':
↪cor_support, 'Chi-2':chi_support, 'RFE':rfe_support, 'Logistics':
↪embeded_lr_support,
                                    'Random Forest':embeded_rf_support})
# count the selected times for each feature
feature_selection_df['Total'] = np.sum(feature_selection_df, axis=1)
# display the top 100
feature_selection_df = feature_selection_df.sort_values(['Total','Feature'] ,␣
↪ascending=False)
feature_selection_df.index = range(1, len(feature_selection_df)+1)
feature_selection_df.head(num_feats)
```

[60]:

	Feature	Pearson	Chi-2	RFE	Logistics	\
1	st_slope_flat	True	True	True	True	
2	st_depression	True	True	True	True	
3	st_slope_upsloping	True	True	True	False	
4	sex_male	True	True	True	True	
5	max_heartrate_achieved	True	True	True	False	
6	exercise_induced_angina	True	True	True	False	
7	cholesterol	True	False	True	True	
8	chestpaintype_non-anginal pain	True	True	True	True	
9	chestpaintype_atypical angina	True	True	True	True	

```
10                              age      True    True    True       False
11                fasting_blood_sugar      True    True   False       False

    Random Forest  Total
1            True      5
2            True      5
3            True      4
4           False      4
5            True      4
6            True      4
7            True      4
8           False      4
9           False      4
10           True      4
11          False      2
```

[61]:
```
#segregating dataset into features(X) and target variable (y)
X = df.
 ↪drop(['target','resting_blood_pressure','sex_male','chestpaintype_non-anginal␣
 ↪pain','chestpaintype_atypical angina'],axis=1)
y = df['target']
```

[62]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪test_size=0.2, shuffle=True, random_state=5)
```

[63]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train[['age','cholesterol','max_heartrate_achieved','st_depression']] =␣
 ↪scaler.
 ↪fit_transform(X_train[['age','cholesterol','max_heartrate_achieved','st_depression']])
X_train.head()
```

[63]:
```
          age  cholesterol  fasting_blood_sugar  max_heartrate_achieved  \
478  0.673469     0.000000                    1                0.303704
253  0.673469     0.594705                    0                0.355556
273  0.551020     0.409369                    0                0.466667
111  0.591837     0.519348                    0                0.185185
50   0.448980     0.474542                    0                0.400000

     exercise_induced_angina  st_depression  chestpaintype_typical angina  \
478                        1       0.454545                             0
253                        1       0.194805                             0
273                        1       0.584416                             0
111                        1       0.584416                             0
50                         1       0.454545                             0

     rest_ecg_left venticular hypertrophy  rest_ecg_normal  st_slope_flat  \
```

```
478                                             0               1            0
253                                             0               0            0
273                                             0               1            1
111                                             0               1            1
50                                              0               1            1

     st_slope_upsloping
478                   0
253                   1
273                   0
111                   0
50                    0
```

[64]:
```
X_test[['age','cholesterol','max_heartrate_achieved','st_depression']] = scaler.
 ↪fit_transform(X_test[['age','cholesterol','max_heartrate_achieved','st_depression']])
X_test.head()
```

[64]:
```
        age  cholesterol  fasting_blood_sugar  max_heartrate_achieved  \
1024  0.700     0.690418                    0                0.205357
182   0.425     0.550369                    0                0.625000
785   0.275     0.579853                    0                0.642857
924   0.575     0.678133                    0                0.285714
780   0.600     0.636364                    0                0.446429

      exercise_induced_angina  st_depression  chestpaintype_typical angina  \
1024                        0       0.606061                            0
182                         0       0.393939                            0
785                         1       0.424242                            0
924                         1       0.484848                            0
780                         1       0.848485                            0

      rest_ecg_left venticular hypertrophy  rest_ecg_normal  st_slope_flat  \
1024                                     1                0              1
182                                      0                1              0
785                                      1                0              1
924                                      1                0              1
780                                      1                0              1

      st_slope_upsloping
1024                   0
182                    1
785                    0
924                    0
780                    0
```

[65]:
```
models = GetBasedModel()
names,results = BasedLine2(X_train, y_train,models)
```

```
LR_L2: 0.822878 (0.047967)
KNN7: 0.807904 (0.044339)
KNN5: 0.803660 (0.045820)
KNN9: 0.815374 (0.036848)
KNN11: 0.809037 (0.042388)
CART: 0.875132 (0.033734)
NB: 0.822889 (0.042157)
SVM Linear: 0.815488 (0.045773)
SVM RBF: 0.797278 (0.049967)
AB: 0.816529 (0.036133)
GBM: 0.863384 (0.027318)
RF_Ent100: 0.922112 (0.026891)
RF_Gini100: 0.924251 (0.025774)
ET100: 0.922112 (0.023835)
ET500: 0.918909 (0.027395)
MLP: 0.837783 (0.041146)
ET1000: 0.922135 (0.028093)
```

8.3 Soft Voting

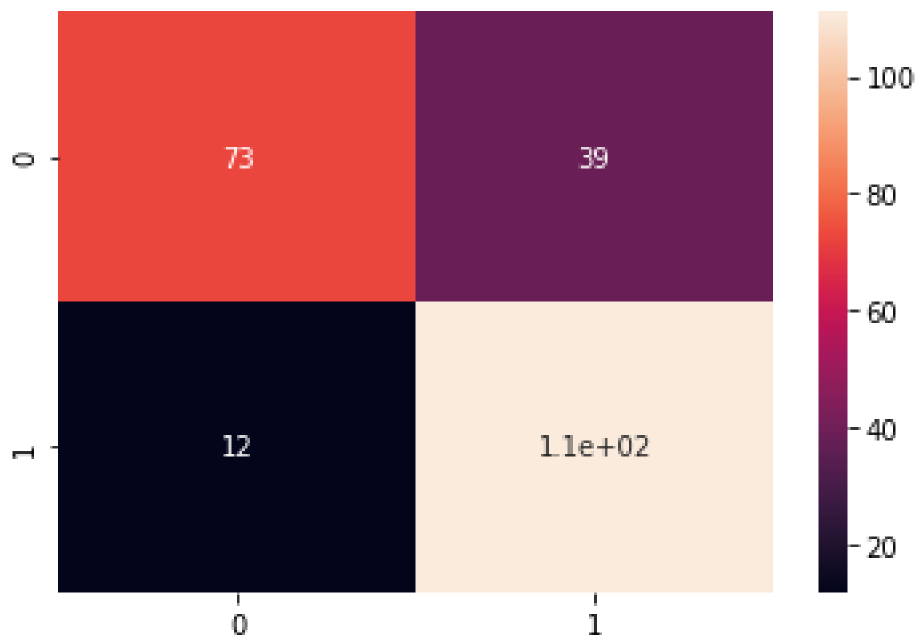
[66]:
```
#here we are selecting the top 3 performing models and we assign weights a/c to␣
↪their accuracy.
clf1=RandomForestClassifier(criterion='entropy',n_estimators=100)

clf2=DecisionTreeClassifier()

clf3=ExtraTreesClassifier(n_estimators= 1000)


eclf1 = VotingClassifier(estimators=[('rfe', clf1), ('dtc', clf2),␣
↪('ET',clf3),],
                         voting='soft', weights=[4,2,5])
eclf1.fit(X_train,y_train)
y_pred_sv =eclf1.predict(X_test)
```

[67]:
```
CM=confusion_matrix(y_test,y_pred_sv)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(y_test, y_pred_sv)
acc= accuracy_score(y_test, y_pred_sv)
roc=roc_auc_score(y_test, y_pred_sv)
precision = precision_score(y_test, y_pred_sv)
```

```
recall = recall_score(y_test, y_pred_sv)
f1 = f1_score(y_test, y_pred_sv)

matthew = matthews_corrcoef(y_test, y_pred_sv)
model_results =pd.DataFrame([['Soft Voting',acc, precision,recall,specificity,␣
↪f1,roc, loss_log,matthew]],
               columns = ['Model', 'Accuracy','Precision',␣
↪'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','matthew_corrcoef'])

model_results
```

[67]:

```
         Model  Accuracy  Precision  Sensitivity  Specificity  F1 Score  \
0  Soft Voting  0.782979       0.74     0.902439     0.651786  0.813187

        ROC  Log_Loss  matthew_corrcoef
0  0.777112  7.495782          0.576093
```



[68]:

```
rf_ent = RandomForestClassifier(criterion='entropy',n_estimators=100)
rf_ent.fit(X_train, y_train)
y_pred_rfe = rf_ent.predict(X_test)
```

[69]:

```
mlp = MLPClassifier()
mlp.fit(X_train,y_train)
y_pred_mlp = mlp.predict(X_test)
```

[70]:
```
knn = KNeighborsClassifier(9)
knn.fit(X_train,y_train)
y_pred_knn = knn.predict(X_test)
```

[71]:
```
et_1000 = ExtraTreesClassifier(n_estimators= 1000)
et_1000.fit(X_train,y_train)
y_pred_et1000 = et_1000.predict(X_test)
```

[72]:
```
svc = SVC(kernel='linear',gamma='auto',probability=True)
svc.fit(X_train,y_train)
y_pred_svc = svc.predict(X_test)
```

[73]:
```
ada = AdaBoostClassifier()
ada.fit(X_train,y_train)
y_pred_ada = ada.predict(X_test)
```

[74]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)
y_pred_dtc = dtc.predict(X_test)
```

8.3.1 Comparing with other models

[75]:
```
data = {
             'Random Forest Entropy': y_pred_rfe,
                'MLP2': y_pred_mlp,
                'KNN2': y_pred_knn,
                'EXtra tree classifier': y_pred_et1000,
                'SVC2': y_pred_svc,
                'Adaboost': y_pred_ada,
                'CART': y_pred_dtc }

models = pd.DataFrame(data)
for column in models:
    CM=confusion_matrix(y_test,models[column])

    TN = CM[0][0]
    FN = CM[1][0]
    TP = CM[1][1]
    FP = CM[0][1]
    specificity = TN/(TN+FP)
    loss_log = log_loss(y_test, models[column])
    acc= accuracy_score(y_test, models[column])
    roc=roc_auc_score(y_test, models[column])
    prec = precision_score(y_test, models[column])
    rec = recall_score(y_test, models[column])
    f1 = f1_score(y_test, models[column])
```

```
    mathew = matthews_corrcoef(y_test, models[column])
    results =pd.DataFrame([[column,acc, prec,rec,specificity, f1,roc,␣
 ↪loss_log,mathew]],
                   columns = ['Model', 'Accuracy','Precision',␣
 ↪'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrcoef'])
    model_results = model_results.append(results, ignore_index = True)

model_results
```

[75]:

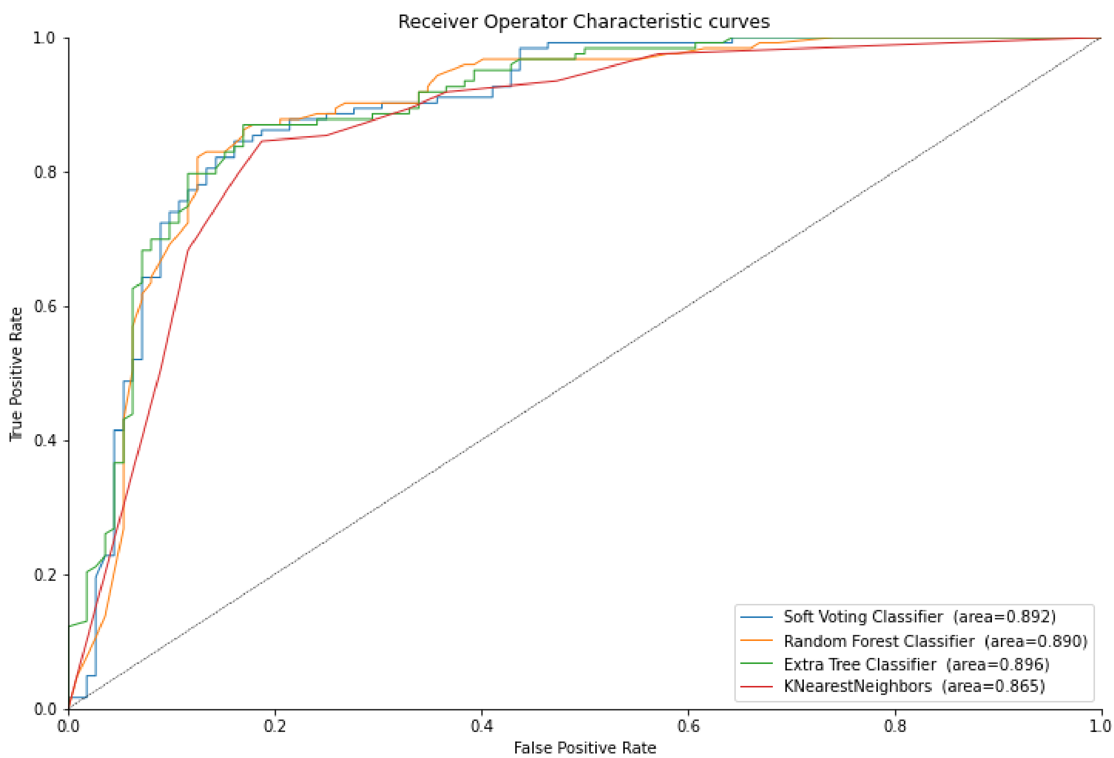
```
                     Model  Accuracy  Precision  Sensitivity  Specificity  \
0              Soft Voting  0.782979   0.740000     0.902439     0.651786
1   Random Forest Entropy  0.800000   0.760274     0.902439     0.687500
2                     MLP2  0.774468   0.736486     0.886179     0.651786
3                     KNN2  0.804255   0.789474     0.853659     0.750000
4   EXtra tree classifier  0.787234   0.748299     0.894309     0.669643
5                     SVC2  0.800000   0.756757     0.910569     0.678571
6                 Adaboost  0.761702   0.718954     0.894309     0.616071
7                     CART  0.744681   0.708609     0.869919     0.607143

   F1 Score       ROC  Log_Loss  matthew_corrcoef  mathew_corrcoef
0  0.813187  0.777112  7.495782          0.576093              NaN
1  0.825279  0.794970  6.907874               NaN         0.607431
2  0.804428  0.768982  7.789729               NaN         0.556448
3  0.820312  0.801829  6.760877               NaN         0.608313
4  0.814815  0.781976  7.348802               NaN         0.581974
5  0.826568  0.794570  6.907878               NaN         0.609383
6  0.797101  0.755190  8.230663               NaN         0.534814
7  0.781022  0.738531  8.818561               NaN         0.497173
```

ROC AUC curve

[76]:

```
def roc_auc_plot(y_true, y_proba, label=' ', l='-', lw=1.0):
    from sklearn.metrics import roc_curve, roc_auc_score
    fpr, tpr, _ = roc_curve(y_true, y_proba[:,1])
    ax.plot(fpr, tpr, linestyle=l, linewidth=lw,
            label="%s (area=%.3f)"%(label,roc_auc_score(y_true, y_proba[:,1])))

f, ax = plt.subplots(figsize=(12,8))

roc_auc_plot(y_test,eclf1.predict_proba(X_test),label='Soft Voting Classifier␣
 ↪',l='-')
roc_auc_plot(y_test,rf_ent.predict_proba(X_test),label='Random Forest␣
 ↪Classifier ',l='-')
roc_auc_plot(y_test,et_1000.predict_proba(X_test),label='Extra Tree Classifier␣
 ↪',l='-')
roc_auc_plot(y_test,knn.predict_proba(X_test),label='KNearestNeighbors ',l='-')

ax.plot([0,1], [0,1], color='k', linewidth=0.5, linestyle='--',
```

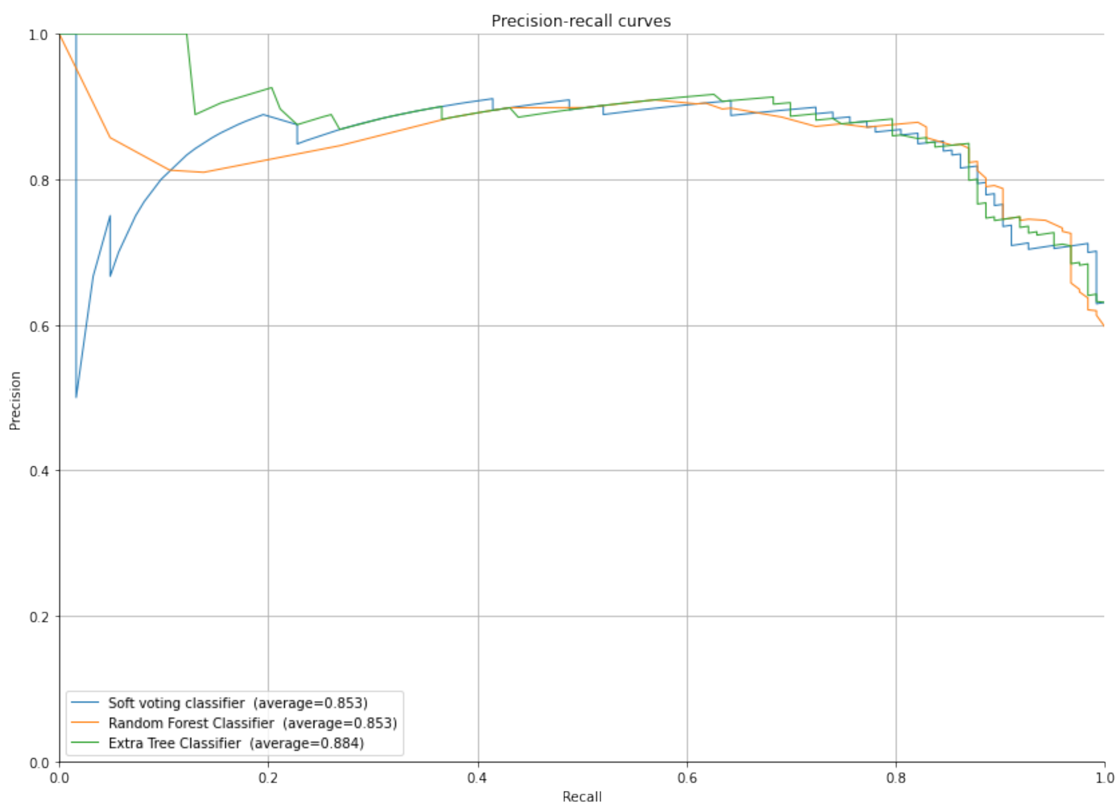
```
        )
ax.legend(loc="lower right")
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_title('Receiver Operator Characteristic curves')
sns.despine()
```



Precision recall curve

[77]:
```
def precision_recall_plot(y_true, y_proba, label=' ', l='-', lw=1.0):
    from sklearn.metrics import precision_recall_curve, average_precision_score
    precision, recall, _ = precision_recall_curve(y_test,
                                                  y_proba[:,1])
    average_precision = average_precision_score(y_test, y_proba[:,1],
                                                   average="micro")
    ax.plot(recall, precision, label='%s (average=%.
 ↪3f)'%(label,average_precision),
            linestyle=l, linewidth=lw)

f, ax = plt.subplots(figsize=(14,10))
```

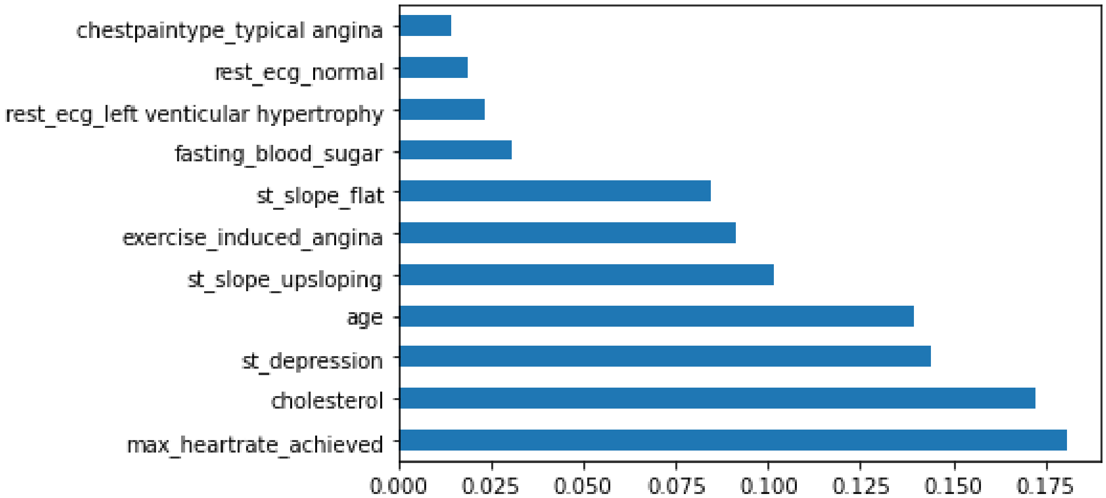
```
precision_recall_plot(y_test,eclf1.predict_proba(X_test),label='Soft voting␣
 ↪classifier ',l='-')
precision_recall_plot(y_test,rf_ent.predict_proba(X_test),label='Random Forest␣
 ↪Classifier ',l='-')
precision_recall_plot(y_test,et_1000.predict_proba(X_test),label='Extra Tree␣
 ↪Classifier ',l='-')

ax.set_xlabel('Recall')
ax.set_ylabel('Precision')
ax.legend(loc="lower left")
ax.grid(True)
ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_title('Precision-recall curves')
sns.despine()
```



8.3.2 Feature Importance

[78]:
```
feat_importances = pd.Series(rf_ent.feature_importances_, index=X_train.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

[78]: <AxesSubplot:>



8.4 Conclusion

1.We see that the best performing models are Random Forest algorithm and Extra Tree Classifier. 2.Also, stacked ensemble of power machine learning algos(softvoting) resulted in higher performance than individual machine learning algorithm/model. 3.Major contributing features are: Max heartrate achieved, Cholesterol, age, st_depression, st_upsloping, Exercise induced angina.

[ ]: