Introduction to Scikit Learn(sKlearn)

what we are going to cover 0. An end to end sckit learn workflow

1. getting to data ready
2. choose the right estimeter/algorithm for our problem
3. fit the model algorithm and use it to make pridiction on our data
4. Evalution a model
5. Improve a model
6. Save and load a traning model
7. putting it all together

0. An end to end Scikit learn Workflow

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
#1 get the data ready
import pandas as pd
heart_disease=pd.read_csv("heart-disease.csv")
heart_disease
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	

303 rows × 14 columns

In [3]:

```
#creqte x( featuring matrix)
x = heart_disease.drop("target", axis=1)
#create y (labels)
y= heart_disease["target"]
```

In [4]:

```
#2. choose the right model and hyperparameter
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
#we will keep the defult hyperparameter
clf.get_params()
```

Out[4]:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 '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 [5]:

```
#3. fit the model to the training data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.2)
```

In [6]:

```
clf.fit(x_train, y_train);
```

In [7]:

```
x_train
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
205	52	1	0	128	255	0	1	161	1	0.0	2	1	3
195	59	1	0	170	326	0	0	140	1	3.4	0	0	3
236	58	1	0	125	300	0	0	171	0	0.0	2	2	3
240	70	1	2	160	269	0	1	112	1	2.9	1	1	3
244	56	1	0	132	184	0	0	105	1	2.1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
289	55	0	0	128	205	0	2	130	1	2.0	1	1	3
116	41	1	2	130	214	0	0	168	0	2.0	1	0	2
79	58	1	2	105	240	0	0	154	1	0.6	1	0	3
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3
78	52	1	1	128	205	1	1	184	0	0.0	2	0	2

242 rows × 13 columns

In [8]:

```
#make a pridition
y_label = clf.predict(np.array([0, 2, 3, 4]))
```

```
C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but RandomFore
stClassifier was fitted with feature names
  warnings.warn(
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[8], line 2
      1 #make a pridition
----> 2 y_label = clf.predict(np.array([0, 2, 3, 4]))

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_forest.py:832, in ForestClassifier.predict(self, X)
    811 def predict(self, X):
    812     """
    813     Predict class for X.
    814 
   (...)
    830         The predicted classes.
    831     """
--> 832     proba = self.predict_proba(X)
    834     if self.n_outputs_ == 1:
    835         return self.classes_.take(np.argmax(proba, axis=1), axis=0)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_forest.py:874, in ForestClassifier.predict_proba(self, X)
    872 check_is_fitted(self)
    873 # Check data
--> 874 X = self._validate_X_predict(X)
    876 # Assign chunk of trees to jobs
    877 n_jobs, _, _ = _partition_estimators(self.n_estimators, self.n_jobs)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_forest.py:605, in BaseForest._validate_X_predict(self, X)
    602 """
    603 Validate X whenever one tries to predict, apply, predict_proba."""
    604 check_is_fitted(self)
--> 605 X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False)
    606 if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.intc):
    607     raise ValueError("No support for np.int64 index based sparse matrices")

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\base.py:577, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, **check_params)
    575     raise ValueError("Validation should be done on X, y or both.")
    576 elif not no_val_X and no_val_y:
--> 577     X = check_array(X, input_name="X", **check_params)
    578     out = X
    579 elif no_val_X and not no_val_y:

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\utils\validation.py:879, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    877     # If input is 1D raise error
    878     if array.ndim == 1:
--> 879         raise ValueError(
    880             "Expected 2D array, got 1D array instead:\narray={}.\n"
```

In [10]:

```
y_preds = clf.predict(x_test)
y_preds
```

Out[10]:

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

In [11]:

```
y_test
```

Out[11]:

```
163    1
104    1
203    0
74     1
      ..
84     1
70     0
232    0
188    0
254    0
Name: target, Length: 61, dtype: int64
```

```
    881             "Reshape your data either using array.reshape(-1, 1) i
f "
    882             "your data has a single feature or array.reshape(1, -
1) "
    883             "if it contains a single sample.".format(array)
    884         )
    886 if dtype_numeric and array.dtype.kind in "USV":
    887     raise ValueError(
    888         "dtype='numeric' is not compatible with arrays of bytes/st
rings."
    889         "Convert your data to numeric values explicitly instead."
    890     )

ValueError: Expected 2D array, got 1D array instead:
array=[0.77049180327868].
Reshape your data either using array.reshape(-1, 1) if your data has a sin
gle feature or array.reshape(1, -1) if it contains a single sample.
```

In [12]:

```
clf.score(x_train, y_train)
```

Out[12]:

```
1.0
```

In [13]:

```
clf.score(x_test, y_test)
```

Out[13]:

```
0.7704918032786885
```

In [14]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.86      0.71      0.77        34
           1       0.70      0.85      0.77        27

    accuracy                           0.77        61
   macro avg       0.78      0.78      0.77        61
weighted avg       0.79      0.77      0.77        61
```

In [15]:

```
confusion_matrix(y_test, y_preds)
```

Out[15]:

```
array([[24, 10],
       [ 4, 23]], dtype=int64)
```

In [16]:

```
accuracy_score(y_test, y_preds)
```

Out[16]:

```
0.7704918032786885
```

In [17]:

```
# Improve a model
#Try diffrent amount of n estimetrs
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    clf = RandomForestClassifier(n_estimators=i).fit(x_train, y_train)
    print(f"Model accuracy on test set: {clf.score(x_test, y_test) * 100:2f}%")
    print("")
```

```
Trying model with 10 estimators...
Model accuracy on test set: 73.770492%

Trying model with 20 estimators...
Model accuracy on test set: 78.688525%

Trying model with 30 estimators...
Model accuracy on test set: 77.049180%

Trying model with 40 estimators...
Model accuracy on test set: 75.409836%

Trying model with 50 estimators...
Model accuracy on test set: 81.967213%

Trying model with 60 estimators...
Model accuracy on test set: 77.049180%

Trying model with 70 estimators...
Model accuracy on test set: 75.409836%

Trying model with 80 estimators...
Model accuracy on test set: 80.327869%

Trying model with 90 estimators...
Model accuracy on test set: 73.770492%
```

In [18]:

```
#6 .  save a model and Load it
import pickle
pickle.dump(clf, open ("random_forest_model_1.pkl", "wb"))
```

In [19]:

```
loaded_model = pickle.load(open("random_forest_model_1.pkl", "rb"))
loaded_model.score(x_test, y_test)
```

Out[19]:

```
0.7377049180327869
```

1. Getting our data ready to be used with machine learning

three main things we have to do

1. split the data into features and labels(usually x and y)
2. filling (also called imputing) or disregarding missing value
3. converting non numerical value to numerical value (also called feturing encoding)

In [20]:

```
heart_disease.head()
```

Out[20]:

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

In [21]:

```
x= heart_disease.drop("target", axis=1)
x.head()
```

Out[21]:

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

In [22]:

```
y= heart_disease["target"]
y.head()
```

Out[22]:

```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [23]:

```
#SPLIT THE DATA INTO TRAINnig and test sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [24]:

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[24]:

```
((242, 13), (61, 13), (242,), (61,))
```

In [25]:

```
x.shape[0]*0.8
```

Out[25]:

```
242.4
```

In [26]:

```
242+61
```

Out[26]:

```
303
```

In [27]:

```
len(heart_disease)
```

Out[27]:

```
303
```

In [28]:

```
car_sales = pd.read_csv("https://raw.githubusercontent.com/mrdbourke/zero-to-mastery-ml/
```

In [29]:

```
car_sales.head()
```

Out[29]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431	4	15323
1	BMW	Blue	192714	5	19943
2	Honda	White	84714	4	28343
3	Toyota	White	154365	4	13434
4	Nissan	Blue	181577	3	14043

In [30]:

```
len(car_sales)
```

Out[30]:

```
1000
```

In [31]:

```
#split into x/y
x= car_sales.drop("Price", axis=1)
y= car_sales["Price"]
#split into training and test
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2)
```

In [32]:

```
#build machine learning moadel
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[32], line 4
      2 from sklearn.ensemble import RandomForestRegressor
      3 model = RandomForestRegressor()
----> 4 model.fit(x_train, y_train)
      5 model.score(x_test, y_test)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_f
orest.py:331, in BaseForest.fit(self, X, y, sample_weight)
    329 if issparse(y):
    330     raise ValueError("sparse multilabel-indicator for y is not sup
ported.")
--> 331 X, y = self._validate_data(
    332     X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    333 )
    334 if sample_weight is not None:
    335     sample_weight = _check_sample_weight(sample_weight, X)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\base.py:59
6, in BaseEstimator._validate_data(self, X, y, reset, validate_separately,
**check_params)
    594         y = check_array(y, input_name="y", **check_y_params)
    595     else:
--> 596         X, y = check_X_y(X, y, **check_params)
    597     out = X, y
    599 if not no_val_X and check_params.get("ensure_2d", True):

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\utils\valid
ation.py:1074, in check_X_y(X, y, accept_sparse, accept_large_sparse, dtyp
e, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensur
e_min_samples, ensure_min_features, y_numeric, estimator)
   1069         estimator_name = _check_estimator_name(estimator)
   1070     raise ValueError(
   1071         f"{estimator_name} requires y to be passed, but the target
y is None"
   1072     )
-> 1074 X = check_array(
   1075     X,
   1076     accept_sparse=accept_sparse,
   1077     accept_large_sparse=accept_large_sparse,
   1078     dtype=dtype,
   1079     order=order,
   1080     copy=copy,
   1081     force_all_finite=force_all_finite,
   1082     ensure_2d=ensure_2d,
   1083     allow_nd=allow_nd,
   1084     ensure_min_samples=ensure_min_samples,
   1085     ensure_min_features=ensure_min_features,
   1086     estimator=estimator,
   1087     input_name="X",
   1088 )
   1090 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, es
timator=estimator)
   1092 check_consistent_length(X, y)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\utils\valid
ation.py:856, in check_array(array, accept_sparse, accept_large_sparse, dt
ype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_sample
```

```
s, ensure_min_features, estimator, input_name)
    854         array = array.astype(dtype, casting="unsafe", copy=False)
    855     else:
--> 856         array = np.asarray(array, order=order, dtype=dtype)
    857 except ComplexWarning as complex_warning:
    858     raise ValueError(
    859         "Complex data not supported\n{}\n".format(array)
    860     ) from complex_warning

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\generi
c.py:2070, in NDFrame.__array__(self, dtype)
   2069 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndar
ray:
-> 2070     return np.asarray(self._values, dtype=dtype)

ValueError: could not convert string to float: 'Toyota'
```

In [34]:

```
 #turn the catogery into number
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
categorical_feature =["Make", "Colour", "Doors"]
one_hot=OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                   one_hot,
                                   categorical_feature)],
                                   remainder="passthrough")

transformed_x =transformer.fit_transform(x)
transformed_x
```

Out[34]:

```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 3.54310e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 1.92714e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 8.47140e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 6.66040e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.15883e+05],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.48360e+05]])
```

In [35]:

```
pd.DataFrame(transformed_x)
```

Out[35]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	35431.0
1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	192714.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	84714.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	154365.0
4	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	181577.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	35820.0
996	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	155144.0
997	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	66604.0
998	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	215883.0
999	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	248360.0

1000 rows × 13 columns

In [36]:

```
dummies =pd.get_dummies(car_sales[["Make", "Colour", "Doors"]])
dummies
```

Out[36]:

	Doors	Make_BMW	Make_Honda	Make_Nissan	Make_Toyota	Colour_Black	Colour_Blu
0	4	0	1	0	0	0	
1	5	1	0	0	0	0	
2	4	0	1	0	0	0	
3	4	0	0	0	1	0	
4	3	0	0	1	0	0	
...	...	...	...	...	...	...	
995	4	0	0	0	1	1	
996	3	0	0	1	0	0	
997	4	0	0	1	0	0	
998	4	0	1	0	0	0	
999	4	0	0	0	1	0	

1000 rows × 10 columns

In [37]:

```
#lets refain the model
np.random.seed(42)
x_train, x_test, y_train, y_test =train_test_split(transformed_x,
                                                    y,
                                                    test_size=0.2)
model.fit(x_train, y_train)
```

Out[37]:

```
RandomForestRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [38]:

```
x.head()
```

Out[38]:

	Make	Colour	Odometer (KM)	Doors
0	Honda	White	35431	4
1	BMW	Blue	192714	5
2	Honda	White	84714	4
3	Toyota	White	154365	4
4	Nissan	Blue	181577	3

In [39]:

```
model.score(x_test, y_test)
```

Out[39]:

```
0.3235867221569877
```

what if there were missing values?

1. fill them with some value(also known as imputation)
2. Remove the samples with data altogether

In [40]:

```
#import car sales missing data
car_sales_missing=pd.read_csv("car_sales_extented.csv")
car_sales_missing.head()
```

Out[40]:

	Unnamed: 0	Make	Colour	Odometer (KM)	Doors	Price
0	0	Honda	White	35431	4	15323
1	1	BMW	Blue	192714	5	19943
2	2	Honda	White	84714	4	28343
3	3	Toyota	White	154365	4	13434
4	4	Nissan	Blue	181577	3	14043

In [41]:

```
car_sales_missing.isna().sum()
```

Out[41]:

```
Unnamed: 0       0
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price            0
dtype: int64
```

In [42]:

```
x=car_sales_missing.drop("Price", axis=1)
y = car_sales_missing["Price"]
```

In [43]:

```
#lets try to convert our data to numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
categorical_feature =["Make", "Colour", "Doors"]
one_hot=OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                   one_hot,
                                   categorical_feature)],
                                   remainder="passthrough")

transformed_x =transformer.fit_transform(car_sales_missing)
transformed_x
```

Out[43]:

```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        3.54310e+04, 1.53230e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        1.92714e+05, 1.99430e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 2.00000e+00,
        8.47140e+04, 2.83430e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 9.97000e+02,
        6.66040e+04, 3.15700e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 9.98000e+02,
        2.15883e+05, 4.00100e+03],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 9.99000e+02,
        2.48360e+05, 1.27320e+04]])
```

Option 2: Fill missing values with scikite learn

In [44]:

```
car_sale_missing_missing=pd.read_csv("car_sales_extented.csv")
```

In [45]:

```
car_sale_missing_missing.isna().sum()
```

Out[45]:

```
Unnamed: 0       0
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price            0
dtype: int64
```

In [46]:

```
#Drop the rows with no lable
car_sale_missing_missing.dropna(subset=["Price"], inplace=True)
car_sale_missing_missing.isna().sum()
```

Out[46]:

```
Unnamed: 0       0
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price            0
dtype: int64
```

In [47]:

```
#split into x and y
x=car_sale_missing_missing.drop("Price", axis=1)
y=car_sale_missing_missing["Price"]
```

In [48]:

```
#fill missing value with sachite learn
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
#fill the catogerical value with missing numerical value with mean
cat_imputer = SimpleImputer(strategy="constant", fill_value="missing")
door_imputer =SimpleImputer(strategy="constant", fill_value=4)
num_imputer =SimpleImputer(strategy="mean")
#define columns
cat_features=["Make", "Colour"]
door_feature=["Doors"]
num_features=["Odometer (KM)"]
#create an imputer(Something that fills missing data)
imputer =ColumnTransformer([
    ("cat_imputer", cat_imputer, cat_features),
    ("door_imputer", door_imputer, door_feature),
    ("num_imputer", num_imputer, num_features)
])
#transfer the data
filled_x =imputer.fit_transform(x)
filled_x
```

Out[48]:

```
array([['Honda', 'White', 4, 35431.0],
       ['BMW', 'Blue', 5, 192714.0],
       ['Honda', 'White', 4, 84714.0],
       ...,
       ['Nissan', 'Blue', 4, 66604.0],
       ['Honda', 'White', 4, 215883.0],
       ['Toyota', 'Blue', 4, 248360.0]], dtype=object)
```

In [49]:

```
car_sales_filled=pd.DataFrame(filled_x,
                              columns=["Make", "Colour", "Doors", "Odometer (KM)"])
car_sales_filled.head()
```

Out[49]:

	Make	Colour	Doors	Odometer (KM)
0	Honda	White	4	35431.0
1	BMW	Blue	5	192714.0
2	Honda	White	4	84714.0
3	Toyota	White	4	154365.0
4	Nissan	Blue	3	181577.0

In [50]:

```
car_sales_filled.isna().sum()
```

Out[50]:

```
Make             0
Colour           0
Doors            0
Odometer (KM)    0
dtype: int64
```

In [51]:

```
#lets try to convert our data to numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
categorical_feature =["Make", "Colour", "Doors"]
one_hot=OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                   one_hot,
                                   categorical_feature)],
                                   remainder="passthrough")

transformed_x =transformer.fit_transform(car_sales_filled)
transformed_x
```

Out[51]:

```
array([[0.0, 1.0, 0.0, ..., 1.0, 0.0, 35431.0],
       [1.0, 0.0, 0.0, ..., 0.0, 1.0, 192714.0],
       [0.0, 1.0, 0.0, ..., 1.0, 0.0, 84714.0],
       ...,
       [0.0, 0.0, 1.0, ..., 1.0, 0.0, 66604.0],
       [0.0, 1.0, 0.0, ..., 1.0, 0.0, 215883.0],
       [0.0, 0.0, 0.0, ..., 1.0, 0.0, 248360.0]], dtype=object)
```

In [52]:

```
#now we got our data as number and fileld
#let fit a model
np.random.seed(42)
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =train_test_split(transformed_x,
                                                    y,
                                                   test_size=0.2)
model=RandomForestRegressor(n_estimators=100)
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[52]:

```
0.3235867221569877
```

2. chhosing the right estimeter for your problem

some thing to note • Sklearn refers to machine learning models, algorithms as estimators.

• Classification problem - predicting a category (heart disease or not) ■ Sometimes you'll see clf (short for classifier) used as a classification estimator

• Regression problem - predicting a number (selling price of a car)

2.1 picking a machine learning model for regression problem

let's us use California housing dataset

In [53]:

```
#Get California Housing dataSet
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
housing
```

Out[53]:

```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,    2.55
555556,
          37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, ...,    2.10984183,
          37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, ...,    2.80225989,
          37.85      , -122.24      ],
        ...,
        [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351 ,
          39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, ...,    2.12320917,
          39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, ...,    2.61698113,
          39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing dataset\n
--------------------------\n\n**Data Set Characteristics:**\n\n    :Number
of Instances: 20640\n\n    :Number of Attributes: 8 numeric, predictive at
tributes and the target\n\n    :Attribute Information:\n        - MedInc
median income in block group\n        - HouseAge      median house age in
block group\n        - AveRooms      average number of rooms per household
\n        - AveBedrms     average number of bedrooms per household\n
- Population    block group population\n        - AveOccup      average nu
mber of household members\n        - Latitude      block group latitude\n
- Longitude     block group longitude\n\n    :Missing Attribute Values: No
ne\n\nThis dataset was obtained from the StatLib repository.\nhttps://www.
dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe target variable is
the median house value for California districts,\nexpressed in hundreds of
thousands of dollars ($100,000).\n\nThis dataset was derived from the 1990
U.S. census, using one row per census\nblock group. A block group is the s
mallest geographical unit for which the U.S.\nCensus Bureau publishes samp
le data (a block group typically has a population\nof 600 to 3,000 peopl
e).\n\nAn household is a group of people residing within a home. Since the
average\nnumber of rooms and bedrooms in this dataset are provided per hou
sehold, these\ncolumns may take surpinsingly large values for block groups
with few households\nand many empty houses, such as vacation resorts.\n\nI
t can be downloaded/loaded using the\n:func:`sklearn.datasets.fetch_califo
rnia_housing` function.\n\n.. topic:: References\n\n    - Pace, R. Kelley
and Ronald Barry, Sparse Spatial Autoregressions,\n      Statistics and Pr
obability Letters, 33 (1997) 291-297\n'}
```

In [54]:

```
housing_df =pd.DataFrame(housing["data"], columns=housing["feature_names"])
housing_df
```

Out[54]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.
...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.

20640 rows × 8 columns

In [55]:

```
housing_df["target"] = housing["target"]
housing_df.head()
```

Out[55]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [56]:

```
housing_df =housing_df.drop("MedHouseVal", axis=1)
```

```
---------------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
Cell In[56], line 1
----> 1 housing_df =housing_df.drop("MedHouseVal", axis=1)

File ~\Machine_learning\project1\env\lib\site-packages\pandas\util\_decora
tors.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.
wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\frame.p
y:5396, in DataFrame.drop(self, labels, axis, index, columns, level, inpla
ce, errors)
   5248 @deprecate_nonkeyword_arguments(version=None, allowed_args=["sel
f", "labels"])
   5249 def drop(  # type: ignore[override]
   5250     self,
   (...)
   5257     errors: IgnoreRaise = "raise",
   5258 ) -> DataFrame | None:
   5259     """
   5260     Drop specified labels from rows or columns.
   5261 
   (...)
   5394             weight  1.0     0.8
   5395     """
-> 5396     return super().drop(
   5397         labels=labels,
   5398         axis=axis,
   5399         index=index,
   5400         columns=columns,
   5401         level=level,
   5402         inplace=inplace,
   5403         errors=errors,
   5404     )

File ~\Machine_learning\project1\env\lib\site-packages\pandas\util\_decora
tors.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.
wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\generi
c.py:4505, in NDFrame.drop(self, labels, axis, index, columns, level, inpl
ace, errors)
   4503 for axis, labels in axes.items():
   4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=err
```

```
ors)
   4507 if inplace:
   4508     self._update_inplace(obj)

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\generi
c.py:4546, in NDFrame._drop_axis(self, labels, axis, level, errors, only_s
lice)
   4544         new_axis = axis.drop(labels, level=level, errors=errors)
   4545     else:
-> 4546         new_axis = axis.drop(labels, errors=errors)
   4547     indexer = axis.get_indexer(new_axis)
   4549 # Case for non-unique axis
   4550 else:

File ~\Machine_learning\project1\env\lib\site-packages\pandas\core\indexes
\base.py:6977, in Index.drop(self, labels, errors)
   6975 if mask.any():
   6976     if errors != "ignore":
-> 6977         raise KeyError(f"{list(labels[mask])} not found in axis")
   6978     indexer = indexer[~mask]
   6979 return self.delete(indexer)

KeyError: "['MedHouseVal'] not found in axis"
```

In [58]:

```
housing_df
```

Out[58]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.
...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.

20640 rows × 9 columns

In [59]:

```
# import algorithm
from sklearn.linear_model import Ridge
#setup random seed
np.random.seed(42)
#create the data
x=housing_df.drop("target", axis=1)
y=housing_df["target"] # median house price 10000
#split inton train and test sets
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.2)
#Instantiate and fit the model (on training sets)
model =Ridge()
model.fit(x_train, y_train)
#chech the score of the model(on the test set)
model.score(x_test, y_test)
```

Out[59]:

```
0.5758549611440128
```

what if rigid model didn't work on the score didn't fit our needs well we can try diffrent model let's try ensable model

In [60]:

```
#import the RandomForestRegressor model class from ensemble moudle
from sklearn.ensemble import RandomForestRegressor
#set up random seed
np.random.seed(42)
#create the data
x=housing_df.drop("target", axis=1)
y = housing_df["target"]
#split into train and test sets
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.2)
#create random forest model
model =RandomForestRegressor()
model.fit(x_train, y_train)
#check the score of the model
model.score(x_test, y_test)
```

Out[60]:

```
0.8065734772187598
```

choosing an estimater for a classification problem

heart_disease = pd.read_csv("heart-disease.csv") heart_disease.head()

In [61]:

```
len (heart_disease)
```

Out[61]:

```
303
```

Consulting the map and it says to try LinerSvc

In [62]:

```
#import the LinearSvc estimators
from sklearn.svm import LinearSVC
#set up random seed
np.random.seed(42)
#make the data
x = heart_disease.drop("target", axis=1)
y =  heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate LinearSVC
clf = LinearSVC()
clf.fit(x_train, y_train)
#evaluate the LinearSVc
clf.score(x_test, y_test)
```

```
C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn\svm
\_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase
the number of iterations.
  warnings.warn(
```

Out[62]:

```
0.8688524590163934
```

In [63]:

```
#import the RandomForestClasifier estimators class
from sklearn.ensemble import RandomForestClassifier
#set up random seed
np.random.seed(42)
#make the data
x = heart_disease.drop("target", axis=1)
y =  heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate random forest classifier
clf = RandomForestClassifier()
clf.fit(x_train, y_train)
#evaluate the random forest classifier
clf.score(x_test, y_test)
```

Out[63]:

```
0.8524590163934426
```

Tidbit: 1. if you have structured data used ensemble methods 2. if you have unstructured data use deep learning or transfer lerning

In [64]:

```
heart_disease
```

Out[64]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	

303 rows × 14 columns

2. Fit the model/ algorithm on our data and use it to make prediction

3.1 Fitting the model to the data x = feartures variable, data y= labels target

In [65]:

```
#import the RandomForestClasifier estimators class
from sklearn.ensemble import RandomForestClassifier
#set up random seed
np.random.seed(42)
#make the data
x = heart_disease.drop("target", axis=1)
y =  heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate random forest classifier
clf = RandomForestClassifier(n_estimators=100)
#fit the model to the data(training the machine learning model)
clf.fit(x_train, y_train)
#evaluate the random forest classifier(use the patterns the model has)
clf.score(x_test, y_test)
```

Out[65]:

```
0.8524590163934426
```

In [66]:

```
x.head()
```

Out[66]:

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

In [67]:

```
y.tail()
```

Out[67]:

```
298    0
299    0
300    0
301    0
302    0
Name: target, dtype: int64
```

fit the model/algorithm on our data and use it make prediction

3.1 Fiting the model to the datAa x = Features features variable data y= label, target variable

In [68]:

```
#import the RandomForestClasifier estimators class
from sklearn.ensemble import RandomForestClassifier
#set up random seed
np.random.seed(42)
#make the data
x = heart_disease.drop("target", axis=1)
y =  heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate random forest classifier
clf = RandomForestClassifier(n_estimators=100)
#fit the model to the data(training machine laerrning modcel)
clf.fit(x_train, y_train)
#evaluate the random forest classifier(use the patterns the model has learned)
clf.score(x_test, y_test)
```

Out[68]:

```
0.8524590163934426
```

3.2 make predication using a machine learning

2 ways to make predicton

1. predict()
2. predict_proba()

In [69]:

```
#use a trsined model to make prediction
clf.predict(np.array([1, 7, 8, 3, 4])) #this doesn't work
```

```
C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn\bas
e.py:450: UserWarning: X does not have valid feature names, but RandomFore
stClassifier was fitted with feature names
  warnings.warn(
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[69], line 2
      1 #use a trsined model to make prediction
----> 2 clf.predict(np.array([1, 7, 8, 3, 4]))

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_forest.py:832, in ForestClassifier.predict(self, X)
    811 def predict(self, X):
    812     """
    813     Predict class for X.
    814 
   (...)
    830         The predicted classes.
    831     """
--> 832     proba = self.predict_proba(X)
    834     if self.n_outputs_ == 1:
    835         return self.classes_.take(np.argmax(proba, axis=1), axis=0)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_forest.py:874, in ForestClassifier.predict_proba(self, X)
    872 check_is_fitted(self)
    873 # Check data
--> 874 X = self._validate_X_predict(X)
    876 # Assign chunk of trees to jobs
    877 n_jobs, _, _ = _partition_estimators(self.n_estimators, self.n_jobs)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\ensemble\_forest.py:605, in BaseForest._validate_X_predict(self, X)
    602 """
    603 Validate X whenever one tries to predict, apply, predict_proba."""
    604 check_is_fitted(self)
--> 605 X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False)
    606 if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.intc):
    607     raise ValueError("No support for np.int64 index based sparse matrices")

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\base.py:577, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, **check_params)
    575     raise ValueError("Validation should be done on X, y or both.")
    576 elif not no_val_X and no_val_y:
--> 577     X = check_array(X, input_name="X", **check_params)
    578     out = X
    579 elif no_val_X and not no_val_y:

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\utils\validation.py:879, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    877     # If input is 1D raise error
    878     if array.ndim == 1:
--> 879         raise ValueError(
    880             "Expected 2D array, got 1D array instead:\narray={}.\n"
```

In [71]:

```
x_test.head()
```

Out[71]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2

In [72]:

```
clf.predict(x_test)
```

Out[72]:

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

```
    881             "Reshape your data either using array.reshape(-1, 1) i
    882             "your data has a single feature or array.reshape(1, -1) "
    883             "if it contains a single sample.".format(array)
    884         )
    886     if dtype_numeric and array.dtype.kind in "USV":
    887         raise ValueError(
    888             "dtype='numeric' is not compatible with arrays of bytes/strings."
    889             "Convert your data to numeric values explicitly instead."
    890         )

ValueError: Expected 2D array, got 1D array instead:
array=[1. 7. 8. 3. 4.].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) if it contains a single sample.
```

In [73]:

```
np.array(y_test)
```

Out[73]:

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

In [74]:

```
#compare prediction to truth labels to evaluate the model
y_preds =clf.predict(x_test)
np.mean(y_preds ==y_test)
```

Out[74]:

```
0.8524590163934426
```

In [75]:

```
clf.score(x_test, y_test)
#see same value mean our prediction is right
```

Out[75]:

```
0.8524590163934426
```

In [76]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_preds)
#another way
```

Out[76]:

```
0.8524590163934426
```

In [77]:

```
#Make prediction using with predict_proba()
#predict_proba return probablites of classification lebel
clf.predict_proba(x_test)
#it give probablity
```

Out[77]:

```
array([[0.89, 0.11],
       [0.49, 0.51],
       [0.43, 0.57],
       [0.84, 0.16],
       [0.18, 0.82],
       [0.14, 0.86],
       [0.36, 0.64],
       [0.95, 0.05],
       [0.99, 0.01],
       [0.47, 0.53],
       [0.26, 0.74],
       [0.7 , 0.3 ],
       [0.11, 0.89],
       [0.95, 0.05],
       [0.03, 0.97],
       [0.02, 0.98],
       [0.01, 0.99],
       [0.84, 0.16],
       [0.95, 0.05],
       [0.98, 0.02],
       [0.51, 0.49],
       [0.89, 0.11],
       [0.38, 0.62],
       [0.29, 0.71],
       [0.26, 0.74],
       [0.34, 0.66],
       [0.2 , 0.8 ],
       [0.22, 0.78],
       [0.83, 0.17],
       [0.15, 0.85],
       [0.94, 0.06],
       [0.92, 0.08],
       [0.96, 0.04],
       [0.62, 0.38],
       [0.46, 0.54],
       [0.89, 0.11],
       [0.44, 0.56],
       [0.16, 0.84],
       [0.33, 0.67],
       [0.08, 0.92],
       [0.13, 0.87],
       [0.17, 0.83],
       [0.38, 0.62],
       [0.32, 0.68],
       [0.77, 0.23],
       [0.39, 0.61],
       [0.  , 1.  ],
       [0.83, 0.17],
       [0.97, 0.03],
       [0.85, 0.15],
       [0.8 , 0.2 ],
       [0.25, 0.75],
       [0.25, 0.75],
       [0.87, 0.13],
       [0.93, 0.07],
       [0.71, 0.29],
       [0.01, 0.99],
       [0.87, 0.13],
```

In [78]:

```
#let's predict() on same data
clf.predict(x_test[:5])#it give single value
#in above if value of right is biger then give 1 if not then 0
```

Out[78]:

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

In [79]:

```
heart_disease["target"].value_counts()
```

Out[79]:

```
1    165
0    138
Name: target, dtype: int64
```

predict() can also be used for regression model

In [80]:

```
housing_df.head()
```

Out[80]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

```
       [1.  , 0.  ],
       [0.86, 0.14]])
```

In [81]:

```
from sklearn.ensemble import RandomForestRegressor
np.random.seed(42)
#create the data
x = housing_df.drop("target", axis=1)
y= housing_df["target"]
#split into training and test sets
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.2)
#create model instance
model =RandomForestRegressor()
#fit the model to the data
model.fit(x_train, y_train)
#make prediction
y_preds =model.predict(x_test)
```

In [82]:

```
y_preds[:10]
```

Out[82]:

```
array([0.49384  , 0.75494  , 4.9285964, 2.54316  , 2.33176  , 1.6525301,
       2.34323  , 1.66182  , 2.47489  , 4.8344779])
```

In [83]:

```
np.array([y_test[:10]])
#similar result
```

```
C:\Users\alokr\AppData\Local\Temp\ipykernel_16480\3867802618.py:1: FutureW
arning: The behavior of `series[i:j]` with an integer-dtype index is depre
cated. In a future version, this will be treated as *label-based* indexin
g, consistent with e.g. `series[i]` lookups. To retain the old behavior, u
se `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  np.array([y_test[:10]])
```

Out[83]:

```
array([[0.477  , 0.458  , 5.00001, 2.186  , 2.78   , 1.587  , 1.982  ,
        1.575  , 3.4    , 4.466  ]])
```

In [84]:

```
len(y_preds)
```

Out[84]:

```
4128
```

In [85]:

```
len(y_test)
```

Out[85]:

```
4128
```

In [86]:

```
#compare the prediction to truth diffrence
```

In [87]:

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_preds)
```

Out[87]:

```
0.32659871732073664
```

In [88]:

```
housing_df["target"]
```

Out[88]:

```
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         ...
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: target, Length: 20640, dtype: float64
```

4.0 Evaluting a machine learning model

Three ways to envaluate sackite learn model

1. Estimator built-in score() method
2. The scoring parameter
3. problem specific metrics function

4.1 Evaluting a model with the score method

In [89]:

```
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)
#make the data
x = heart_disease.drop("target", axis=1)
y =  heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate random forest classifier
clf = RandomForestClassifier(n_estimators=100)
#fit the model to the data(training machine laerrning modcel)
clf.fit(x_train, y_train)
```

Out[89]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [90]:

```
#The highest value for the score() is 1.0 the lowewst is 0.0
clf.score(x_train, y_train)
```

Out[90]:

```
1.0
```

In [91]:

```
clf.score(x_test, y_test)
```

Out[91]:

```
0.8524590163934426
```

In [92]:

```
#let's use score() on our regression probleam
from sklearn.ensemble import RandomForestRegressor
np.random.seed(42)
#create the data
x = housing_df.drop("target", axis=1)
y= housing_df["target"]
#split into training and test sets
x_train, x_test, y_train, y_test =train_test_split(x, y, test_size=0.2)
#create model instance
model =RandomForestRegressor(n_estimators=100)

#fit the model to the data
model.fit(x_train, y_train)
```

Out[92]:

```
RandomForestRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [93]:

```
model.score(x_test, y_test)
```

Out[93]:

```
0.8065734772187598
```

4.2 Evaluting a model using the scoring parameter

In [94]:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)
#make the data
x = heart_disease.drop("target", axis=1)
y =  heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate random forest classifier
clf = RandomForestClassifier(n_estimators=100)
#fit the model to the data(training machine laerrning modcel)
clf.fit(x_train, y_train)
```

Out[94]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [95]:

```
clf.score(x_test, y_test)
```

Out[95]:

```
0.8524590163934426
```

In [96]:

```
cross_val_score(clf, x, y, cv=5)
#it give five diffrent val coz it take 20 percent 5 time
```

Out[96]:

```
array([0.81967213, 0.86885246, 0.81967213, 0.78333333, 0.76666667])
```

In [97]:

```
cross_val_score(clf, x, y, cv=10)
```

Out[97]:

```
array([0.90322581, 0.80645161, 0.87096774, 0.9       , 0.86666667,
       0.8       , 0.73333333, 0.86666667, 0.73333333, 0.8       ])
```

In [98]:

```
np.random.seed(42)
#single training and test split
clf_single_score =clf.score(x_test, y_test)

#Take the mean of 5 fold cross validation score
clf_cross_val_score =np.mean(cross_val_score(clf, x, y, cv=5))

#comparing the two
clf_single_score, clf_cross_val_score
```

Out[98]:

```
(0.8524590163934426, 0.8248087431693989)
```

In [ ]:

```

```

In [99]:

```
#scoring parameter set to none by defult
cross_val_score(clf, x, y, cv=5, scoring=None)
```

Out[99]:

```
array([0.78688525, 0.86885246, 0.80327869, 0.78333333, 0.76666667])
```

4.2.1 Classification model evaluation metrics

1. Accuracy
2. Area under ROC curve
3. Confusion Matrics
4. Classification report

In [100]:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)
x= heart_disease.drop("target", axis=1)
y=heart_disease["target"]

clf=RandomForestClassifier()
cross_val_score =cross_val_score(clf, x, y, cv=5)
```

In [101]:

```
np.mean(cross_val_score)
```

Out[101]:

```
0.8248087431693989
```

Area under the reciver operating characterstic curve(auc/ roc)

1. Area under curve (AUC)
2. ROC curve Roc curves are comparision of a model true positive rate(tpr) varse model false positive (fpr) • True positive = model predicts 1 when truth is 1

• False positive = model predicts 1 when truth is 0

• True negative = model predicts 0 when truth is 0 False negative = model predicts 0 when truth is 1

In [102]:

```
#create test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [103]:

```
from sklearn.metrics import roc_curve
#fit the classifier
clf.fit(x_train, y_train)
#make prediction with problities
y_probs= clf.predict_proba(x_test)
y_probs[:10], len(y_probs)
```

Out[103]:

```
(array([[0.51, 0.49],
        [0.17, 0.83],
        [0.51, 0.49],
        [0.72, 0.28],
        [0.43, 0.57],
        [0.12, 0.88],
        [0.3 , 0.7 ],
        [0.97, 0.03],
        [0.15, 0.85],
        [0.4 , 0.6 ]]),
 61)
```

In [104]:

```
y_probs_positive = y_probs[:, 1]
y_probs_positive[:10]
```

Out[104]:

```
array([0.49, 0.83, 0.49, 0.28, 0.57, 0.88, 0.7 , 0.03, 0.85, 0.6 ])
```

In [105]:

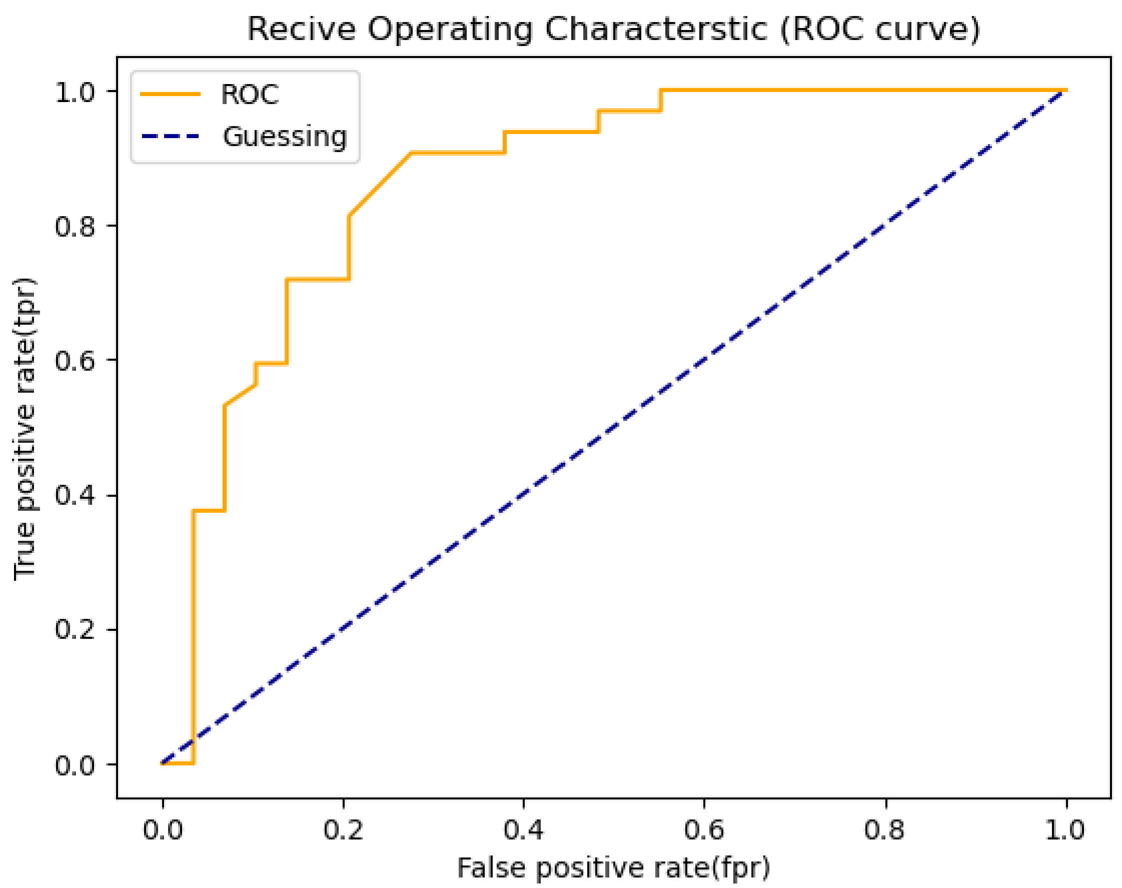
```
#caculate fpr, tpr, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)

#check the false positive rates
fpr
```

Out[105]:

```
array([0.        , 0.03448276, 0.03448276, 0.03448276, 0.03448276,
       0.03448276, 0.03448276, 0.06896552, 0.06896552, 0.06896552,
       0.10344828, 0.10344828, 0.13793103, 0.13793103, 0.13793103,
       0.20689655, 0.20689655, 0.20689655, 0.27586207, 0.37931034,
       0.37931034, 0.48275862, 0.48275862, 0.55172414, 0.55172414,
       1.        ])
```

In [106]:

```
#Create a funtion for ploting curve
import matplotlib.pyplot as plt
def plot_roc_curve(fpr, tpr):
    """
    plots a Roc curve given the false positive rate(fpr)
    and true positive raate (tpr) of a model
    """
    #plot roc curve
    plt.plot(fpr, tpr, color="orange", label="ROC")
    #plot line with no prediction power (baseline)
    plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--", label="Guessing")

    #customise the plot
    plt.xlabel("False positive rate(fpr)")
    plt.ylabel("True positive rate(tpr)")
    plt.title("Recive Operating Characterstic (ROC curve)")
    plt.legend()
    plt.show()

plot_roc_curve(fpr, tpr)
```



In [107]:

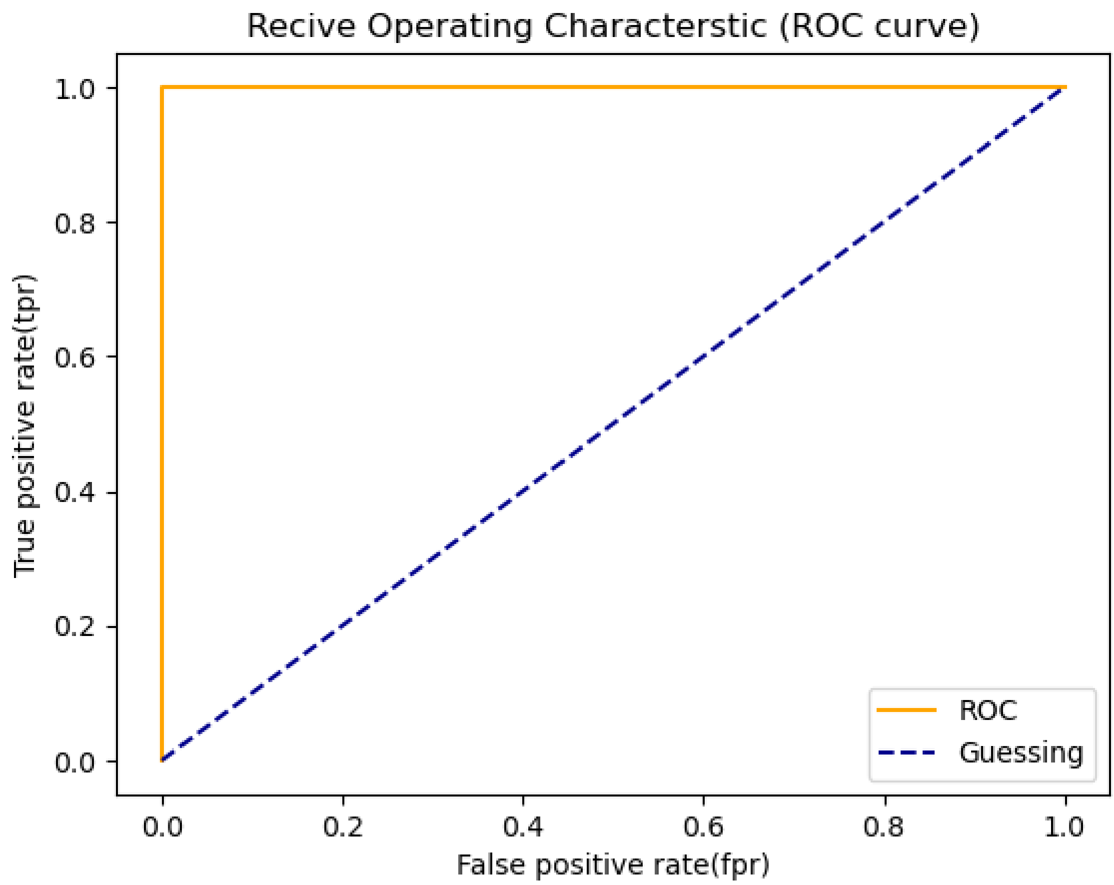
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_probs_positive)
```

Out[107]:

```
0.8669181034482759
```

In [108]:

```
#plot perfect roc curve and auc curve
fpr, tpr, thresholds = roc_curve(y_test, y_test)
plot_roc_curve(fpr, tpr)
```



In [109]:

```
#perfect auc score
roc_auc_score(y_test, y_test)
```

Out[109]:

```
1.0
```

Confusion matrix

a confusion matrix is quick way to compare the labels a model and that the actual labels it was supposed to pridict in essance giving an idea of where the model is gatiing confused

In [110]:

```
from sklearn.metrics import confusion_matrix

y_preds = clf.predict(x_test)
confusion_matrix(y_test, y_preds)
```

Out[110]:

```
array([[23,  6],
       [ 6, 26]], dtype=int64)
```

In [111]:

```
#visualize confusion matrix with pd.crosstab()
pd.crosstab(y_test,
            y_preds,
            rownames=["Actual Labels"],
            colnames=["Predicted Labels"])
```

Out[111]:

Predicted Labels	0	1
Actual Labels		
0	23	6
1	6	26

In [112]:

```
23+6+6+26
```

Out[112]:

61

In [113]:

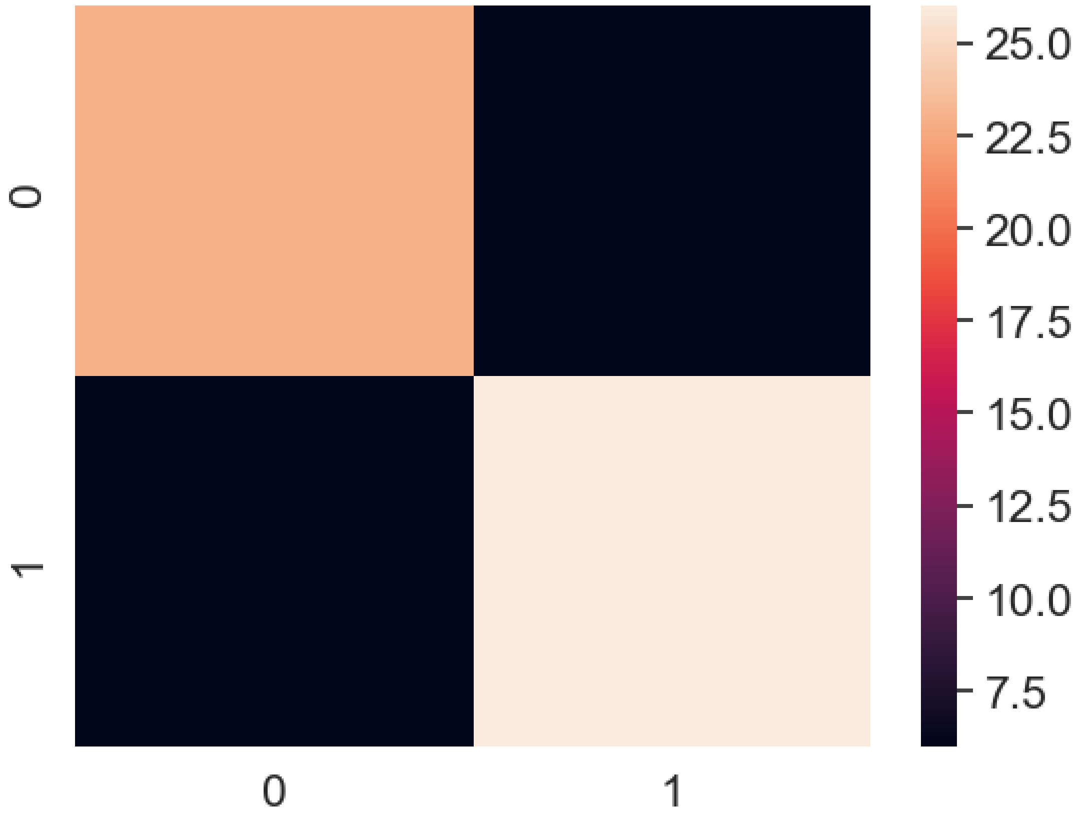
```
len(x_test)
```

Out[113]:

61

In [114]:

```
#make our confussion matrix mode visual with seaborn's heatmap()
import seaborn as sns

#set the font scale
sns.set(font_scale=1.5)

#create a confussion matrix
conf_mat= confusion_matrix(y_test, y_preds)
#plot it using seaborn
sns.heatmap(conf_mat);
```



Creating a confussion matrix using Scikit learn

In [115]:

```
clf
```

Out[115]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [116]:

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(estimator=clf, x=x, y=y)
```

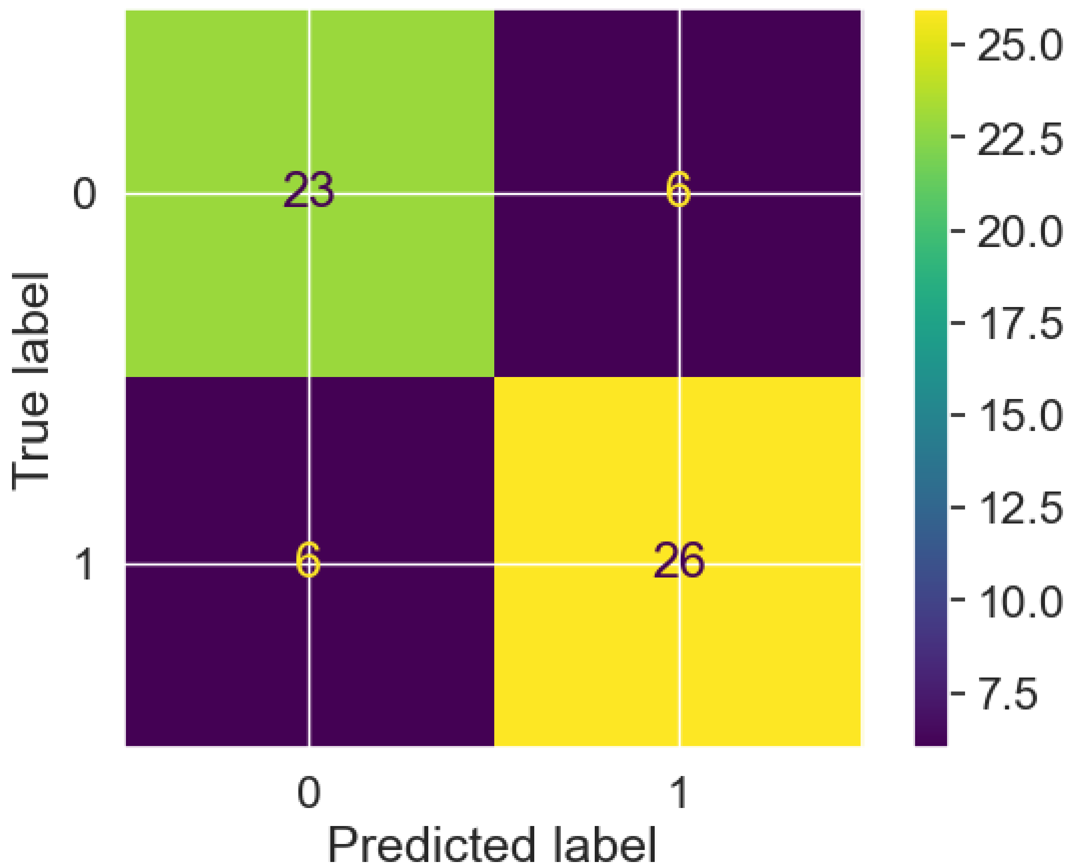
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
Cell In[116], line 2
      1 from sklearn.metrics import ConfusionMatrixDisplay
----> 2 ConfusionMatrixDisplay.from_estimator(estimator=clf, x=x, y=y)

TypeError: ConfusionMatrixDisplay.from_estimator() got an unexpected keywo
rd argument 'x'
```

In [118]:

```
ConfusionMatrixDisplay.from_predictions(y_true=y_test,
                                        y_pred=y_preds)
```

Out[118]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x290993
d3190>
```



Classification report

In [119]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.79      0.79      0.79        29
           1       0.81      0.81      0.81        32

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

In [120]:

```
#where precision and recall become valuable
disease_true=np.zeros(10000)
disease_true[0]=1 #only one positive case

disease_preds =np.zeros(10000) #model predicts every case zero

pd.DataFrame(classification_report(disease_true,
                                   disease_preds,
                                   output_dict=True))
```

```
C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn\met
rics\_classification.py:1334: UndefinedMetricWarning: Precision and F-scor
e are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn\met
rics\_classification.py:1334: UndefinedMetricWarning: Precision and F-scor
e are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn\met
rics\_classification.py:1334: UndefinedMetricWarning: Precision and F-scor
e are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[120]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.99990	0.0	0.9999	0.499950	0.99980
recall	1.00000	0.0	0.9999	0.500000	0.99990
f1-score	0.99995	0.0	0.9999	0.499975	0.99985
support	9999.00000	1.0	0.9999	10000.000000	10000.00000

4.2.2 Regression model evaluation metrics

The ones we're going to cover are:

1. R^2 (pronounced r-squared) or coefficient of determination
2. Mean absolute error (MAE)
3. Mean squared error (MSE)

In [121]:

```
from sklearn.ensemble import RandomForestRegressor
np.random.seed(42)
#make the data
x = housing_df.drop("target", axis=1)
y =  housing_df["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantitate random forest classifier
model = RandomForestRegressor(n_estimators=100)
#fit the model to the data(training machine laerrning modcel)
model.fit(x_train, y_train)
```

Out[121]:

```
RandomForestRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [122]:

```
model.score(x_test, y_test)
```

Out[122]:

```
0.8065734772187598
```

In [123]:

```
housing_df.head()
```

Out[123]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [124]:

```
y_test.mean()
```

Out[124]:

```
2.0550030959302323
```

In [125]:

```
from sklearn.metrics import  r2_score

#fill an array with y_test mean
y_test_mean =np.full(len(y_test), y_test.mean())
```

In [126]:

```
y_test_mean[:10]
```

Out[126]:

```
array([2.0550031, 2.0550031, 2.0550031, 2.0550031, 2.0550031, 2.0550031,
       2.0550031, 2.0550031, 2.0550031, 2.0550031])
```

In [127]:

```
r2_score(y_true=y_test,
         y_pred=y_test_mean)
```

Out[127]:

```
0.0
```

In [128]:

```
r2_score(y_true=y_test,
         y_pred=y_test)
```

Out[128]:

```
1.0
```

Mean absolute error (MAE)

MAE is the average of the absolute differences between predictions and actual values.

It gives you an idea of how wrong your models predictions are.

In [129]:

```
#MAE
from sklearn.metrics import mean_absolute_error
y_preds = model.predict(x_test)
mae= mean_absolute_error(y_test, y_preds)
mae
```

Out[129]:

```
0.32659871732073664
```

In [130]:

```
df = pd.DataFrame(data={"actual values" : y_test,
                        "predicted values" : y_preds})
df["diffrences"] =df["predicted values"] - df["actual values"]
df.head(10)
```

Out[130]:

	actual values	predicted values	diffrences
20046	0.47700	0.493840	0.016840
3024	0.45800	0.754940	0.296940
15663	5.00001	4.928596	-0.071414
20484	2.18600	2.543160	0.357160
9814	2.78000	2.331760	-0.448240
13311	1.58700	1.652530	0.065530
7113	1.98200	2.343230	0.361230
7668	1.57500	1.661820	0.086820
18246	3.40000	2.474890	-0.925110
5723	4.46600	4.834478	0.368478

In [131]:

```
#Mae using formulas and diffrences
np.abs(df["diffrences"]).mean()
```

Out[131]:

```
0.32659871732073664
```

Mean squared error (MSE)

MSE is the mean of the square of the errors between actual and predicted values.

In [132]:

```
#mean square error
from sklearn.metrics import mean_squared_error

y_preds = model.predict(x_test)
mse = mean_squared_error(y_test, y_preds)
mse
```

Out[132]:

```
0.2534678520824551
```

In [133]:

```
df["squared_diffrences"] = np.square(df["diffrences"])
df.head()
```

Out[133]:

	actual values	predicted values	diffrences	squared_diffrences
20046	0.47700	0.493840	0.016840	0.000284
3024	0.45800	0.754940	0.296940	0.088173
15663	5.00001	4.928596	-0.071414	0.005100
20484	2.18600	2.543160	0.357160	0.127563
9814	2.78000	2.331760	-0.448240	0.200919

In [134]:

```
#calculate a mse by hand
squared =np.square(df["diffrences"])
squared.mean()
```

Out[134]:

```
0.2534678520824551
```

In [135]:

```
df_large_error =df.copy()
df_large_error.iloc[0]["squared_diffrences"] =16
```

In [136]:

```
df_large_error.head()
```

Out[136]:

	actual values	predicted values	diffrences	squared_diffrences
20046	0.47700	0.493840	0.016840	16.000000
3024	0.45800	0.754940	0.296940	0.088173
15663	5.00001	4.928596	-0.071414	0.005100
20484	2.18600	2.543160	0.357160	0.127563
9814	2.78000	2.331760	-0.448240	0.200919

In [137]:

```
#calculate MSE with large error
df_large_error["squared_diffrences"].mean()
```

Out[137]:

```
0.2573437523766412
```

In [138]:

```
df_large_error.iloc[1:100] = 20
df_large_error
```

Out[138]:

	actual values	predicted values	diffrences	squared_diffrences
20046	0.47700	0.493840	0.016840	16.000000
3024	20.00000	20.000000	20.000000	20.000000
15663	20.00000	20.000000	20.000000	20.000000
20484	20.00000	20.000000	20.000000	20.000000
9814	20.00000	20.000000	20.000000	20.000000
...	...	...	...	...
15362	2.63300	2.220380	-0.412620	0.170255
16623	2.66800	1.947760	-0.720240	0.518746
18086	5.00001	4.836378	-0.163632	0.026775
2144	0.72300	0.717820	-0.005180	0.000027
3665	1.51500	1.679010	0.164010	0.026899

4128 rows × 4 columns

4.2.3 Finally using the scoring parameter

In [139]:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)
x= heart_disease.drop("target", axis=1)
y =heart_disease["target"]

clf =RandomForestClassifier(n_estimators=100)
```

In [140]:

```
np.random.seed(42)

#cross validation accurecy
cv_acc = cross_val_score(clf, x, y, cv=5, scoring=None)
cv_acc
```

Out[140]:

```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

In [141]:

```
#cross validation accuracy
print(f"The cross validation accuracy is: {np.mean(cv_acc)*100:2f}%")
```

```
The cross validation accuracy is: 82.480874%
```

In [142]:

```
np.random.seed(42)

#cross validation accurecy
cv_acc = cross_val_score(clf, x, y, cv=5, scoring="accuracy")
```

In [143]:

```
print(f"The cross validation accuracy is: {np.mean(cv_acc)*100:2f}%")
```

```
The cross validation accuracy is: 82.480874%
```

In [144]:

```
#precision
np.random.seed(42)
cv_precision = cross_val_score(clf, x, y, cv=5, scoring="precision")
cv_precision
```

Out[144]:

```
array([0.82352941, 0.93548387, 0.84848485, 0.79411765, 0.76315789])
```

In [145]:

```
# cross validatrion precision
print(f"The cross validation precision is: {np.mean(cv_precision)}")
```

```
The cross validation precision is: 0.8329547346025924
```

In [146]:

```
#recall
np.random.seed(42)
cv_recall = cross_val_score(clf, x, y, cv=5, scoring="recall")
cv_recall
```

Out[146]:

```
array([0.84848485, 0.87878788, 0.84848485, 0.81818182, 0.87878788])
```

In [147]:

```
print(f"The cross validation recall is: {np.mean(cv_recall)}")
```

```
The cross validation recall is: 0.8545454545454545
```

let's see the scorong parameter being using for a regression problem

In [148]:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

X = housing_df.drop("target", axis=1)

y = housing_df [ "target"]

model = RandomForestRegressor (n_estimators=100)
```

In [149]:

```
np.random.seed(42)

cv_r2 = cross_val_score(model, x, y, cv=3, scoring=None)

np.mean(cv_r2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[149], line 3
      1 np.random.seed(42)
----> 3 cv_r2 = cross_val_score(model, x, y, cv=3, scoring=None)
      5 np.mean(cv_r2)

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\model_selection\_validation.py:515, in cross_val_score(estimator, X, y, groups, scoring, cv, n_jobs, verbose, fit_params, pre_dispatch, error_score)
    512 # To ensure multimetric format is not supported
    513 scorer = check_scoring(estimator, scoring=scoring)
--> 515 cv_results = cross_validate(
    516     estimator=estimator,
    517     X=X,
    518     y=y,
    519     groups=groups,
    520     scoring={"score": scorer},
    521     cv=cv,
    522     n_jobs=n_jobs,
    523     verbose=verbose,
    524     fit_params=fit_params,
    525     pre_dispatch=pre_dispatch,
    526     error_score=error_score,
    527 )
    528 return cv_results["test_score"]

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\model_selection\_validation.py:252, in cross_validate(estimator, X, y, groups, scoring, cv, n_jobs, verbose, fit_params, pre_dispatch, return_train_score, return_estimator, error_score)
     49 def cross_validate(
     50     estimator,
     51     X,
   (...)
     63     error_score=np.nan,
     64 ):
     65     """Evaluate metric(s) by cross-validation and also record fit/score times.
     66 
     67     Read more in the :ref:`User Guide <multimetric_cross_validation>`.
   (...)
    250     [0.28009951 0.3908844  0.22784907]
    251     """
--> 252     X, y, groups = indexable(X, y, groups)
    254     cv = check_cv(cv, y, classifier=is_classifier(estimator))
    256     if callable(scoring):

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\utils\validation.py:433, in indexable(*iterables)
    414 """Make arrays indexable for cross-validation.
    415 
    416 Checks consistent length, passes through None, and ensures that everything
   (...)
    429     sparse matrix, or dataframe) or `None`.
    430 """
    432 result = [_make_indexable(X) for X in iterables]
```

```
--> 433 check_consistent_length(*result)
    434 return result

File ~\Machine_learning\project1\env\lib\site-packages\sklearn\utils\valid
ation.py:387, in check_consistent_length(*arrays)
    385 uniques = np.unique(lengths)
    386 if len(uniques) > 1:
--> 387     raise ValueError(
    388         "Found input variables with inconsistent numbers of sample
s: %r"
    389         % [int(l) for l in lengths]
    390     )

ValueError: Found input variables with inconsistent numbers of samples: [3
03, 20640]
```

4.3 Using diffrent evaluation metrics as Schite learn funtions

the 3rd way to evaluate scikit learn models/estimators

In [151]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
np.random.seed(42)
#create x and y
x = heart_disease.drop("target", axis=1)
y = heart_disease["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#create the model
clf = RandomForestClassifier()
#fit the model

clf.fit(x_train, y_train)
# make prediction
y_preds =clf.predict(x_test)
#evaluate the model using evaluation funtion
print("Classifier metrics on test set")
print(f"Accuracy: {accuracy_score(y_test, y_preds)*100:.2f}%")
print (f"Precision: {precision_score (y_test, y_preds)}")
print (f"Recall: {recall_score (y_test, y_preds)}")
print (f"F1: {f1_score(y_test, y_preds)}")
```

```
Classifier metrics on test set
Accuracy: 85.25%
Precision: 0.8484848484848485
Recall: 0.875
F1: 0.8615384615384615
```

In [152]:

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
np.random.seed(42)

#Create X & y

x = housing_df.drop("target", axis=1)
y =  housing_df["target"]
#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Create model

model = RandomForestRegressor()

#Fit model

model.fit(x_train, y_train)

#Make predictions
y_preds = model.predict(x_test)

#Evaluate model using evaluation functions

print("Regression metrics on the test set")

print(f"R2 score: {r2_score (y_test, y_preds)}")

print(f"MAE:{mean_absolute_error (y_test, y_preds)}")

print(f"MSE: {mean_squared_error(y_test, y_preds)}")
```

```
Regression metrics on the test set
R2 score: 0.8065734772187598
MAE:0.32659871732073664
MSE: 0.2534678520824551
```

5. Improving a model

First predictions = baseline predictions. First model = baseline model.

From a data perspective:

Could we collect more data? (generally, the more data, the better)

Could we improve our data?

From a model perspective:

Is there a better model we could use?

- Could we improve the current model?

Hyperparameters vs. Parameters |

- Parameters = model find these patterns in data

Hyperparameters = settings on a model you can adject to improve its ability to find

three ways to adjust hyperparameter

1. by hand
2. Randomly with RandomSearchCV
3. Exhaustively with gridsearchcv

In [153]:

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

In [154]:

```
clf.get_params()
```

Out[154]:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 '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}
```

5.1 Tuning hyperparameters by test

Let's make 3 sets training validation and test

In [155]:

```
clf.get_params()
```

Out[155]:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 '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}
```

we're going to try and adjust

- 'max_depth'
- 'max_features'
- 'min_sample_leaf'
- 'min_sample_split'
- n_estimators

In [156]:

```
def evaluate_preds(y_true, y_preds):
    """
    performs evaluation comparision on y_true labels vs. y_preds label
    on classifification
    """
    accuracy = accuracy_score(y_true, y_preds)
    precision = precision_score(y_true, y_preds)
    recall = recall_score(y_true, y_preds)
    f1 = f1_score(y_true, y_preds)
    metrics_dict = {"accuracy": round(accuracy, 2),
                    "precision": round(precision, 2),
                    "recall": round(recall, 2),
                    "f1": round(f1, 2)}
    print(f"Acc: {accuracy * 100:.2f}%")
    print(f"precision: {precision:}.2f")
    print(f"Recall: {recall:.2f}")
    print(f"F1 score:{f1:.2f}")

    return metrics_dict
```

In [157]:

```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)
#suffle  the data
heart_disease_shuffled = heart_disease.sample(frac=1)

#split the data
x= heart_disease_shuffled.drop("target", axis=1)
y= heart_disease_shuffled["target"]

#split the data into train validaiton and test sets
train_split = round(0.7 * len(heart_disease_shuffled)) # 70% of data
valid_split =round(train_split + 0.15 *len(heart_disease_shuffled)) #15% of 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]

len(x_train), len(x_valid), len(x_test)
clf = RandomForestClassifier()
clf.fit(x_train, y_train)
#make baseline predication
y_preds = clf.predict(x_valid)
#Evaluationthe classifier on validation set
baseline_metrics = evaluate_preds(y_valid, y_preds)
baseline_metrics
```

```
C:\Users\alokr\AppData\Local\Temp\ipykernel_16480\1156575478.py:14: Future
Warning: The behavior of `series[i:j]` with an integer-dtype index is depr
ecated. In a future version, this will be treated as *label-based* indexin
g, consistent with e.g. `series[i]` lookups. To retain the old behavior, u
se `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  x_train, y_train =x[:train_split], y[:train_split]
C:\Users\alokr\AppData\Local\Temp\ipykernel_16480\1156575478.py:15: Future
Warning: The behavior of `series[i:j]` with an integer-dtype index is depr
ecated. In a future version, this will be treated as *label-based* indexin
g, consistent with e.g. `series[i]` lookups. To retain the old behavior, u
se `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  x_valid, y_valid = x[train_split:valid_split], y[train_split:valid_spli
t]
C:\Users\alokr\AppData\Local\Temp\ipykernel_16480\1156575478.py:16: Future
Warning: The behavior of `series[i:j]` with an integer-dtype index is depr
ecated. In a future version, this will be treated as *label-based* indexin
g, consistent with e.g. `series[i]` lookups. To retain the old behavior, u
se `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  x_test, y_test =x[valid_split:], y[:valid_split]

Acc: 82.22%
precision: 0.8148148148148148.2f
Recall: 0.88
F1 score:0.85
```

Out[157]:

```
{'accuracy': 0.82, 'precision': 0.81, 'recall': 0.88, 'f1': 0.85}
```

In [158]:

```
np.random.seed(42)
#create a second classifier with diffrent hyperparameters
clf_2 = RandomForestClassifier(n_estimators=100)
clf_2.fit(x_train, y_train)

#make prediction with diffrent hyperparameters
y_preds_2 =clf_2.predict(x_valid)

#evaluate the 2nd classifier
clf_2_metrics = evaluate_preds(y_valid, y_preds_2)
```

```
Acc: 82.22%
precision: 0.84.2f
Recall: 0.84
F1 score:0.84
```

5.2 Hyperparameter tuning with RandomizedSearchCV

In [159]:

```
from sklearn.model_selection import RandomizedSearchCV
grid = {"n_estimators":[10, 100, 200, 500, 1000, 1200],
        "max_depth":[None, 5, 10, 20, 30],
        "max_features":["auto", "sqrt"],
        "min_samples_split": [2, 4, 6],
        "min_samples_leaf": [1, 2, 4]}
np.random.seed(42)
#split the data
x= heart_disease_shuffled.drop("target", axis=1)
y= heart_disease_shuffled["target"]

#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantiate RandomforestClassifier
clf = RandomForestClassifier(n_jobs=1)

# setup RandomizedsearchCV
rs_clf = RandomizedSearchCV(estimator=clf,
                            param_distributions=grid,
                            n_iter=10, #number of model to try
                            cv=5,
                            verbose=2)

#fit the RandomizedSearchCV version of clf
rs_clf.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=1200; total time=   3.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=1200; total time=   3.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=1200; total time=   3.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=1200; total time=   3.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=1200; total time=   3.4s

C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn
\ensemble\_forest.py:427: FutureWarning: `max_features='auto'` has been
deprecated in 1.1 and will be removed in 1.3. To keep the past behaviou
r, explicitly set `max_features='sqrt'` or remove this parameter as it
is also the default value for RandomForestClassifiers and ExtraTreesCla
ssifiers.
  warn(
```

In [160]:

```
#show best params avove result
rs_clf.best_params_
```

Out[160]:

```
{'n_estimators': 200,
 'min_samples_split': 6,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': None}
```

In [161]:

```
#Make predictions with the best hyperparameters
rs_y_preds = rs_clf.predict(x_test)

#Evaluate the predictions
rs_metrics = evaluate_preds (y_test, rs_y_preds)
```

```
Acc: 81.97%
precision: 0.7741935483870968.2f
Recall: 0.86
F1 score:0.81
```

5.3 Hyperparameter tuning with GridSearchCV

In [162]:

```
grid
```

Out[162]:

```
{'n_estimators': [10, 100, 200, 500, 1000, 1200],
 'max_depth': [None, 5, 10, 20, 30],
 'max_features': ['auto', 'sqrt'],
 'min_samples_split': [2, 4, 6],
 'min_samples_leaf': [1, 2, 4]}
```

In [163]:

```
grid_2 = {'n_estimators': [100, 200, 500],
          'max_depth': [None],
          'max_features': ['auto', 'sqrt'],
          'min_samples_split': [6],
          'min_samples_leaf': [1, 2]}
```

In [164]:

```
from sklearn.model_selection import GridSearchCV, train_test_split
np.random.seed(42)
#split the data
x= heart_disease_shuffled.drop("target", axis=1)
y= heart_disease_shuffled["target"]

#split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
#Instantiate RandomforestClassifier
#clf = RandomForestClassifier(n_jobs=1)

# setup GridsearchCV
## gs_clf = GridSearchCV(estimator=clf,
##                       param_grid=grid_2,
##                       cv=5,
##                       verbose=2)
#n_iter is not here becouse it is brute forse it check every combination of grid
#fit the gridSearchCV version of clf
gs_clf.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn
\ensemble\_forest.py:427: FutureWarning: `max_features='auto'` has been
deprecated in 1.1 and will be removed in 1.3. To keep the past behaviou
r, explicitly set `max_features='sqrt'` or remove this parameter as it
is also the default value for RandomForestClassifiers and ExtraTreesCla
ssifiers.
  warn(

[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sam
ples_split=6, n_estimators=100; total time=   0.2s

C:\Users\alokr\Machine_learning\project1\env\lib\site-packages\sklearn
\ensemble\_forest.py:427: FutureWarning: `max_features='auto'` has been
deprecated in 1.1 and will be removed in 1.3. To keep the past behaviou
r, explicitly set `max_features='sqrt'` or remove this parameter as it
is also the default value for RandomForestClassifiers and ExtraTreesCla
ssifiers.
  warn(
```

In [165]:

```
gs_clf.best_params_
```

Out[165]:

```
{'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'n_estimators': 200}
```

In [166]:

```
gs_y_preds = gs_clf.predict(x_test)
#evaluate the prediction
gs_metrics = evaluate_preds(y_test, gs_y_preds)
```

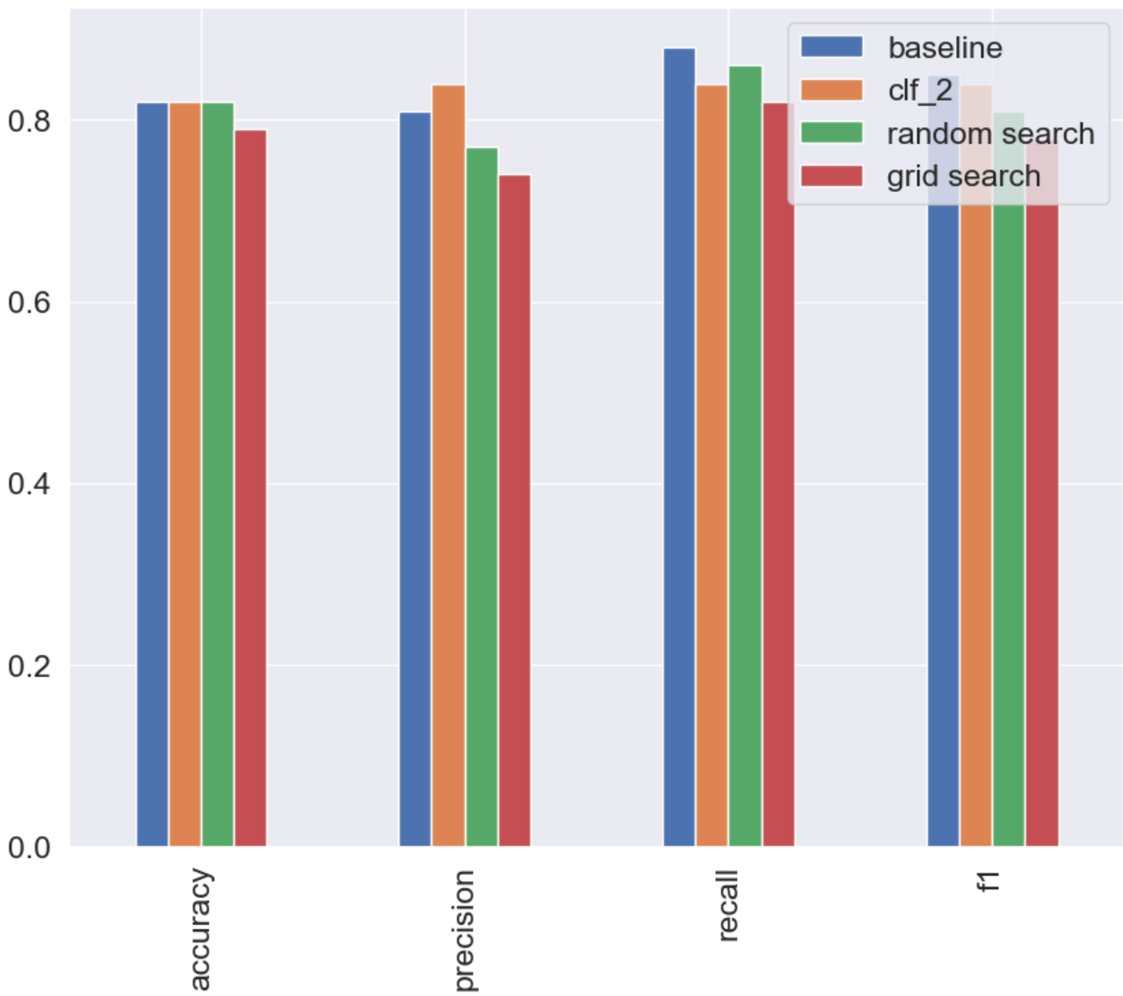
```
Acc: 78.69%
precision: 0.7419354838709677.2f
Recall: 0.82
F1 score:0.78
```

In [167]:

```
#let's compare our diffrent model metrics
compare_metrics = pd.DataFrame({"baseline": baseline_metrics,
                                "clf_2": clf_2_metrics,
                                "random search": rs_metrics,
                                "grid search": gs_metrics})
compare_metrics.plot.bar(figsize=(10, 8))
```

Out[167]:

```
<AxesSubplot: >
```



6.0 Saaving and loading trained machine learning models

Two ways to save and load machine learning models:

1. with python's pickle models

2. with the jobling models

In [169]:

```
import pickle
# save an extisting model of file
pickle.dump(gs_clf, open("gs_random_forest_model_1.pkl", "wb"))
```

In [171]:

```
#load save model
loaded_pickle_model = pickle.load(open("gs_random_forest_model_1.pkl", "rb"))
```

In [173]:

```
#make some predictions
pickle_y_preds = loaded_pickle_model.predict(x_test)
evaluate_preds(y_test, pickle_y_preds)
```

```
Acc: 78.69%
precision: 0.7419354838709677.2f
Recall: 0.82
F1 score:0.78
```

Out[173]:

```
{'accuracy': 0.79, 'precision': 0.74, 'recall': 0.82, 'f1': 0.78}
```

In [177]:

```
#save using joblib
from joblib import dump, load

#save model to file
dump(gs_clf, filename="gs_random_forest_model_1.joblib")
```

Out[177]:

```
['gs_random_forest_model_1.joblib']
```

In [179]:

```
#import a saved joblib model
loaded_joblib_model = load(filename="gs_random_forest_model_1.joblib")
```

In [180]:

```
# make and evaluate some prediction
joblib_y_preds =loaded_joblib_model.predict(x_test)
evaluate_preds(y_test,joblib_y_preds)
```

```
Acc: 78.69%
precision: 0.7419354838709677.2f
Recall: 0.82
F1 score:0.78
```

Out[180]:

```
{'accuracy': 0.79, 'precision': 0.74, 'recall': 0.82, 'f1': 0.78}
```

In [ ]:

```

```