<-------- Complete Workflow ------->

Introduction to Scikit Learn

what we're going to cover.

0. An end to end. Scikit learn workflow.
1. getting the data ready
2. choose the right estimator / algorithm for our problems
3. fit the model / algorithm and use it to make predictions on our data.
4. evaluating a model
5. improve a model
6. save and load a trained model
7. putting it all together.

0. An end to end. Scikit learn workflow

1. getting the data ready

In [1]:
```
import numpy as np
```

In [2]:
```
# 1. getting the data ready
import pandas as pd
heart_disease = pd.read_csv("11.2 heart-disease.csv")
heart_disease
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [3]:
```
# Create X (features matrix)
x = heart_disease.drop("target",axis=1)

# Create y (labels)
y = heart_disease["target"]
```

2. choose the right estimator / algorithm for our problems & hyperparameters

In [5]:
```
# 2. Choose the right model and hyperparameters
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators= 100)

# We'll keep the default hyperparameters
clf.get_params()
```

Out[5]:
```
{'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}
```

3. fit the model to the data to train data

In [4]:
```
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 [5]:
```
clf.fit(x_train, y_train);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-afd5f004f1a8> in <module>
----> 1 clf.fit(x_train, y_train);
```

```
NameError: name 'clf' is not defined
```

In [ ]:
```
x_train
```

In [27]:
```
# make a predictions
# our model can't make pridiction on data which are not of same shape
y_label = clf.predict(np.array([0,2,3,4]))
# so it throw error
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-4f28c678245c> in <module>
      1 # make a predictions
      2 # our model can't make pridiction on data which are not of same shape
----> 3 y_label = clf.predict(np.array([0,2,3,4]))

~\desktop\sample_project\env\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    628             The predicted classes.
    629         """
--> 630         proba = self.predict_proba(X)
    631
    632         if self.n_outputs_ == 1:

~\desktop\sample_project\env\lib\site-packages\sklearn\ensemble\_forest.py in predict_proba(self, X)
    672         check_is_fitted(self)
    673         # Check data
--> 674         X = self._validate_X_predict(X)
    675
    676         # Assign chunk of trees to jobs

~\desktop\sample_project\env\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self, X)
    420         check_is_fitted(self)
    421
--> 422         return self.estimators_[0]._validate_X_predict(X, check_input=True)
    423
    424     @property

~\desktop\sample_project\env\lib\site-packages\sklearn\tree\_classes.py in _validate_X_predict(self, X, check_input)
    405         """Validate the training data on predict (probabilities)."""
    406         if check_input:
--> 407             X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr",
    408                                     reset=False)
    409             if issparse(X) and (X.indices.dtype != np.intc or

~\desktop\sample_project\env\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **c
heck_params)
    419             out = X
    420         elif isinstance(y, str) and y == 'no_validation':
--> 421             X = check_array(X, **check_params)
    422             out = X
    423         else:

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py 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)
    692             # If input is 1D raise error
    693             if array.ndim == 1:
--> 694                 raise ValueError(
    695                     "Expected 2D array, got 1D array instead:\narray={}.\n"
    696                     "Reshape your data either using array.reshape(-1, 1) if "

ValueError: Expected 2D array, got 1D array instead:
array=[0. 2. 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 [8]:
```
y_preds = clf.predict(x_test)
y_preds
```

Out[8]:
```
array([0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0], dtype=int64)
```

In [9]:
```
y_test
```

Out[9]:
```
295    0
300    0
58     1
210    0
95     1
      ..
139    1
262    0
118    1
136    1
229    0
Name: target, Length: 61, dtype: int64
```

In [10]:
```
#4. Evaluate the model on the training data and test data
clf.score(x_train, y_train)
```

Out[10]:
```
1.0
```

In [11]:
```
clf.score(x_test, y_test)
```

Out[11]:
```
0.819672131147541
```

In [12]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

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

```
              precision    recall  f1-score   support

           0       0.83      0.74      0.78        27
           1       0.81      0.88      0.85        34

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

In [13]:
```
confusion_matrix(y_test, y_preds )
```

Out[13]:
```
array([[20,  7],
       [ 4, 30]], dtype=int64)
```

In [14]:
```
accuracy_score(y_test, y_preds)
```

Out[14]:
```
0.819672131147541
```

In [15]:
```
# 5. Improve a model
# Try different amount of n_estimators
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:75.409836%

Trying model with 20 estimators...
Model accuracy on test set:83.606557%

Trying model with 30 estimators...
Model accuracy on test set:80.327869%

Trying model with 40 estimators...
Model accuracy on test set:81.967213%

Trying model with 50 estimators...
Model accuracy on test set:80.327869%

Trying model with 60 estimators...
Model accuracy on test set:78.688525%

Trying model with 70 estimators...
Model accuracy on test set:83.606557%

Trying model with 80 estimators...
Model accuracy on test set:78.688525%

Trying model with 90 estimators...
Model accuracy on test set:80.327869%
```

In [16]:
```
# 6. save a model and load it
import pickle

pickle.dump(clf, open("Random_forst_model_1.pkl","wb"))
```

In [17]:
```
loaded_model = pickle.load(open("Random_forst_model_1.pkl","rb"))
loaded_model.score(x_test, y_test)
```

Out[17]:
```
0.8032786885245902
```

<----- Complete Workflow Ends ----->

Now we breakdown each component and learn them one by one

In [4]:
```
# Standerd imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

1. Getting the data ready to be used with machine learing

getting the data ready to be used with machine learing

Three main things we have to do:

1. Split the data into features and labels (usually `x` & `y`)
2. Filling (also called imputing) or disregarding missing values
3. Converting non-numerical values to numerical values (also called feature encoding)

Well we need a data set to begin with first and we our using heart disease data set. So in this case we want to use the feature columns to predict Y.

1. X = feature = all the columns like (age, sex, cp, fbs, thal etc), which is use to predict Y.
2. Y = labels = which is going to be are target columns.

In [5]:
```
# importting data
heart_disease = pd.read_csv("11.2 heart-disease.csv")
```

In [6]:
```
# All the columns expect the 'target'
x = heart_disease.drop("target", axis = 1)
x.head()
```

Out[6]:

	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 [21]:
```
y = heart_disease["target"]
y.head(5)
```

Out[21]:
```
0    1
1    1
2    1
```

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

- Split data into training and testing data sets
- In machine learning one of the most fundamental principles is never evaluate or test your modelson data that it is learned from.
- Scikit Learn has a convenient function which allowing us to do that.

In [22]:
```
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 [23]:
```
x_train.shape , x_test.shape , y_train.shape , y_test.shape
```

Out[23]:
```
((242, 13), (61, 13), (242,), (61,))
```

In [24]:
```
len(heart_disease)
```

Out[24]:
```
303
```

1.1. Converting non-numerical values to numerical values

- Make sure its all numarical

In [25]:
```
car_sales = pd.read_csv("car-sales-extended.csv")
car_sales.head()
```

Out[25]:

	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 [26]:
```
len(car_sales)
```

Out[26]:
```
1000
```

In [27]:
```
car_sales.dtypes
```

Out[27]:
```
Make             object
Colour           object
Odometer (KM)     int64
Doors             int64
Price             int64
dtype: object
```

So first of all what we're going to do, is split the data into x and y so we'll use these four columns make, color, odometer, doors. By using these four we try and predict the price of a car.

In [28]:
```
# Split into x & y
x = car_sales.drop('Price',axis = 1)
y = car_sales["Price"]
```

In [29]:
```
# split into training and testing
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 [30]:
```
# build a machine learning model
from sklearn.ensemble import RandomForestRegressor

# model / clf
model = RandomForestRegressor()
model.fit(x_train, y_train)
model.score( x_test, y_test)

# this will give us error because two of are columns data is object and machine learning models work on numaric data,
# So we have to convert are data to numaric values
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-30-d23ea99edb2c> in <module>
      4 # model / clf
      5 model = RandomForestRegressor()
----> 6 model.fit(x_train, y_train)
      7 model.score( x_test, y_test)
      8

~\desktop\sample_project\env\lib\site-packages\sklearn\ensemble\_forest.py in fit(self, X, y, sample_weight)
    302                 "sparse multilabel-indicator for y is not supported."
    303             )
--> 304         X, y = self._validate_data(X, y, multi_output=True,
    305                                    accept_sparse="csc", dtype=DTYPE)
    306         if sample_weight is not None:

~\desktop\sample_project\env\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **c
heck_params)
    431                 y = check_array(y, **check_y_params)
    432             else:
--> 433                 X, y = check_X_y(X, y, **check_params)
    434             out = X, y
    435

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_sparse, accept_large_sp
arse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_nu
meric, estimator)
    869         raise ValueError("y cannot be None")
    870
--> 871     X = check_array(X, accept_sparse=accept_sparse,
    872                     accept_large_sparse=accept_large_sparse,
    873                     dtype=dtype, order=order, copy=copy,
```

```
~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py 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)
    671                     array = array.astype(dtype, casting="unsafe", copy=False)
    672                 else:
--> 673                     array = np.asarray(array, order=order, dtype=dtype)
    674             except ComplexWarning as complex_warning:
    675                 raise ValueError("Complex data not supported\n"

~\desktop\sample_project\env\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order, like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104


~\desktop\sample_project\env\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   1988
   1989     def __array__(self, dtype: NpDtype | None = None) -> np.ndarray:
-> 1990         return np.asarray(self._values, dtype=dtype)
   1991
   1992     def __array_wrap__(

~\desktop\sample_project\env\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order, like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104


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

In [45]:
```
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make","Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder = "passthrough")

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

Out[45]:
```
<1000x16 sparse matrix of type '<class 'numpy.float64'>'
        with 4000 stored elements in Compressed Sparse Row format>
```

In [55]:
```
pd.DataFrame(transformed_x)
```

Out[55]:

	0
0	(0, 1)\t1.0\n (0, 9)\t1.0\n (0, 12)\t1.0\n...
1	(0, 0)\t1.0\n (0, 6)\t1.0\n (0, 13)\t1.0\n...
2	(0, 1)\t1.0\n (0, 9)\t1.0\n (0, 12)\t1.0\n...
3	(0, 3)\t1.0\n (0, 9)\t1.0\n (0, 12)\t1.0\n...
4	(0, 2)\t1.0\n (0, 6)\t1.0\n (0, 11)\t1.0\n...
...	...
995	(0, 3)\t1.0\n (0, 5)\t1.0\n (0, 12)\t1.0\n...
996	(0, 4)\t1.0\n (0, 9)\t1.0\n (0, 11)\t1.0\n...
997	(0, 2)\t1.0\n (0, 6)\t1.0\n (0, 12)\t1.0\n...
998	(0, 1)\t1.0\n (0, 9)\t1.0\n (0, 12)\t1.0\n...
999	(0, 3)\t1.0\n (0, 6)\t1.0\n (0, 12)\t1.0\n...

1000 rows × 1 columns

In [54]:
```
dummies = pd.get_dummies(car_sales[["Make","Colour","Doors"]])
dummies
```

Out[54]:

	Doors	Make_BMW	Make_Honda	Make_Nissan	Make_Toyota	Colour_Black	Colour_Blue	Colour_Green	Colour_Red	Colour_White
0	4	0	1	0	0	0	0	0	0	1
1	5	1	0	0	0	0	1	0	0	0
2	4	0	1	0	0	0	0	0	0	1
3	4	0	0	0	1	0	0	0	0	1
4	3	0	0	1	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...
995	4	0	0	0	1	1	0	0	0	0
996	3	0	0	1	0	0	0	0	0	1
997	4	0	0	1	0	0	1	0	0	0
998	4	0	1	0	0	0	0	0	0	1
999	4	0	0	0	1	0	1	0	0	0

1000 rows × 10 columns

In [ ]:
```
# Lets refit 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)
```

In [ ]:
```
model.score(x_test, y_test)
```

1.2 what if there were missing values ??

There's two main ways to deal with missing data

There's two main ways to deal with missing data:

- fill them with some value (Also known as imputation.)
- remove the samples with missing data altogether.

In [56]:
```
car_sales_missing = pd.read_csv("car-sales-extended-missing-data.csv")
car_sales_missing
```

Out[56]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

1000 rows × 5 columns

In [57]:
```
# For counting missing values
car_sales_missing.isna().sum()
```

Out[57]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [61]:
```
# Split / creating into x & y
x = car_sales_missing.drop('Price',axis = 1)
y = car_sales_missing["Price"]
```

In [71]:
```
# lets try and convert our data to numbers
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make","Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder = "passthrough")

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

Out[71]:
```
<950x15 sparse matrix of type '<class 'numpy.float64'>'
        with 3800 stored elements in Compressed Sparse Row format>
```

Option 1: Fill missing data with pandas

In [63]:
```
# Fill the "Make" column
car_sales_missing["Make"].fillna("missing", inplace = True)

# Fill the "Colour" column
car_sales_missing["Colour"].fillna("missing", inplace = True)

# Fill the "Odometer (KM)" column
car_sales_missing["Odometer (KM)"].fillna(car_sales_missing["Odometer (KM)"].mean(), inplace = True)

# Fill the "Odometer (KM)" column
car_sales_missing["Doors"].fillna(4, inplace = True)
```

In [64]:
```
# check out  dataframe again
car_sales_missing.isna().sum()
```

Out[64]:
```
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price           50
dtype: int64
```

In [65]:
```
# Remove rows with missing price value
car_sales_missing.dropna(inplace=True)
```

In [66]:
```
car_sales_missing.isna().sum()
```

Out[66]:
```
Make            0
Colour          0
Odometer (KM)   0
Doors           0
Price           0
dtype: int64
```

In [67]:
```
x = car_sales_missing.drop("Price", axis = 1)
y = car_sales_missing["Price"]
```

In [69]:
```
# lets try and convert our data to numbers
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make","Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder = "passthrough")
```

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

Out[69]:
```
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, ..., 0.00000e+00,
        8.47140e+04, 2.83430e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 0.00000e+00,
        6.66040e+04, 3.15700e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        2.15883e+05, 4.00100e+03],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        2.48360e+05, 1.27320e+04]])
```

Option 2: Fill missing values with Scikit-Learn

In [87]:
```
car_sales_missing = pd.read_csv("car-sales-extended-missing-data.csv")
car_sales_missing
```

Out[87]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

1000 rows × 5 columns

In [73]:
```
car_sales_missing.isna().sum()
```

Out[73]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [74]:
```
# drop the rows with no labels
car_sales_missing.dropna(subset=["Price"], inplace = True)
car_sales_missing.isna().sum()
```

Out[74]:
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
Price             0
dtype: int64
```

In [75]:
```
# Split / creating into x & y
x = car_sales_missing.drop('Price',axis = 1)
y = car_sales_missing["Price"]
```

In [88]:
```
# fill missing values with Scikit-Learn
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# Fill categorical values with 'missing' & numerical with mean
# hear we are just defining imputer, imputer is just filling missing data
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_features = ["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_features),
    ("num_imputer", num_imputer, num_features)
])


# Transform the data
filled_x = imputer.fit_transform(x)
filled_x
```

Out[88]:
```
array([['Honda', 'White', 4.0, 35431.0],
       ['BMW', 'Blue', 5.0, 192714.0],
       ['Honda', 'White', 4.0, 84714.0],
       ...,
       ['Nissan', 'Blue', 4.0, 66604.0],
       ['Honda', 'White', 4.0, 215883.0],
       ['Toyota', 'Blue', 4.0, 248360.0]], dtype=object)
```

In [89]:
```
car_sales_filled = pd.DataFrame(filled_x, columns = ["Make", "Colour","Doors", "Odometer (KM)" ])

car_sales_filled
```

Out[89]:

	Make	Colour	Doors	Odometer (KM)
0	Honda	White	4.0	35431.0
1	BMW	Blue	5.0	192714.0
2	Honda	White	4.0	84714.0
3	Toyota	White	4.0	154365.0
4	Nissan	Blue	3.0	181577.0
...	...	...	...	...

945	Toyota	Black	4.0	35820.0
946	missing	White	3.0	155144.0
947	Nissan	Blue	4.0	66604.0
948	Honda	White	4.0	215883.0
949	Toyota	Blue	4.0	248360.0

950 rows × 4 columns

In [90]:
```
car_sales_filled.isna().sum()
```

Out[90]:
```
Make            0
Colour          0
Doors           0
Odometer (KM)   0
dtype: int64
```

In [ ]:
```

```

In [91]:
```
# Lets try and convert our data to numbers
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

categorical_features = ["Make","Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder = "passthrough")

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

Out[91]:
```
<950x15 sparse matrix of type '<class 'numpy.float64'>'
        with 3800 stored elements in Compressed Sparse Row format>
```

In [92]:
```
# Now we've got our data as numbers and filled (no missing values)
# Letsfit 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()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[92]:
```
0.21990196728583944
```

2. choose the right estimator / algorithm for our problems

Scikit-learn users estimator as another term for machine learning model or algorithm.

Well some other things to note is that first of all before you choose an estimate slash algorithm for your problem is you have to figure out what kind of problem are you working with.

- `classification` predicting whether a sample is one thing or another. so classification is like our heart disease problem. We're trying to predict whether someone has heart disease or not.
- And `regression` predicting a number like with the Boston Housing dataset or with our car sales data set. We're trying to predict a house price or a car price

Step 1. Check the socket line machine learning map.

2.1 Picking a machine learning model for our regression problem

In [45]:
```
# import Boston housing dataset
from sklearn.datasets import load_boston
boston = load_boston()
boston
# So it imports as a dictionary we've got data as one of the keys. Target is one of the keys. And then we have a feature name
```

Out[45]:
```
{'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', 'DIS', '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 order):\n        - CRIM     per capita crime rate by town\n
- ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS    proportion of non-retail bus
iness acres per town\n        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NO
X      nitric oxides concentration (parts per 10 million)\n        - RM       average number of rooms per dwelling\n
- AGE      proportion of owner-occupied units built prior to 1940\n        - DIS      weighted distances to five Boston empl
oyment centres\n        - RAD      index of accessibility to radial highways\n        - TAX      full-value property-tax rat
e per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk is the proporti
on of black people by town\n        - LSTAT    % lower status of the population\n        - MEDV     Median value of owner-oc
cupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Rubinfeld, 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 Boston house-price data of Harriso
n, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-10
2, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are use
d in the table on\npages 244-261 of the latter.\n\nThe Boston house-price data has been used in many machine learning papers
that address regression\nproblems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostics:
Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance
-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, Universi
ty of Massachusetts, Amherst. Morgan Kaufmann.\n",
 'filename': 'C:\\Users\\toshiba c55t-a\\desktop\\sample_project\\env\\lib\\site-packages\\sklearn\\datasets\\data\\boston_h
ouse_prices.csv'}
```

In [46]:
```
boston_df = pd.DataFrame(boston["data"], columns =boston["feature_names"])
boston_df["target"] = pd.Series(boston["target"])
boston_df
```

Out[46]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [11]:
```
# Lets try the Ridge Regression model
from sklearn. linear_model import Ridge

#Setup random seed
np.random.seed(42)

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

# # split into train 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)

# Instantite Ridge model
model = Ridge()
model.fit(x_train,y_train)

# Check the score of the Ridge model on test data
model.score(x_test, y_test)
```

Out[11]: 0.6662221670168522

- How do weimprove this score
- what if Rigde wasn't working?

In that case we refer back to the map ... we will try another model / estemator / algorithms

In [12]:
```
# Let's  try the Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

# # split into train 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)
```

```
# Instantite Random Forest Regressor
model = RandomForestRegressor()
model.fit(x_train,y_train)

# Check the score of the Random Forest Regressor model on test data
# Evaluate the Random Forest Regressor
model.score(x_test, y_test)
```

Out[12]: 0.8654448653350507

2.2 Choosing and estimator for a classification problem

first of all visit map

In [13]:
```
# importing data
heart_disease = pd.read_csv("11.2 heart-disease.csv")
heart_disease.head()
```

Out[13]:

	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

Consulting the map and it says to try LinearSVC

In [16]:
```
# Import the LinearSVC estimator class
from sklearn.svm import LinearSVC

#Setup random seed
np.random.seed(42)

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

# # split into train 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)

# Instantite LinearSVC
clf = LinearSVC()
clf.fit(x_train,y_train)

# Check the score of the LinearSVC model on test data
# Evaluate the LinearSVC
clf.score(x_test, y_test)
```

```
C:\Users\toshiba c55t-a\desktop\sample_project\env\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear
failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[16]: 0.8688524590163934

Now comparing LinearSVC with Random Forest Regressor

In [19]:
```
# Let's  try the Random Forest Regressor for comparision with LinearSVC
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

# # split into train 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)

# Instantite Random Forest Regressor
model = RandomForestRegressor()
model.fit(x_train,y_train)

# Check the score of the Random Forest Regressor model on test data
# Evaluate the Random Forest Regressor
model.score(x_test, y_test)
```

Out[19]: 0.5106393318965518

Tidbit:

1. If you have structured data use ensemble method
2. If you have unstructured data, use deep learning or transfer learning.

3. fit the model / algorithm and use it to make predictions on our data.

3.1. Fitting the model to the data

Different name for:

- X = features, features variables, data
- y = labels, targets, target variables

In [28]:
```
# Import the RandomForestRegressor estimator class
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

```
# # split into train 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)

# Instantite Random Forest Regressor
model = RandomForestRegressor()

### Fit the model to the data (trainind the machine learning model)
model.fit(x_train,y_train)


# Evaluate the Random Forest Regressor (Ue=se the patterns the model has learned)
model.score(x_test, y_test)
```

Out[28]: 0.5106393318965518

In [20]:
```
x.head()
```

Out[20]:

	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 [21]:
```
y.head()
```

Out[21]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

3.2. Making predictions using a machine learning model

Two ways to make predictions

1. `predict()`
2. `predict_proba()`

Make predictions with `predict()`

In [31]:
```
# Using upper LinearSVC estimator class model
# Use a trained model to make predictions

clf.predict(np.array([1,7,8,3,4])) # this doesn't work....
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-3d4261e37655> in <module>
      2 # Use a trained model to make predictions
      3
----> 4 clf.predict(np.array([1,7,8,3,4])) # this doesn't work....

~\desktop\sample_project\env\lib\site-packages\sklearn\linear_model\_base.py in predict(self, X)
    307             Predicted class label per sample.
    308         """
--> 309         scores = self.decision_function(X)
    310         if len(scores.shape) == 1:
    311             indices = (scores > 0).astype(int)

~\desktop\sample_project\env\lib\site-packages\sklearn\linear_model\_base.py in decision_function(self, X)
    282         check_is_fitted(self)
    283
--> 284         X = check_array(X, accept_sparse='csr')
    285
    286         n_features = self.coef_.shape[1]

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\desktop\sample_project\env\lib\site-packages\sklearn\utils\validation.py 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)
    692             # If input is 1D raise error
    693             if array.ndim == 1:
--> 694                 raise ValueError(
    695                     "Expected 2D array, got 1D array instead:\narray={}.\n"
    696                     "Reshape your data either using array.reshape(-1, 1) if "

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 [25]:
```
x_test.head()
```

Out[25]:

	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 [30]:
```
# Using upper LinearSVC estimator class model
clf.predict(x_test)
```

Out[30]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [32]:
```
np.array(y_test)
```

Out[32]:
```
array([0, 0, 1, 0, 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 [33]:
```
# compare predictions to truth labels to evaluate the model
y_preds = clf.predict(x_test)
np.mean(y_preds == y_test)
```

Out[33]:
```
0.8688524590163934
```

In [44]:
```
clf.score(x_test,y_test)
```

Out[44]:
```
0.8688524590163934
```

In [45]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_preds)
```

Out[45]:
```
0.8688524590163934
```

Make predictions with `predict_proba()`

In [6]:
```
# predict_proba() returns probabilities of a classification label

clf.predict_proba(x_test[:5])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-3495c1924064> in <module>
      1 # predict_proba() returns probabilities of a classification label
      2
----> 3 clf.predict_proba(x_test[:5])

NameError: name 'clf' is not defined
```

`predict()` can also be used for regression models

In [61]:
```
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
model = RandomForestRegressor()
model.fit(x_train,y_train)

#Make predictions
y_preds = model.predict(x_test)
```

In [62]:
```
y_preds[:10]
```

Out[62]:
```
array([23.081, 30.574, 16.759, 23.46 , 16.893, 21.644, 19.113, 15.334,
       21.14 , 20.639])
```

In [63]:
```
np.array(y_test[:10])
```

Out[63]:
```
array([23.6, 32.4, 13.6, 22.8, 16.1, 20. , 17.8, 14. , 19.6, 16.8])
```

In [64]:
```
# Compare the predictions to the truth
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_preds)
```

Out[64]:
```
2.136382352941176
```

| this mean on averange, for every single prediction 2.1 away for the target |

4. Evaluating a model

There are 3 different APIs for evaluating the quality of a model's predictions:

1. Estimator `score` method
2. `Scoring` parameter
3. Metric functions / Problem-specific metric functions

4.1. Evaluating a model with the score method

Now we've already seen this one because this is basically the default. It's a way to get a quick sniff, a quick understanding of how our is doing.

In [12]:
```
from sklearn.ensemble import RandomForestClassifier

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
clf = RandomForestClassifier()
clf.fit(x_train,y_train)
```

```
Out[12]: RandomForestClassifier()
```

```
In [13]: # Evaluating
clf.score(x_train, y_train)
```

```
Out[13]: 1.0
```

```
In [14]: clf.score(x_test, y_test)
```

```
Out[14]: 0.8524590163934426
```

Let's do the same but for regression...

```
In [10]: from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
model = RandomForestRegressor()
model.fit(x_train,y_train)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-bea6f57ef8cf> in <module>
      5
      6 # create the data
----> 7 x = boston_df.drop("target", axis=1)
      8 y = boston_df["target"]
      9

NameError: name 'boston_df' is not defined
```

```
In [20]: model.score(x_test, y_test)
```

```
Out[20]: 0.8654448653350507
```

4.2. Evaluating a model using the `Scoring` parameter

```
In [12]: from sklearn.model_selection import cross_val_score ##

from sklearn.ensemble import RandomForestClassifier

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
clf = RandomForestClassifier()
clf.fit(x_train,y_train)
```

```
Out[12]: RandomForestClassifier()
```

```
In [22]: clf.score(x_test, y_test)
```

```
Out[22]: 0.8524590163934426
```

```
In [26]: cross_val_score(clf, x, y, cv=5)
```

```
Out[26]: array([0.78688525, 0.90163934, 0.78688525, 0.81666667, 0.8       ])
```

```
In [27]: np.random.seed(42)

# Single training and test split score
clf_single_score = clf.score(x_test,y_test)

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

# compare the two
clf_single_score, clf_cross_val_score
```

```
Out[27]: (0.8524590163934426, 0.8248087431693989)
```

4.2.1 Classification model evaluation metrics

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

1. Accuracy

```
In [6]: from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

#Setup random seed
np.random.seed(42)

# create the data
```

```
x = heart_disease.drop("target", axis=1)
y = heart_disease["target"]


# Instantite Random Forest Classifier
clf = RandomForestClassifier()
cross_val_score = cross_val_score(clf, x, y)
```

In [7]:
```
#mean accuracy of the model
np.mean(cross_val_score)
```

Out[7]:
```
0.8248087431693989
```

In [8]:
```
print(f"Heart Disease Classifier Cross-Validated Accuracy: {np.mean(cross_val_score) *100:.2f}%" )
```

```
Heart Disease Classifier Cross-Validated Accuracy: 82.48%
```

that maean 82.48%, are model will pridict the right label so that of how you would present your models accuracy in print out

---

2. Area under ROC curve

Area under the receiver operating characteristic curve **(AUC/ROC)**.

- Area under curve (AUC)
- ROC curve

ROC curves are a comparision of a model's true positive rate (tpr) versus a model false positive rate (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 [13]:
```
from sklearn.metrics import roc_curve

# Make predictions with probabilities
y_probs = clf.predict_proba(x_test)

y_probs[:10]
```

Out[13]:
```
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]])
```

ROC curves are a comparision of a model's **true positive rate (tpr)** versus a model **false positive rate (fpr)** So, we only want probabilities that the model has predicted for the positive class. And we wants Right values (index 1), so we use slicing

In [14]:
```
y_probs_positive = y_probs[:, 1]
y_probs_positive[:10]
```

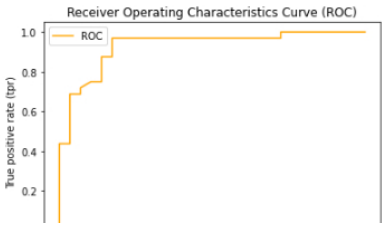
Out[14]:
```
array([0.11, 0.51, 0.57, 0.16, 0.82, 0.86, 0.64, 0.05, 0.01, 0.53])
```

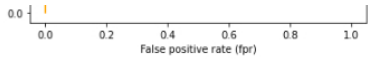
In [15]:
```
# Calculating fpr, tpr and thresholds
fpr, tpr, thresholds = roc_curve(y_test,y_probs_positive)

# Check the false positive rate
fpr
```

Out[15]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.03448276, 0.03448276, 0.03448276, 0.03448276, 0.06896552,
       0.06896552, 0.10344828, 0.13793103, 0.13793103, 0.17241379,
       0.17241379, 0.27586207, 0.4137931 , 0.48275862, 0.55172414,
       0.65517241, 0.72413793, 0.72413793, 0.82758621, 1.        ])
```

Looking at these on its own doesn't really make much sense, it's much easier to see it visually.

In [18]:
```
# Create a function for plotting ROC curves
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):
    """
    Plots a ROC curve givrn the false positive rate (fpr)
    and true positive rate (tpr) of a model
    """

    # Plot roc curve
    plt.plot(fpr, tpr, color = "orange", label="ROC")
    # Plot line with no predictive power (baseline)
    #plt.plot([0,1],[0,1], color='darkblue', linestyle="--",label="Guessing")

    # customize the plot
    plt.xlabel("False positive rate (fpr)")
    plt.ylabel("True positive rate (tpr)")
    plt.title("Receiver Operating Characteristics Curve (ROC)")
    plt.legend()
    plt.show


plot_roc_curve(fpr, tpr)
```

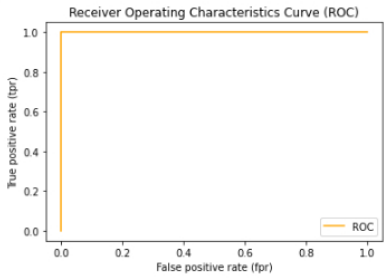




In [17]:
```
# auc = area under curve
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_probs_positive)
```

Out[17]:
```
0.9304956896551724
```

In [19]:
```
# Plot perface ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test,y_test)
# manually build
plot_roc_curve(fpr, tpr)
```



In [20]:
```
# perfect auc score
roc_auc_score(y_test, y_test)
```

Out[20]:
```
1.0
```

3. Confustion Matrix

A `confusion` matrix is a quick way to compare the labels a model predicts and the actual labels it was supposed to predict.

In essence, giving you an idea of where the model is getting confused.

In [22]:
```
from sklearn.metrics import confusion_matrix

y_preds = clf.predict(x_test)

confusion_matrix(y_test, y_preds)
```

Out[22]:
```
array([[24,  5],
       [ 4, 28]], dtype=int64)
```

In [23]:
```
# Visualize confusion matrix with pd.crosstab()
pd.crosstab(y_test,
            y_preds,
            rownames=["Actual Labels"],
            colnames=["Predicted Labels"] )
```

Out[23]:

Predicted Labels	0	1
Actual Labels		
0	24	5
1	4	28

So in our case where the actual label is 0 and the predicted label is 0 we have 22 examples and then where the predicted label is 1 and the actual label is 1 we have 24 examples

And now if we total all of these up 24 + 5 + 4 + 28 = 61 and our x_test / and their pridiction is also 61

In this case we have

- 4 False negative
- 5 False positive
- 24 True negitive
- 28 true positive

In [28]:
```
24 + 5 + 4 + 28
```

Out[28]:
```
61
```

In [29]:
```
len(y_preds) , len(x_test)
```

Out[29]:
```
(61, 61)
```

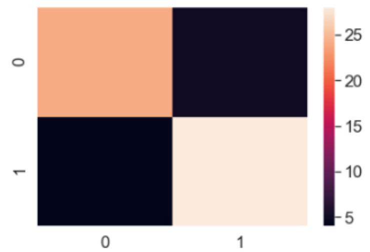
Make our confusion matrix more visual with Seaborn's heatmap() Seaborn heatmap plot rectangular data as a color-encoded matrix. Seabourn is a visualization library that is built on top of matplotlib and it's pretty relatively easy to use but we're going to mostly just take care of the heatnmap function

In [34]:
```
# installing libary/ package within jupyter notebook
# import sys
# !conda install --yes --prefix {sys.prefix} seaborn
```

```
EnvironmentLocationNotFound: Not a conda environment: C:\Users\toshiba
```

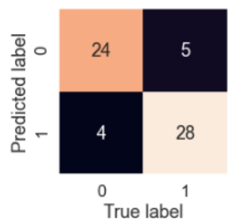
In [35]:
```
# install seaborn module using anaconda command
# Make our confusion matrix more visual with Seaborn's heatmap()

import seaborn as sns

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

# Creat a confusion matrix
conf_mat = confusion_matrix(y_test, y_preds)

# plot it using seaborn
sns.heatmap(conf_mat)
```

Out[35]: <AxesSubplot:>



Adding information to confusion mtrix

In [40]:
```
def plot_conf_mat(conf_mat):
    """
    Plot a confusion matrix using Seaborn's heatmap()
    """

    fig, ax = plt.subplots(figsize=(3,3))
    ax = sns.heatmap(conf_mat,
                     annot = True, #Annotate the boxes with conf_map info
                     cbar = False)

    plt.xlabel("True label")
    plt.ylabel("Predicted label")


plot_conf_mat(conf_mat)
```



Classification Report

Classification report is also a collection of different evaluation metrics rather than a single one.

In [42]:
```
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

           0       0.86      0.83      0.84        29
           1       0.85      0.88      0.86        32

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

Lets see a scenario of **Classification Report**. So for example, let's say there were 10000 people and one of them had a disease and you're asked to build a model to predict who has it.

So this is where precision and recall become valuable and in fact all the metrics in our classification report become valuable.

In [43]:
```
# where precision and recall become valuable
disease_true = np.zeros(10000)
disease_true[0] = 1 # only one positive value

disease_preds = np.zeros(10000) # model pridicts every case as 0

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

```
C:\Users\toshiba c55t-a\desktop\sample_project\env\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\toshiba c55t-a\desktop\sample_project\env\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\toshiba c55t-a\desktop\sample_project\env\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[43]:

	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

To summarize classifacation metrics

- **Accuracy** is a good measure to start with if all classes are balanced (e.g same amount of samples which are labelled with 0 or 1).
- **Precision** and **recall** become more important when classes are imbalanced
- If false positive predictions are worse then false negitives, aim for higher precision.
- If false negitive predictions are worse then false positives, aim for higher recall
- **F1-score** is a combination of precision and recall

4.2.2. Regression model evaluation metrics

RandomForestRegressor

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

In [32]:
```
# import Boston housing dataset
from sklearn.datasets import load_boston
boston = load_boston()
# So it imports as a dictionary we've got data as one of the keys. Target is one of the keys. And then we have a feature name
```

In [33]:
```
boston_df = pd.DataFrame(boston["data"], columns =boston["feature_names"])
boston_df["target"] = pd.Series(boston["target"])
boston_df
```

Out[33]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [4]:
```
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
model = RandomForestRegressor()
model.fit(x_train,y_train)
```

Out[4]:
```
RandomForestRegressor()
```

In [5]:
```
model.score(x_test, y_test)
```

Out[5]:
```
0.8654448653350507
```

1. R^2 (pronounced r-squared) or coefficient of determination

What R-squared does: Compares your model's predictions to the mean of the target. Values of R^2 can range from negative infinity(a very poor model) to 1 For example, if all your model does is predict the mean of the targets, it's a R^2 value would be 0. And if your model perfectly predicts a range of numbers it's R^2 value would be 1.

In [6]:
```
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 [8]:
```
r2_score(y_test, y_test_mean)
```

Out[8]:
```
2.220446049250313e-16
```

In [10]:
```
r2_score(y_test, y_test)
```

Out[10]:
```
1.0
```

2. Mean absolute error (MAE)

In [7]:
```
# Mean absolute error (MAE)
from sklearn.metrics import mean_absolute_error

y_preds = model.predict(x_test)
mea = mean_absolute_error(y_test ,y_preds)
mea
```

Out[7]:
```
2.136382352941176
```

In [9]:
```
df = pd.DataFrame(data = {"actual values": y_test,
                          "predicted values": y_preds
})
df
```

Out[9]:

	actual values	predicted values
173	23.6	23.081
274	32.4	30.574
491	13.6	16.759
72	22.8	23.460
452	16.1	16.893
...	...	...
412	17.9	13.159

436	9.6	12.476
411	17.2	13.612
86	22.5	20.205
75	21.4	23.832

102 rows × 2 columns

In [11]:
```
df["differences"] = df["predicted values"] - df["actual values"]
df
```

Out[11]:

	actual values	predicted values	differences
173	23.6	23.081	-0.519
274	32.4	30.574	-1.826
491	13.6	16.759	3.159
72	22.8	23.460	0.660
452	16.1	16.893	0.793
...	...	...	...
412	17.9	13.159	-4.741
436	9.6	12.476	2.876
411	17.2	13.612	-3.588
86	22.5	20.205	-2.295
75	21.4	23.832	2.432

102 rows × 3 columns

Mean Absolute Error (MAE) is the average of the absolute differences between predictions and actual values. So, it gives you an idea of how wrong your model predictions are.

3. Mean squared error

MSE will always be higher than mean absolute error because it squares the errors rather than only taking the absolute difference

In [12]:
```
# Mean squared error
from sklearn.metrics import mean_squared_error

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

Out[12]: 9.867437068627442

In [15]:
```
# Calculate MSE by hand (manually)
squared = np.square(df["differences"])
squared.mean()
```

Out[15]: 9.867437068627439

4.2.3 Finally using the `scoring` parameter

In [19]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
clf = RandomForestClassifier()
```

In [20]:
```
np.random.seed(42)
cv_acc = cross_val_score(clf, x, y, cv=5 )
cv_acc
```

Out[20]: array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])

In [21]:
```
# Cross-validated accuracy
print(f'The cross-validated accuracy is: {np.mean(cv_acc)*100:.2f}%')
```

```
The cross-validated accuracy is: 82.48%
```

In [22]:
```
np.random.seed(42)
cv_acc = cross_val_score(clf, x, y, cv=5, scoring="accuracy" )
print(f'The cross-validated accuracy is: {np.mean(cv_acc)*100:.2f}%')
```

```
The cross-validated accuracy is: 82.48%
```

In [26]:
```
# precision
cv_precision = cross_val_score(clf, x, y, cv=5, scoring="precision" )
cv_precision.mean()
```

Out[26]: 0.8222673160173161

In [29]:
```
# recall
cv_recall = cross_val_score(clf, x, y, cv=5, scoring="recall" )
np.mean(cv_recall)
```

Out[29]: 0.8606060606060606

In [31]:
```
cv_f1 = cross_val_score(clf, x, y, cv=5, scoring="f1" )
np.mean(cv_f1)
```

Out[31]: 0.8252798417167047

How about regression model?

In [34]:
```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

model = RandomForestRegressor()
```

In [40]:
```
np.random.seed(42)
cv_r2 = cross_val_score(model, x, y, cv=5, scoring="r2")
np.mean(cv_r2)
```

Out[40]: 0.6243870737930857

In [43]:
```
# Mean absolute error
cv_mae = cross_val_score(model, x, y, cv=5, scoring="neg_mean_absolute_error")
cv_mae
```

Out[43]: array([-2.13045098, -2.49771287, -3.45471287, -3.81509901, -3.11813861])

In [46]:
```
# Mean squared error
cv_mse = cross_val_score(model, x, y, cv=5, scoring="neg_mean_squared_error")
cv_mse.mean()
```

Out[46]: -21.729149843894373

4.3. Using different evaluation matrics as Scikit-Learn functions

Classification evaluation functions

In [48]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier

#Setup random seed
np.random.seed(42)

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

# # split into train 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)

# Instantite
clf = RandomForestClassifier()
clf.fit(x_train,y_train)

# Make some predictions
y_preds = clf.predict(x_test)

# Evaluate the classifier
print("Clssifier metrics on the test set")
print(f"Accuracy: {accuracy_score(y_test, y_preds)}")
print(f"Presision: {precision_score(y_test, y_preds)}")
print(f"Recall: {recall_score(y_test, y_preds)}")
print(f"F1: {f1_score(y_test, y_preds)}")
```

```
Clssifier metrics on the test set
Accuracy: 0.8524590163934426
Presision: 0.8484848484848485
Recall: 0.875
F1: 0.8615384615384615
```

Regression Evaluation Function

In [49]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor

#Setup random seed
np.random.seed(42)

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

# split into train 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)

# Instantite Random Forest Regressor and fit model
model = RandomForestRegressor()
model.fit(x_train,y_train)

# Make predictions using our regression model
y_preds = model.predict(x_test)

# Evaluating the regression model
print("Regression model metrics on the test set")
print(f"R^2: {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 model metrics on the test set
R^2: 0.8654448653350507
MAE: 2.136382352941176
MSE: 9.867437068627442
```

5. Improving a model

- first predictions = baseline predictions
- first model = baseline model

There's two main ways to improve the model

1. From a data perspective.

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

If there's 10000 examples rather than 1000 example of something chances are if there's patterns in that data the machine learning model will pick them up.

- Improve our data

for example an hour car sales problem where we're using the make the color the odometer and the number of doors to try and predict the sale price of a car.So what you would search for here is maybe more features about each car so you would have more information about each sample.

2. From a model perspective

- Is there a better model we could use?
- Could we improve the current model?

parameters vs. Hyperparameters

parameters = model finds these patterns in data

hyperparameters = settings on a model you can adjust to (potentially) improve its ability to find patterns

Three ways to adjust hyperparameters

1. By hand
2. Randomly with RandomSearchCV
3. Exhaustively with GridSearchCV

In [1]:
```
# How to find models hyperparameters
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
#Once the model is Instantited, we can find hyperparameters by calling a function

clf.get_params()

# sklearn calls hyperparameters as parameters
# Every models has it's own parameters / hyperparameters
```

Out[1]:
```
{'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}
```

5.1 Tuning Hyperparameters by hand

So far we've talked about dealing with training and test data sets a model gets trained on the training set, it finds patterns and then it gets evaluated on the test set. So, it uses those patterns but hyper parameter tuning introduces a third set called as a **validation set**.

Let's make 3 sets training, vaidation and test

In [2]:
```
clf.get_params()
```

Out[2]:
```
{'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}
```

we're going to try and adjust

- `max_depth`
- `max_features`
- `min_samples_leaf`
- `min_samples_split`
- `n_estimators`

In [25]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
def evaluate_preds(y_true, y_preds):
    """
    performs evaluation comparison on y_true labels vs. y_pred labels.
    on a classification models
    """

    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)

    metric_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:{f1:.2f}")

    return metric_dict
```

In [26]:
```
from sklearn.ensemble import RandomForestClassifier

#Setup random seed
np.random.seed(42)

#Shuffle the data
heart_disease_shuffled = heart_disease.sample(frac=1)


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

# split into train, validation and test sets
# We have to do it manually
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) # Checking

clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Make baseline predictions
y_preds = clf.predict(X_valid)
# We predict on the validation data because we want to tune our model on the validation split.

# Evaluate the classifier on validation set
# evaluate_preds manually crated function
baseline_metrics = evaluate_preds(y_valid, y_preds)
baseline_metrics
```

```
Acc:82.22%
precision:0.81
recall:0.88
f1:0.85
```

Out[26]:
```
{'accuracy': 0.82, 'precision': 0.81, 'recall': 0.88, 'f1': 0.85}
```

Adjusting hyperparameters by hand

In [27]:
```
np.random.seed(42)

# Create a second classifier with different hyperparameters
clf_2 = RandomForestClassifier(n_estimators=100)
clf_2.fit(X_train, y_train)
#diffrent model same data

# Make predictions with diffrent hyperparameters
y_preds_2 = clf_2.predict(X_valid)


# Evalute the 2nd classifier
clf_2_metrices = evaluate_preds(y_valid, y_preds_2)
```

```
Acc:82.22%
precision:0.84
recall:0.84
f1:0.84
```

In [28]:
```
clf_3 = RandomForestClassifier(n_estimators=100, max_depth=10)
clf_3.fit(X_train, y_train)
#diffrent model same data

# Make predictions with diffrent hyperparameters
y_preds_3 = clf_3.predict(X_valid)


# Evalute the 3nd classifier
clf_3_metrices = evaluate_preds(y_valid, y_preds_3)
```

```
Acc:82.22%
precision:0.81
recall:0.88
f1:0.85
```

5.2 Hyperparameter tuning with with RandomSearchCV (RandomizedSearchCV)

In [30]:
```
from sklearn.model_selection import RandomizedSearchCV

grid = {
    "n_estimators":[10,100,200,1000,1200],
    "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)
x = heart_disease_shuffled.drop("target", axis=1)
y = heart_disease_shuffled["target"]

# split into train 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)

# Instantite Random Forest Classifier
clf = RandomForestClassifier(n_jobs = 1)
```

```
# Setup RandomSearchCV
rs_clf = RandomizedSearchCV(estimator = clf,
                            param_distributions = grid,
                            n_iter=10, # number of models to try
                            cv = 5,
                            verbose = 2)

# fit the RandomSearchCV 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=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.2s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.1s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.4s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.2s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.3s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=1000; total time=   4.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=1200; total time=   5.2s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=1200; total time=   5.6s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=1200; total time=   5.5s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=1200; total time=   5.1s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=1200; total time=   5.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.4s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=4, min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=4, min_samples_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=4, min_samples_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=4, min_samples_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=4, min_samples_split=6, n_estimators=100; total time=   0.4s
```

Out[30]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(n_jobs=1),
                   param_distributions={'max_depth': [None, 5, 10, 20, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 4, 6],
                                        'n_estimators': [10, 100, 200, 1000,
                                                         1200]},
                   verbose=2)
```

In [31]:
```
# Which combinations of these give the best result
rs_clf.best_params_
```

Out[31]:
```
{'n_estimators': 1200,
 'min_samples_split': 4,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 5}
```

In [32]:
```
# Making 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:83.61%
precision:0.78
recall:0.89
f1:0.83
```

5.3 Hyperparameter tuning with GridSearchCV

In [45]:
```
grid
```

Out[45]:
```
{'n_estimators': [10, 100, 200, 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]}
```

Key difference here between randomize search and grid search CV is that randomize search CV has a parameter called n_iter.which we can set to limit the number of models to try. So, in our case we used 10. GridSearchCV will go through every single combination that is available here.

we less the parameter, to compute less, if we compute more parameters it uses more system resources

In [16]:
```
grid_2 = { 'n_estimators': [200, 500],
           'max_depth': [None],
           'max_features': ['auto', 'sqrt'],
           'min_samples_split': [6],
           'min_samples_leaf': [1, 2]}
print("Run")
```

Run

In [15]:
```
from sklearn.model_selection import GridSearchCV, train_test_split

np.random.seed(42)

x = heart_disease_shuffled.drop("target", axis=1)
y = heart_disease_shuffled["target"]

# split into train 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)

# Instantite Random Forest Classifier
clf = RandomForestClassifier(n_jobs = 1)

# Setup GridSearchCV
gs_clf = GridSearchCV(estimator = clf,
                      param_grid = grid_2,
                      cv = 5,
                      verbose = 2)

# fit the GridSearchCV version of clf

gs_clf.fit(x_train, y_train)
print("Run")
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; total time=   0.9s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; total time=   2.1s
Run
```

In [18]:
```
gs_clf.best_params_
```

Out[18]:
```
{'max_depth': None,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 6,
 'n_estimators': 200}
```

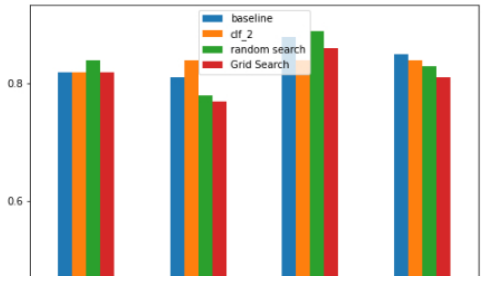
In [20]:
```
gs_y_preds = gs_clf.predict(x_test)

# evaluating the prsdictions
gs_metrics = evaluate_preds(y_test, gs_y_preds)
```

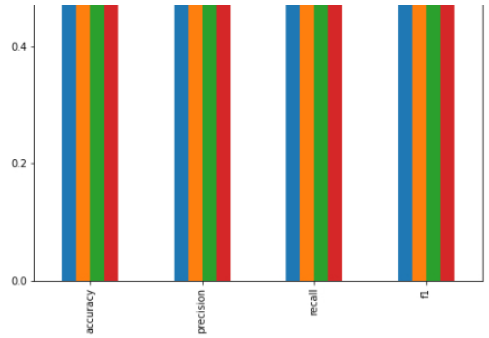
```
Acc:81.97%
precision:0.77
recall:0.86
f1:0.81
```

Let's compare our different models metrics

In [36]:
```
compare_metrics = pd.DataFrame({
    "baseline": baseline_metrics,
    "clf_2":clf_2_metrices,
    "random search": rs_metrics,
    "Grid Search":gs_metrics
})

compare_metrics.plot.bar(figsize=(8,10))
```

Out[36]:
```
<AxesSubplot:>
```





6. Saving and loading Trained machine learning models

Two ways to save and load machine learning models.

1. With Python's `pickle` module.
2. with the `joblib` module.

pickle module

In [42]:
```
import pickle

# Save an extisting model to file
pickle.dump(gs_clf, open("gs_random_forest_model_1.pk1","wb"))
```

In [43]:
```
# Load a saved model
loaded_pickle_model = pickle.load(open("gs_random_forest_model_1.pk1","rb"))
```

In [44]:
```
# Make some predictions
pickle_y_Preds = loaded_pickle_model.predict(x_test)
evaluate_preds(y_test, pickle_y_Preds)
```

```
Acc:81.97%
precision:0.77
recall:0.86
f1:0.81
```

Out[44]:
```
{'accuracy': 0.82, 'precision': 0.77, 'recall': 0.86, 'f1': 0.81}
```

joblib module.

In [45]:
```
from joblib import dump, load

# Save model to file
dump(gs_clf, filename="gs_random_forest_model_2.joblib")
```

Out[45]:
```
['gs_random_forest_model_2.joblib']
```

In [46]:
```
# Import a saved joblib model
loaded_joblib_model = load(filename="gs_random_forest_model_2.joblib")
```

In [48]:
```
# Make and evaluate joblib model
joblib_y_preds = loaded_joblib_model.predict(x_test)
evaluate_preds(y_test, joblib_y_preds)
```

```
Acc:81.97%
precision:0.77
recall:0.86
f1:0.81
```

Out[48]:
```
{'accuracy': 0.82, 'precision': 0.77, 'recall': 0.86, 'f1': 0.81}
```

7. Putting it all together.

In [49]:
```
data = pd.read_csv("car-sales-extended-missing-data.csv")
data
```

Out[49]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

1000 rows × 5 columns

In [50]:
```
data.dtypes
```

Out[50]:
```
Make              object
Colour            object
Odometer (KM)    float64
Doors            float64
Price            float64
dtype: object
```

In [51]:
```
# Null values
data.isna().sum()
```

Out[51]:
```
Make            49
Colour          50
Odometer (KM)   50
Doors           50
Price           50
dtype: int64
```

Steps we want to do (all in one)

1. Fill missing data
2. Convert data to numbers
3. Build a model on the data

In [62]:
```python
# Getting data ready
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Modelling
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV

# Setup random seed
import numpy as np
np.random.seed(42)

# Import data and drop rows with missing labels
data = pd.read_csv("car-sales-extended-missing-data.csv")
data.dropna(subset = ["Price"], inplace = True)

# Define different features and transformer pipeline
categorical_features = ["Make", "Colour"]
categorical_transformer = Pipeline(steps=[
    ("imputer",SimpleImputer(strategy="constant", fill_value="missing")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])


door_features = ["Doors"]
door_transformer = Pipeline(steps=[
    ("imputer",SimpleImputer(strategy="constant", fill_value=4))

])

numaric_feature = ["Odometer (KM)"]
numaric_transformer = Pipeline(steps=[
    ("imputer",SimpleImputer(strategy="mean"))
])


# Setup preprocessing steps fill missing values, then convert to numbers
preprocessor = ColumnTransformer(
transformers=[
      ("cat",categorical_transformer, categorical_features),
      ("door",door_transformer, door_features ),
      ("num",numaric_transformer, numaric_feature)
])


# Creating a preprocessing and modelling pipeline
model = Pipeline(steps=[("preprocessor",preprocessor),
                       ("model",RandomForestRegressor())])

# Split data
X = data.drop("Price", axis=1)
y = data["Price"]

# split into train 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)

# Fit and score the model
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Out[62]: `0.22188417408787875`

It's also possible to use `GridSearchCV` or `RandomizedSearchCV` with our `Pipeline`

In [64]:
```python
# Use GridSearchCV with our regression Pipeline
from sklearn.model_selection import GridSearchCV

pipe_grid = {
    "preprocessor__num__imputer__strategy": ["mean", "median"],
    "model__n_estimators": [100, 1000],
    "model__max_depth":[None, 5],
    "model__max_features":["auto"],
    "model__min_samples_split":[2, 4]

}

gs_model = GridSearchCV(model, pipe_grid, cv=5, verbose=2)
gs_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=mean; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=mean; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=mean; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=mean; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=mean; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=median; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=median; total time=   0.9s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=median; total time=   0.8s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
or__num__imputer__strategy=median; total time=   0.7s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimators=100, preprocess
```

In [65]:
```
gs_model.score(X_test, y_test)
```

Out[65]:
```
0.3339554263158365
```

In [ ]:
```

```