There is map available in skilearn to choosed ML model. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

How to chose ML model in sklear(search)?

In sklearn, model is called estimator. ML applications has been divided ainto four broad categories. Classification Clustering Regression Dimensionality regression.

Every application has pre_built models.

How to choose these models? following things must be known to decide about choosing the model: what is to be predicted? E.g if number is to be predicted, it will be regression problem.

Predicting numbers is regression.

Whether labelled data is available or not?

Number of samples we have.

iF Ridge doesn't give high score we will move to Ensemble Regressor.

Ensemble Methods: Take many regressor model and give average of it.

Forestof randomzed tree: it is collection of if else statement.

Classification: Choosing either yes or no is classification. E.g either a person is ill or healthy?

If we have structured data, prefer ensemble method. If we have unstructured data, apply deep learning.

Note: For structured data prefer ensemble methods. For unstructured data prefer deep learning.

Importing Data, Boston Hosusing

In [2]:
```
import pandas as pd
from sklearn.datasets import load_boston
boston=load_boston()
boston
```

Out[2]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
```

```
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DI
S', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n----------------
-----------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506
\n\n    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.\n\n    :Attribute Information (in orde
r):\n        - CRIM     per capita crime rate by town\n        - ZN       propo
rtion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS
proportion of non-retail business acres per town\n        - CHAS     Charles Ri
ver dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX
nitric oxides concentration (parts per 10 million)\n        - RM       average
number of rooms per dwelling\n        - AGE      proportion of owner-occupied u
nits built prior to 1940\n        - DIS      weighted distances to five Boston
employment centres\n        - RAD      index of accessibility to radial highway
s\n        - TAX      full-value property-tax rate per $10,000\n        - PTRAT
IO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk
is the proportion of blacks by town\n        - LSTAT    % lower status of the p
opulation\n        - MEDV     Median value of owner-occupied homes in $1000's\n
\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Rubinfe
ld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.
edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was taken from the
StatLib library which is maintained at Carnegie Mellon University.\n\nThe Bosto
n house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the
demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-102, 197
8.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980.
N.B. Various transformations are used in the table on\npages 244-261 of the lat
ter.\n\nThe Boston house-price data has been used in many machine learning pape
rs that address regression\nproblems.   \n     \n.. topic:: References\n\n   -
Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data an
d Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Comb
ining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Inte
rnational Conference of Machine Learning, 236-243, University of Massachusetts,
Amherst. Morgan Kaufmann.\n",
 'filename': 'C:\\Users\\inn\\anaconda3\\lib\\site-packages\\sklearn\\datasets
\\data\\boston_house_prices.csv'}
```

In [3]:
```
df_boston= pd.DataFrame(boston["data"], columns=boston["feature_names"])
df_boston["target"]= pd.Series(boston["target"])
df_boston.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [4]:
```
len(df_boston)
```

Out[4]:
```
506
```

In [5]:
```
from sklearn.linear_model import Ridge
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
np.random.seed(1)
#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
reg_model= Ridge()
reg_model.fit(x_train, y_train)
#check the score of model
reg_model.score(x_test, y_test)
```

Out[5]:
```
0.7655800611077144
```

In [6]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
np.random.seed(1)
#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
reg_model= RandomForestRegressor()
reg_model.fit(x_train, y_train)
#check the score of model
reg_model.score(x_test, y_test)
```

Out[6]:
```
0.9124687687774722
```

In [7]:
```
heart_data=pd.read_csv('heart_failure_clinical_records_dataset.csv')
heart_data
```

Out[7]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	p
0	75.0	0	582	0	20	1	26
1	55.0	0	7861	0	38	0	26
2	65.0	0	146	0	20	0	16
3	50.0	1	111	0	20	0	21
4	65.0	1	160	1	20	0	32
...	...	...	...	...	...	...	
294	62.0	0	61	1	38	1	15
295	55.0	0	1820	0	38	0	27
296	45.0	0	2060	1	60	0	74
297	45.0	0	2413	0	38	0	14
298	50.0	0	196	0	45	0	39

299 rows × 13 columns

In [8]:
```
len(heart_data)
```

Out[8]:
```
299
```

In [9]:
```
#Changing model from regression libraries to improve result.
from sklearn.svm import LinearSVC
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
#np.random.seed(1)
#create data
x=heart_data.drop("DEATH_EVENT", axis=1)
y=heart_data["DEATH_EVENT"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
svm_model= LinearSVC()
svm_model.fit(x_train, y_train)
#check the score of model
svm_model.score(x_test, y_test)
```

```
C:\Users\inn\anaconda3\lib\site-packages\sklearn\svm\_base.py:976: ConvergenceW
arning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[9]: 0.2833333333333333

In [10]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
#np.random.seed(1)
#create data
x=heart_data.drop("DEATH_EVENT", axis=1)
y=heart_data["DEATH_EVENT"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
svm_model= RandomForestClassifier()
svm_model.fit(x_train, y_train)
#check the score of model
svm_model.score(x_test, y_test)
```

Out[10]: 0.8666666666666667

Making Prediction:

There are two ways of prediction: predict(). predict_proba().

Stages of execution of ML model: Stage 1: Training. Stage 2 Predicting.

When score is calculated, there must a predicted value. When score is run, it calls predicted values in the background.

For stage 2 there ust be actual value as well as predicted value. Now, scre compares actual and predicted and tells how much they resemble.

firstly, system is trained. When test data is given then firstly compiler predicts output of test data and then compare it with real output.

predict() method predicts individual value. Predict_proba gives probability of individual target value.

Prediction on Classification Models

Predict() Method:

In [11]:
```
heart_dis = pd.read_csv('heart.csv')
heart_dis.head()
```

Out[11]:

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

In [12]:
```
x1= heart_dis.drop('target', axis=1)
y1=heart_dis['target']
```

In [21]:
```
#Choosing ML model.
from sklearn.ensemble import RandomForestClassifier
#splitting data into test and train
from sklearn.model_selection import train_test_split
x1_test, x1_train, y1_test, y1_train = train_test_split(x1,y1, test_size=0.2)
clf=RandomForestClassifier().fit(x1_train, y1_train)
```

In [ ]:
```

```

In [22]:
```
#evaluating mode
y1_predicted= clf.predict(x1_test)
y1_predicted # predicted output of y1_test data
```

Out[22]:
```
array([0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1],
      dtype=int64)
```

In [23]:
```
np.array(y1_test) #real output of test data
```

Out[23]:
```
array([0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

In [24]:
```
np.mean(y1_predicted == y1_test) #method 1 to find score
```

Out[24]:
```
0.8140495867768595
```

In [25]:
```
clf.score(x1_test, y1_test) # METHOD 2 to find score.
```

Out[25]:
```
0.8140495867768595
```

Here actually score() method is called upon model RandomforestClassifier which is trained by train data an now test data is given to find score or to find how much y1_test is as similar as predicted by machine.

Preddict_proba()

Predict_proba gives probability of individual target value

In [26]:
```
clf.predict(x1_test)[:5]
```

Out[26]:
```
array([0, 1, 1, 1, 0], dtype=int64)
```

In [27]:
```
clf.predict_proba(x1_test)[:5]
```

Out[27]:
```
array([[0.78, 0.22],
       [0.2 , 0.8 ],
       [0.39, 0.61],
       [0.09, 0.91],
       [0.93, 0.07]])
```

Here for every value a probability is predicted and then decided 0 or 1. E.g for ist 0 probability of 0 is 0.9 while probability for 1 is 0.1. Therefore 0 shown.

Prediction on Regression Models:

In [28]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
np.random.seed(1)

#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]

#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)

#instantiate ridge model
reg_model= RandomForestRegressor()
reg_model.fit(x_train, y_train)

#check the score of model
predict_y_test = reg_model.predict(x_test)
```

In [29]:
```
predict_y_test[:5] # predicted value.
```

Out[29]:
```
array([29.926, 27.022, 20.343, 20.593, 19.64 ])
```

In [30]:
```
np.array(y_test)[:5] # real value of y_test, ground reality.
```

Out[30]:
```
array([28.2, 23.9, 16.6, 22. , 20.8])
```

In [31]:
```
#compare prediction with real values of y_test, ground reality
# or finding deviation of predicted value from the real ones i.e y_test.
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, predict_y_test)
```

Out[31]:
```
2.2971568627451
```

Predicted values are only 2.29 percent away.

Evaluating ML Models:

Three ways:
Estimator "Score Method.
The Scoring Parameter.
Problem_Specific metric method.

https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

In [32]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
#np.random.seed(1)
#create data
x=heart_data.drop("DEATH_EVENT", axis=1)
y=heart_data["DEATH_EVENT"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
svm_model= RandomForestClassifier()
svm_model.fit(x_train, y_train)
#check the score of model
svm_model.score(x_test, y_test)
```

Out[32]:
```
0.8333333333333334
```

check help regarding score methods.

"Return the mean accuracy on the given test data and labels."

scrore is using mean accuracy metrics.

In [33]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
np.random.seed(1)
#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
reg_model= RandomForestRegressor()
reg_model.fit(x_train, y_train)
#check the score of model
reg_model.score(x_test, y_test)
```

Out[33]:
```
0.9124687687774722
```

check help regarding score methods.

```
    Return the coefficient of determination R^2 of the prediction.

    scrore is using R^2 accuracy metrics.
```

Cross Validation:

CV splits data into numbers given to CV If CV=5, it will split data into 5 parts. Moreover, if test_size=0.2, CV will take 20 percent of data from every split.

Cross validation gives u the ability to use different scoring matrices.

CV: cross validation splitting strategy.

In [34]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
#np.random.seed(1)
#create data
x=heart_data.drop("DEATH_EVENT", axis=1)
y=heart_data["DEATH_EVENT"]

x_train, x_test , y_train, y_test= train_test_split(x,y, test_size=0.2)
heart_model= RandomForestClassifier().fit(x_train,y_train)

heart_model.score(x_test, y_test)
```

Out[34]:
```
0.8333333333333334
```

In [35]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(heart_model, x, y, cv=5)
```

Out[35]:
```
array([0.45      , 0.85      , 0.83333333, 0.71666667, 0.6779661 ])
```

Here, see the probability of every part. First has the lowest while third is higest. So, there is flaw in first part.

Moreover, CV uses defualt scoring method i.e for mean accuracy for default and R^2 for regression, but it also gives opportunity to change it,

In [36]:
```
cross_val_score(heart_model, x, y, cv=5, scoring=None)
```

Out[36]:
```
array([0.45      , 0.81666667, 0.85      , 0.71666667, 0.6779661 ])
```

In [37]:
```
heart_model.score(x_test, y_test), np.mean(cross_val_score(heart_model, x, y, cv=
```

Out[37]:
```
(0.8333333333333334, 0.6889830508474576)
```

Classification Model Evaluation:

it can be evaluated using using four metrics:

1 Accuracy.
2 Area Under the Curve ROC.
3 Confusion Matrix
4 Classification Report

In [38]:
```
print(f"heart disease cross validation accuracy is { np.mean(cross_val_score(hear
```

```
heart disease cross validation accuracy is 68.559322
```

Area Under the Curve (AUC) or Reciever Operating Characteristic Curve (ROC)

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve

x_train, x_test , y_train, y_test= train_test_split(x,y, test_size=0.2)
clf=RandomForestClassifier().fit(x_train, y_train)

clf.fit(x_train,y_train)
y_prob = clf.predict_proba(x_test)
```

predict_proba gives multiple probabilities and mean of all these is taken. This is one dimensional vector.

while score gives only single value which is zero dimesional.

to increase reliability we are taking two dimensional vectors of fpr and tpr. In order to calculate this, we use area under the curve.

In [40]:
```
y_prob[:5]
```

Out[40]:
```
array([[0.79, 0.21],
       [0.84, 0.16],
       [0.54, 0.46],
       [0.64, 0.36],
       [0.37, 0.63]])
```

True positive: Model's prediction of 1 similar to real value 1.

False positive: Model's prediction of 0 does not similar to real value i.e 1.

True negative: Model's prediction of 0 similar to real value 0.

False negative: Model's prediction of 0 does not similar to real value i.e 1.

(positive: Model's 1, Negative: Model's 0)

Cases:

Case 1: True Positive : Truth= 1, Model=1

Case 2: False Positive : Truth= 0, Model=1

Case 3: True Negative : Truth= 0, Model=0

Case 4: False Negative : Truth= 1, Model=0

In [41]:
```
y_prob_positive= y_prob[:,1] # finding probability of y only.
```

In [42]:
```
y_prob_positive
```

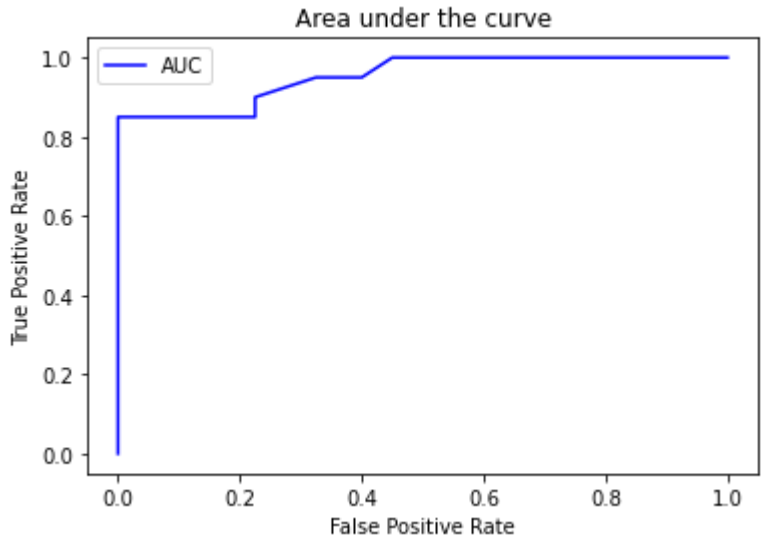
Out[42]:
```
array([0.21, 0.16, 0.46, 0.36, 0.63, 0.03, 0.81, 0.21, 0.54, 0.78, 0.17,
       0.1 , 0.13, 0.11, 0.05, 0.48, 0.03, 0.11, 0.06, 0.69, 0.07, 0.03,
       0.15, 0.02, 0.36, 0.53, 0.08, 0.83, 0.74, 0.03, 0.31, 0.28, 0.07,
       0.  , 0.74, 0.76, 0.09, 0.12, 0.42, 0.13, 0.24, 0.06, 0.21, 0.56,
       0.84, 0.07, 0.8 , 0.49, 0.74, 0.79, 0.02, 0.23, 0.13, 0.06, 0.21,
       0.36, 0.1 , 0.87, 0.21, 0.11])
```

In [43]:
```
fpr, tpr, threshold= roc_curve(y_test, y_prob_positive)
```

In [44]:
```
fpr #false_positive_rate
```

Out[44]:
```
array([0.   , 0.   , 0.   , 0.   , 0.   , 0.075, 0.15 , 0.225, 0.225,
       0.325, 0.4  , 0.45 , 0.475, 0.55 , 0.6  , 0.65 , 0.8  , 0.825,
       0.925, 0.975, 1.   ])
```

In [45]:
```
#Creatinf funtion for plotting
import matplotlib.pyplot as plt
def plot_roc(fpr,tpr):

    #plot roc curve
    plt.plot(fpr,tpr, color="blue", label="AUC")

    #Customize the plot
    plt.title("Area under the curve")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()

plot_roc(fpr,tpr)
```



In [46]:
```
#value of AUC
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_prob_positive)
```

Out[46]:
```
0.95375
```

Confusion Matrics:

It finds out Where model is confused or has lost his minds.
model is giving wrong prediction in case 2 and case 4.

In [49]:
```
from sklearn.metrics import confusion_matrix
y_predicted = heart_model.predict(x_test)
confusion_matrix(y_test, y_predicted)
```

Out[49]:
```
array([[39,  1],
       [ 3, 17]], dtype=int64)
```

Here, at position of 3 and 1 model is confused.

In [52]:
```
#Installing seaborn Heatmap
import sys
!conda install -- prefix {sys.prefix} seaborn
```

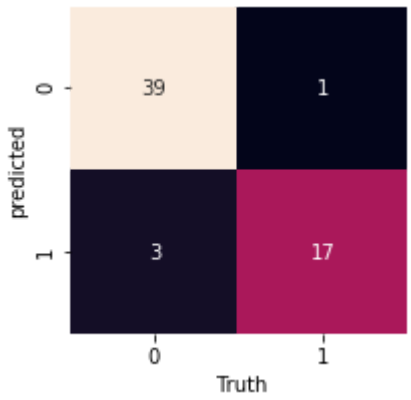
```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying w
ith flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying w
ith flexible solve.


PackagesNotFoundError: The following packages are not available from current ch
annels:

  - prefix
  - \users\inn\anaconda3

Current channels:

  - https://repo.anaconda.com/pkgs/main/win-64 (https://repo.anaconda.com/pkgs/
main/win-64)
  - https://repo.anaconda.com/pkgs/main/noarch (https://repo.anaconda.com/pkgs/
main/noarch)
  - https://repo.anaconda.com/pkgs/r/win-64 (https://repo.anaconda.com/pkgs/r/w
in-64)
  - https://repo.anaconda.com/pkgs/r/noarch (https://repo.anaconda.com/pkgs/r/n
oarch)
  - https://repo.anaconda.com/pkgs/msys2/win-64 (https://repo.anaconda.com/pkg
s/msys2/win-64)
  - https://repo.anaconda.com/pkgs/msys2/noarch (https://repo.anaconda.com/pkg
s/msys2/noarch)

To search for alternate channels that may provide the conda package you're
looking for, navigate to

    https://anaconda.org (https://anaconda.org)

and use the search bar at the top of the page.
```

In [53]:
```
import seaborn as sns

conf_mat= confusion_matrix(y_test, y_predicted)

def plot_conf_matrix(conf_mat):
    """
    This function plots confuion matrix
    """
    fig, ax=plt.subplots(figsize=(3,3))
    ax=sns.heatmap(conf_mat, annot=True, cbar=False)

    plt.xlabel(" Truth")
    plt.ylabel("predicted")
```

https://seaborn.pydata.org/generated/seaborn.heatmap.html (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

use this link to read documentation of seaborn map.

In [54]:
```
plot_conf_matrix(conf_mat)
```



Here balck region is showing where matrix is getting confused.

Classification Report:

Three basic concepts:

1 Class imbalance.
2 Recall.
3 Precision.
4 F1 Score

class imbalance is actually imbalance between data and required data. Recall means to include data to find relevent things in data. Precision means to include relevent data. Recall and precision are inversaly proportional.

There must be balance between recall and precision which s called F1 Score :

1 Recall= TP/(TP+FN) 2 Precision =TP/(TP+FP) 3 cobination of precision and recall.

In [55]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95        40
           1       0.94      0.85      0.89        20

    accuracy                           0.93        60
   macro avg       0.94      0.91      0.92        60
weighted avg       0.93      0.93      0.93        60
```

https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

Regression Model Evaluation Metrics:

1 R^2: Compares your model with the mean of your target.

2 Mean Absolute Error (MAE)

3 Mean Squared Error (MSE)

https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

R^2:

In regression model value of R^2 model has is to be maximized.

In [56]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
np.random.seed(1)

#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]

#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)

#instantiate ridge model
reg_model= RandomForestRegressor()
reg_model.fit(x_train, y_train)

reg_model.score(x_test,y_test) # Here regression metrics is used to find score. Y
```

Out[56]:
```
0.9124687687774722
```

Mean Absolute Error:

It is the average of absolute(plus or complete) difference between predicted and actual. Rule of Thumb: Have to minimize this value.

Moreover, MAE take differences, add them up, turn them positive and take mean of them.

In [57]:
```
from sklearn.metrics import mean_absolute_error

y_predicted = reg_model.predict(x_test,)
MAE= mean_absolute_error(y_test, y_predicted)

MAE
```

Out[57]:
```
2.2971568627451
```

In [58]:
```
df = pd.DataFrame(data={"actual_values":y_test, "predicted_values": y_predicted}
df["differences"]= df["actual_values"] - df["predicted_values"]
df
```

Out[58]:

	actual_values	predicted_values	differences
307	28.2	29.926	-1.726
343	23.9	27.022	-3.122
47	16.6	20.343	-3.743
67	22.0	20.593	1.407
362	20.8	19.640	1.160
...	...	...	...
92	22.9	23.644	-0.744
224	44.8	42.159	2.641
110	21.7	20.336	1.364
426	10.2	15.765	-5.565
443	15.4	15.709	-0.309

102 rows × 3 columns

collectively differences is 2.3.

Mean Squared Error, MSE:

Takes difference, takes square, add them up and takes average. It will always be far higher than MAV

In [59]:
```
from sklearn.metrics import mean_squared_error

y_pred= reg_model.predict(x_test)
mse= mean_squared_error(y_test, y_pred)
mse
```

Out[59]:
```
8.650483078431385
```

In [60]:
```
#taking MSE manually:
mse_manual= np.square(df["differences"]).mean()
mse_manual
```

Out[60]:
```
8.650483078431384
```

MSE just amplify error by squaring it,

Type *Markdown* and LaTeX: α^2

In [61]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
#np.random.seed(1)
#create data
x=heart_data.drop("DEATH_EVENT", axis=1)
y=heart_data["DEATH_EVENT"]

x_train, x_test , y_train, y_test= train_test_split(x,y, test_size=0.2)
heart_model= RandomForestClassifier().fit(x_train,y_train)

heart_model.score(x_test, y_test)

cross_acc= cross_val_score(heart_model, x, y, cv=5, scoring=None)
cross_acc
```

Out[61]:
```
array([0.45      , 0.85      , 0.83333333, 0.71666667, 0.6779661 ])
```

Using different parameters of scoring: use following link to know more about these parameters: https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

In [62]:
```
cross_acc=cross_val_score(heart_model, x, y, cv=5, scoring=None)
cross_acc.mean()
```

Out[62]:
```
0.7022598870056498
```

In [63]:
```
cross_acc=cross_val_score(heart_model, x, y, cv=5, scoring="accuracy")
cross_acc.mean()
```

Out[63]:
```
0.6889830508474576
```

In [64]:
```
cross_acc=cross_val_score(heart_model, x, y, cv=5, scoring="precision")
cross_acc.mean()
```

```
C:\Users\inn\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[64]:
```
0.4600956937799043
```

In [65]:
```
cross_acc=cross_val_score(heart_model, x, y, cv=5, scoring="f1")
cross_acc.mean()
```

Out[65]:
```
0.489365721997301
```

Scoring Parameters for Regression:

In [66]:
```
#Changing model from regression libraries to improve result.
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

#to prouce same results for following ML
np.random.seed(7)

#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]

#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)

#instantiate ridge model
cl_f= RandomForestRegressor()
cl_f.fit(x_train, y_train)

cl_f.score(x_test,y_test) # Here regression metrics is used to find score. You co
```

Out[66]:
```
0.8092443411593175
```

In [67]:
```
cross_acc= cross_val_score(cl_f, x,y, cv=5, scoring=None)
cross_acc.mean()
```

Out[67]:
```
0.6314044391665344
```

In [68]:
```
cross_acc= cross_val_score(cl_f, x,y,cv=5, scoring="r2")
cross_acc.mean()
```

Out[68]:
```
0.618621010722269
```

In [69]:
```

cross_acc= cross_val_score(cl_f, x,y,cv=5, scoring="neg_mean_absolute_error")
cross_acc.mean()
```

Out[69]:
```
-2.968636594835953
```

In [70]:
```

cross_acc= cross_val_score(cl_f, x,y,cv=5, scoring="neg_mean_squared_error")
cross_acc.mean()
```

Out[70]:
```
-21.8976219086585
```

Metric Function for Classifiction:

In [71]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, f1_score, precision_sco

#to prouce same results for following ML
#np.random.seed(1)
#create data
x=heart_data.drop("DEATH_EVENT", axis=1)
y=heart_data["DEATH_EVENT"]

x_train, x_test , y_train, y_test= train_test_split(x,y, test_size=0.2)
heart_model= RandomForestClassifier().fit(x_train,y_train)

heart_model.score(x_test, y_test)
#predicting
y_predict = heart_model.predict(x_test)

print(f"accuracy score is {accuracy_score(y_test, y_predict)}")
print(f"recall score is {recall_score(y_test, y_predict)}")
print(f"f1 score is {f1_score(y_test, y_predict)}")
print(f"precision score is {precision_score(y_test, y_predict)}")
```

```
accuracy score is 0.85
recall score is 0.6818181818181818
f1 score is 0.7692307692307693
precision score is 0.8823529411764706
```

Evaluation Using Function Regression:

In [72]:
```
#Changing model from regression libraries to improve result.
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import train_test_split

#to prouce same results for following ML
np.random.seed(7)
#create data
x=df_boston.drop("target", axis=1)
y=df_boston["target"]
#split into test and train
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)
#instantiate ridge model
reg_model= RandomForestRegressor()
reg_model.fit(x_train, y_train)
#check the score of model
y_pred= reg_model.predict(x_test)
```

In [73]:
```
print(f"r2_score value     : {r2_score(y_pred, y_test)}")
print(f"mean_absolute_error: {mean_absolute_error(y_pred, y_test)}")
print(f"mean_squared_error : {mean_squared_error(y_pred, y_test)}")
```

```
r2_score value     : 0.7875612536529618
mean_absolute_error: 2.3153627450980383
mean_squared_error : 15.414251382352935
```

Improve Models: 3 Methods:

1 Improving quality and quantity of data.
2 Choosing model from less to more complex/robust e.g from Ridge to RnadomForestRegressor.
3 Improving Model using Hyperparameters.

Hyperparameters:

Parameters used for fine tuning is called. Every model has particular set of parameters.

It can be changed by three ways:

#1 Manually.

#2 Randomly with RandomSearchCV.

#3 Exaustively GridSearchCV.

To apply hyperparameters data is splitted into another third part i.e called Validation Part, and upon VP hyperparameters are applied.

Task 1:

Split data into test, train and validation.

Task 2:

Apply parameters on it.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier()
clf.get_params() # giving all the parameters.
```

Out[74]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [75]:
```
from sklearn.ensemble import RandomForestRegressor
clf1=RandomForestRegressor()
clf1.get_params()
```

Out[75]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [77]:
```
#prediction evaluation function:
from sklearn.metrics import accuracy_score, recall_score, f1_score, precision_sc

def evaluate_hyp_metrics(y_true, y_predicted):
    accuracy=accuracy_score(y_true, y_predicted)
    precision=precision_score(y_true, y_predicted)
    recall=recall_score(y_true, y_predicted)
    f1=f1_score(y_true, y_predicted)

    metrics_dict={"Accuracy": accuracy,
                "precision": precision,
                "recall": recall,
                "f1":f1}

    print(f"Accuracy: {round(accuracy,2)}") # round(): to round up figure.
    print(f"precision:{ round(precision,2)}")
    print(f"recall: {round(recall,2)}")
    print(f"f1: {round(f1,2)}")
    return metrics_dict  #generally no need of ot
```

In [78]:
```
#Shufling whole data
mix_heart_data= heart_data.sample(frac=1)
#Splitting data into three parts i.e test, train and validation.
x=mix_heart_data.drop("DEATH_EVENT", axis=1)
y=mix_heart_data["DEATH_EVENT"]

train_split= round(0.7*len(mix_heart_data))
valid_split= round(train_split + 0.15*len(mix_heart_data))
#test_split= round(0.15*len(mix_heart_data))

x_train, y_train=x[:train_split], y[:train_split]
x_valid, y_valid=x[train_split:valid_split], y[train_split:valid_split]
x_test, y_test=x[valid_split:], y[valid_split:]
```

In [80]:
```
clf=RandomForestClassifier()
clf.fit(x_train,y_train)
#baseline prediction
y1_predicted=clf.predict(x_valid)

first_result=evaluate_hyp_metrics(y_valid, y1_predicted)
first_result
```

```
Accuracy: 0.89
precision:0.89
recall: 0.67
f1: 0.76
```

Out[80]:
```
{'Accuracy': 0.8888888888888888,
 'precision': 0.8888888888888888,
 'recall': 0.6666666666666666,
 'f1': 0.761904761904762}
```

In [81]:
```
clf2=RandomForestClassifier(n_estimators=10)
clf2.fit(x_train,y_train)
#baseline prediction
y2_predicted=clf2.predict(x_valid)

second_result=evaluate_hyp_metrics(y_valid,y2_predicted)
second_result
```

```
Accuracy: 0.89
precision:0.89
recall: 0.67
f1: 0.76
```

Out[81]:
```
{'Accuracy': 0.8888888888888888,
 'precision': 0.8888888888888888,
 'recall': 0.6666666666666666,
 'f1': 0.761904761904762}
```

In [82]:
```
clf3=RandomForestClassifier(n_estimators=10, max_depth=10)
clf3.fit(x_train,y_train)
#baseline prediction
y3_predicted=clf3.predict(x_valid)

third_result=evaluate_hyp_metrics(y_valid,y3_predicted)
third_result
```

```
Accuracy: 0.84
precision:0.86
recall: 0.5
f1: 0.63
```

Out[82]:
```
{'Accuracy': 0.8444444444444444,
 'precision': 0.8571428571428571,
 'recall': 0.5,
 'f1': 0.631578947368421}
```

In [83]:
```
len(mix_heart_data)
```

Out[83]:
```
299
```

In [84]:
```
len(x_train), len(x_valid), len(x_test)
```

Out[84]:
```
(209, 45, 45)
```

In [85]:
```
#Showing parameters.
clf4=RandomForestClassifier()
clf4.get_params()
```

Out[85]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Now, it is very difficult to put every parameter manually and find the results. Therefore, we need to find some method in which every parameter is applied by it self and we retrieve value. That method is RSCV.

Randomized Search Cross Validation (RSCV):

```
Tuning hyperparameters automatically.
```

n_estimators:[10,100,200,500,1000]

max_depth:[None,5,10,20,30]

min_samples_split[2,4,6]

min_samples_leaf[1,2,4]

max_features["auto", "sqrt"]

n_jobs

In [86]:
```
hparams= {"n_estimators":[10,100,200,500,1000],
"max_depth":[None,5,10,20,30],
"min_samples_split":[2,4,6],
"min_samples_leaf":[1,2,4],
"max_features":["auto", "sqrt"]}

x=mix_heart_data.drop("DEATH_EVENT", axis=1)
y=mix_heart_data["DEATH_EVENT"]

x_train, x_test , y_train, y_test= train_test_split(x,y, test_size=0.2)

clf=RandomForestClassifier(n_jobs=1) #n_jobs will make processor to work on full
                                     #It will make program to execute early.
```

In [87]:
```
from sklearn.model_selection import RandomizedSearchCV
rscv_clf=RandomizedSearchCV(estimator=clf, param_distributions=hparams, n_iter=10
rscv_clf.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] n_estimators=500, min_samples_split=6, min_samples_leaf=4, max_features=
auto, max_depth=20

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.

[CV]  n_estimators=500, min_samples_split=6, min_samples_leaf=4, max_features
=auto, max_depth=20, total=   1.4s
[CV] n_estimators=500, min_samples_split=6, min_samples_leaf=4, max_features=
auto, max_depth=20

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.3s remaining:    0.
0s
```

#Here RSCV will take hparams and applied it on estimator. #n_iter: set of combination of values of parameters. #total 10 combination, as 5 layers and 10 combinations for every layer.

In [88]:
```
rscv_clf.best_params_
```

Out[88]:
```
{'n_estimators': 100,
 'min_samples_split': 6,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 5}
```

Now, it is suggesting that if we take following combination of parameters then we will have the best results. such as n_estimator = 10, min_samples_split = 6 and so on.

In [89]:
```
rscv_clf_predict= rscv_clf.predict(x_test)
fourth_result= evaluate_hyp_metrics(y_test, rscv_clf_predict) # evaluate_hyp_metr
```

```
Accuracy: 0.78
precision:0.68
recall: 0.65
f1: 0.67
```

Result is far better than the previous one: third result:

```
Accuracy: 0.82
precision:0.82
recall: 0.6
f1: 0.69
```

Total combinations were 5 *5 *3 *3 *2 = 450 but we check only 10 n_iter=10 out of these. Can we check all of these combinationa? yes, by Grid Search Cross Validation

grid Search Cross Validation:

Tuning the Hyper parameter using Grid Search Cross Validation(GSCV).

As a RSCV has given us most suitable values for every parameter. So, instead of taking all of the values given by we take only thos values which are close to values given by RSCV. if we take all of these then it would become 450*5.

In [90]:
```
from sklearn.model_selection import GridSearchCV


hparams= {"n_estimators":[100,200,500,],
"max_depth":[None,5,10],
"min_samples_split":[2,4],
"min_samples_leaf":[2,4],
"max_features":["auto", "sqrt"]}

gscv_clf=GridSearchCV(estimator=clf, param_grid=hparams, cv=5, verbose=2)
gscv_clf.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
[CV] max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split
=2, n_estimators=100

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.

[CV]  max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_spli
t=2, n_estimators=100, total=   0.3s
[CV] max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split
=2, n_estimators=100

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.
0s
```

In [91]:
```
gscv_clf.best_params_
```

Out[91]:
```
{'max_depth': None,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 100}
```

Result of RSCV, GSCV suggested that n_estimaor=100 would be the best.

{'n_estimators': 200,

'min_samples_split': 2,

'min_samples_leaf': 4,

'max_features': 'auto',

'max_depth': None}

In [92]:
```
gscv_clf_predict= gscv_clf.predict(x_test)
fifth_result= evaluate_hyp_metrics(y_test, gscv_clf_predict) # evaluate_hyp_metri
fifth_result
```

```
Accuracy: 0.78
precision:0.68
recall: 0.65
f1: 0.67
```

Out[92]:
```
{'Accuracy': 0.7833333333333333,
 'precision': 0.6842105263157895,
 'recall': 0.65,
 'f1': 0.6666666666666667}
```

y1

Accuracy: 0.87 precision:0.88 recall: 0.58 f1: 0.7

rsvc

Accuracy: 0.83 precision:0.71 recall: 0.71 f1: 0.71

gsvc

Accuracy: 0.92 precision:0.93 recall: 0.76 f1: 0.84

1sr r

Accuracy: 0.82 precision:0.79 recall: 0.69 f1: 0.73

2 r

Accuracy: 0.8 precision:0.89 recall: 0.5 f1: 0.64

3 r

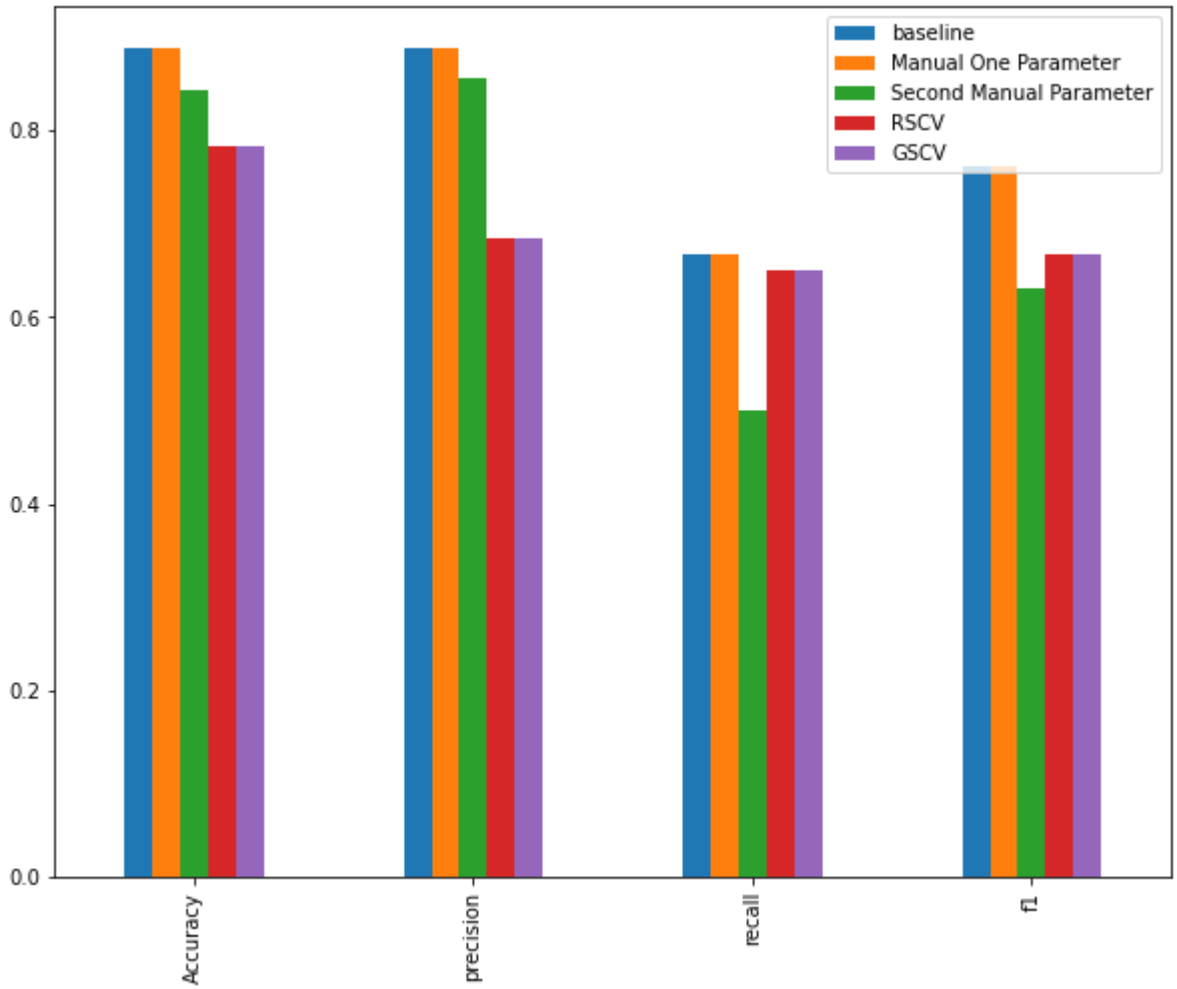
Accuracy: 0.76 precision:0.78 recall: 0.44 f1: 0.56

rsvc

Accuracy: 0.82 precision:0.83 recall: 0.53 f1: 0.65

In [96]:
```
comparison= pd.DataFrame({"baseline":first_result,
            "Manual One Parameter":second_result,
            "Second Manual Parameter": third_result,
            "RSCV":fourth_result,
            "GSCV":fifth_result})
comparison.plot.bar(figsize=(10,8))
```

Out[96]: <AxesSubplot:>



Saving Model:

In [110]:
```
import pickle
pickle.dump(gscv_clf,open("gscv_clf_RFM.pkl","wb"))
```

In [111]:
```
pickled_model=pickle.load(open("gscv_clf_RFM.pkl","rb"))
```

In [112]:
```
predicted_pickled=pickled_model.predict(x_test)
evaluate_hyp_metrics(y_test, predicted_pickled)
```

```
Accuracy: 0.78
precision:0.68
recall: 0.65
f1: 0.67
```

Out[112]:
```
{'Accuracy': 0.7833333333333333,
 'precision': 0.6842105263157895,
 'recall': 0.65,
 'f1': 0.6666666666666667}
```

Another way to save model:

In [122]:
```
from joblib import dump,load
#Saving the model.
dump(gscv_clf, filename="joblib_gscv_model.joblib")
```

Out[122]:
```
['joblib_gscv_model.joblib']
```

In [123]:
```
joblib_model= load(filename="joblib_gscv_model.joblib")
```

In [124]:
```
y_joblib_predicted = joblib_model.predict(x_test)
evaluate_hyp_metrics(y_test, y_joblib_predicted)
```

```
Accuracy: 0.78
precision:0.68
recall: 0.65
f1: 0.67
```

Out[124]:
```
{'Accuracy': 0.7833333333333333,
 'precision': 0.6842105263157895,
 'recall': 0.65,
 'f1': 0.6666666666666667}
```

In [ ]:
```

```