Introduction to Scikit-Learn (sklearn)

This notebook demonstrates some of the most useful functions of the beautiful Scikit-Learn library.

What I'm 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. Puting it all together!

In [37]:
```
what_iam_covering = ["0. An end-to-end Scikit-Learn workflow", "1. Gettin
# Exibir todos os itens
for item in what_iam_covering:
    print(item)
```

```
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!
```

In [38]:
```
# Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

0. An end-to-end Scikit-Learn workflow

In [39]:
```
# 1. Get the data ready
import pandas as pd
heart_disease = pd.read_csv("/home/user/heart-disease.csv")
heart_disease
```

Out[39]:

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

303 rows × 14 columns

In [40]:

```
# Create X (features matrix)
X = heart_disease.drop("target",axis =1)

# Create Y (features matrix)
y = heart_disease["target"]
```

In [41]:

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in ./miniconda3/envs/env/lib/python3.12/site-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in ./miniconda3/envs/env/lib/python3.12/site-packages (from scikit-learn) (2.2.4)
Requirement already satisfied: scipy>=1.6.0 in ./miniconda3/envs/env/lib/python3.12/site-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in ./miniconda3/envs/env/lib/python3.12/site-packages (from scikit-learn) (1.5.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in ./miniconda3/envs/env/lib/python3.12/site-packages (from scikit-learn) (3.6.0)
```

In [42]:

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100)
clf.get_params()
```

Out[42]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [43]:
```
# 3. Fit the model to the training data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
```

In [44]:
```
import sklearn
sklearn.show_versions()
```

```
System:
    python: 3.12.9 | packaged by Anaconda, Inc. | (main, Feb  6 2025, 18:5
6:27) [GCC 11.2.0]
executable: /home/user/miniconda3/envs/env/bin/python
   machine: Linux-6.8.0-58-generic-x86_64-with-glibc2.35

Python dependencies:
      sklearn: 1.6.1
          pip: 25.0
   setuptools: 75.8.0
        numpy: 2.2.4
        scipy: 1.15.3
       Cython: None
       pandas: 2.2.3
   matplotlib: 3.10.0
       joblib: 1.5.0
threadpoolctl: 3.6.0

Built with OpenMP: True

threadpoolctl info:
       user_api: blas
   internal_api: mkl
    num_threads: 2
         prefix: libmkl_rt
       filepath: /home/user/miniconda3/envs/env/lib/libmkl_rt.so.2
        version: 2023.1-Product
threading_layer: intel

       user_api: openmp
   internal_api: openmp
    num_threads: 2
         prefix: libiomp
       filepath: /home/user/miniconda3/envs/env/lib/libiomp5.so
        version: None

       user_api: blas
   internal_api: openblas
    num_threads: 2
         prefix: libscipy_openblas
       filepath: /home/user/miniconda3/envs/env/lib/python3.12/site-packag
es/scipy.libs/libscipy_openblas-68440149.so
        version: 0.3.28
threading_layer: pthreads
   architecture: Nehalem

       user_api: openmp
   internal_api: openmp
    num_threads: 2
         prefix: libgomp
       filepath: /home/user/miniconda3/envs/env/lib/python3.12/site-packag
es/scikit_learn.libs/libgomp-a34b3233.so.1.0.0
        version: None
```

In [45]:
```
clf.fit(X_train,y_train)
```

Out[45]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [46]:

```
X_test
```

Out[46]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
179	57	1	0	150	276	0	0	112	1	0.6	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0
111	57	1	2	150	126	1	1	173	0	0.2	2	1
246	56	0	0	134	409	0	0	150	1	1.9	1	2
60	71	0	2	110	265	1	0	130	0	0.0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...
249	69	1	2	140	254	0	0	146	0	2.0	1	3
104	50	1	2	129	196	0	1	163	0	0.0	2	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2
193	60	1	0	145	282	0	0	142	1	2.8	1	2
184	50	1	0	150	243	0	0	128	0	2.6	1	0

61 rows × 13 columns

In [47]:

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

Out[47]:

```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0])
```

In [48]:

```
# 4. Evaluate the model
clf.score (X_train, y_train)
```

Out[48]:

```
1.0
```

In [49]:

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

Out[49]:

```
0.8524590163934426
```

In [50]:

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

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

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        29
           1       1.00      1.00      1.00        32

    accuracy                           1.00        61
   macro avg       1.00      1.00      1.00        61
weighted avg       1.00      1.00      1.00        61
```

In [51]:
```
confusion_matrix(y_test, y_preds)
```

Out[51]:
```
array([[24,  5],
       [ 4, 28]])
```

In [52]:
```
#5. Improve a model
# Try different amount of n_estimators
import numpy as np
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 accurancy of test set: {clf.score(X_test, y_test)*100:.
```

```
Trying model with 10 estimators...
Model accurancy of test set: 85.25%
Trying model with 20 estimators...
Model accurancy of test set: 80.33%
Trying model with 30 estimators...
Model accurancy of test set: 83.61%
Trying model with 40 estimators...
Model accurancy of test set: 80.33%
Trying model with 50 estimators...
Model accurancy of test set: 86.89%
Trying model with 60 estimators...
Model accurancy of test set: 83.61%
Trying model with 70 estimators...
Model accurancy of test set: 83.61%
Trying model with 80 estimators...
Model accurancy of test set: 83.61%
Trying model with 90 estimators...
Model accurancy of test set: 81.97%
```

In [53]:
```
#6. Save a model and load it
import pickle

pickle.dump(clf,open("random_forest_model_1.pkl","wb"))
```

In [54]:
```
loaded_model = pickle.load(open("random_forest_model_1.pkl","rb"))
loaded_model.score(X_test,y_test)
```

Out[54]:
```
0.819672131147541
```

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

Three main things we have to do:

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

In [55]:
```
heart_disease.head()
```

Out[55]:

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

In [56]:
```
X = heart_disease.drop ("target",axis = 1)
X.head()
```

Out[56]:

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

In [57]:
```
y = heart_disease["target"]
y.head()
```

Out[57]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [174…
```
# Split the data into training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split (X,y,test_size = 0.2)
```

In [59]:
```
X_train.shape, X_test.shape,y_train.shape, y_test.shape
```

Out[59]:
```
((242, 13), (61, 13), (242,), (61,))
```

In [60]:
```
X.shape[0]* 0.8
```

Out[60]:
```
242.4
```

1.1 Make sure it's all numerical

In [61]:
```
car_sales = pd.read_csv("/home/user/Downloads/car-sales-extended.csv")
car_sales.head()
```

Out[61]:

	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 [62]:
```
len(car_sales)
```

Out[62]:
```
1000
```

In [63]:
```
car_sales.dtypes
```

Out[63]:
```
Make              object
Colour            object
Odometer (KM)      int64
Doors              int64
Price              int64
dtype: object
```

Let's convert the Make and Colours columns in numbers

We will try to predict (here we are using machine learning) the price car's model with only Make, Colour, Odomeeter, Doors, is it possible?

In [187…
```
# 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)
transformed_X = transformer.fit_transform(X)
transformed_X
```

Out[187…
```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 3800 stored elements and shape (950, 16)>
```

In [188…
```
X.head()
```

Out[188…

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

In [186…

```
pd.DataFrame(transformed_X)
```

Out[186…

	0
0	<Compressed Sparse Row sparse matrix of dtype ...
1	<Compressed Sparse Row sparse matrix of dtype ...
2	<Compressed Sparse Row sparse matrix of dtype ...
3	<Compressed Sparse Row sparse matrix of dtype ...
4	<Compressed Sparse Row sparse matrix of dtype ...
...	...
945	<Compressed Sparse Row sparse matrix of dtype ...
946	<Compressed Sparse Row sparse matrix of dtype ...
947	<Compressed Sparse Row sparse matrix of dtype ...
948	<Compressed Sparse Row sparse matrix of dtype ...
949	<Compressed Sparse Row sparse matrix of dtype ...

950 rows × 1 columns

The `car_sales` table, the `Make`, `Colour` and `Doors` columns have been transformed into columns with numbers. Note that the columns in y only appear three '1' in each row. This means that they are the three columns mentioned

In [189…

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

Out[189…

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

1000 rows × 10 columns

1.2 What if there were missing values?

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

In [192…

```
# Import car sales missing data
car_sales_missing = pd.read_csv("/home/user/Downloads/car-sales-extended-
car_sales_missing.head()
```

Out[192…

	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

In [182…

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

Out[182…

```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [193…

```
# Create X & y
X = car_sales_missing.drop("Price", axis = 1)
y = car_sales_missing["Price"]
```

In [194…

```
# Let's try to convert our data to 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)
transformed_X = transformer.fit_transform(X)
transformed_X
```

Out[194…

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 4000 stored elements and shape (1000, 16)>
```

Option 1: Fill missing data with Pandas

In [195…

```
car_sales_missing["Doors"].value_counts()
```

Out[195…

```
Doors
4.0    811
5.0     75
3.0     64
Name: count, dtype: int64
```

In [196…

```
# Fill the "Make" column
car_sales_missing["Make"] = car_sales_missing["Make"].fillna("missing")
# Fill the "Colour" column
car_sales_missing["Colour"] = car_sales_missing["Colour"].fillna("missing
```

```python
# Fill the "Odometer (KM)" column
car_sales_missing["Odometer (KM)"] = car_sales_missing["Odometer (KM)"].f
#Fill the "Doors" column
car_sales_missing["Doors"] = car_sales_missing["Doors"].fillna(4)
```

In [197…

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

Out[197…

```
Make              0
Colour            0
Odometer (KM)     0
Doors             0
Price            50
dtype: int64
```

In [198…

```python
len(car_sales_missing)
```

Out[198…

```
1000
```

In [199…

```python
# Remove rows with missing Price value
car_sales_missing.dropna(inplace = True)
```

In [200…

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

Out[200…

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

In [201…

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

In [202…

```python
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)
transformed_X = transformer.fit_transform(car_sales_missing)
transformed_X
```

Out[202…

```
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 [203…
```
car_sales_missing = pd.read_csv("/home/user/Downloads/car-sales-extended-
car_sales_missing.head()
```

Out[203…

	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

In [204…
```
car_sales_missing.isna().sum()
```

Out[204…
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [205…
```
car_sales_missing.dropna(subset = ['Price'], inplace = True)
car_sales_missing.isna().sum()
```

Out[205…
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
Price             0
dtype: int64
```

In [206…
```
# Split into X & y
X = car_sales_missing.drop("Price",axis = 1)
y = car_sales_missing["Price"]
```

In [208…
```
X.isna().sum()
```

Out[208…
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
dtype: int64
```

In [209…
```
# Fill missing values with Scikit-Learn
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# Fill categorical values with 'missing'  & numerical values with mean
cat_imputer = SimpleImputer(strategy ='constant', fill_value = 'missing')
door_imputer = SimpleImputer (strategy ='constant', fill_value = 4)
num_imputer = SimpleImputer(strategy = 'mean')

# Define columns
cat_features = ['Make', "Colour"]
door_features = ['Doors']
num_features = ['Odometer (KM)']
```

```
# Create an imputer (something that fills missing data)
imputer = ColumnTransformer([("cat_imputer", cat_imputer, cat_features),

#Transform the data
filled_X =  imputer.fit_transform(X)
filled_X
```

Out[209…

```
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 [210…

```
car_sales_filled = pd.DataFrame(filled_X, columns = ["Make", "Colour", "D
car_sales_filled.head()
```

Out[210…

	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

In [211…

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

Out[211…

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

In [212…

```
# Let's try to convert our data to 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)
transformed_X = transformer.fit_transform(car_sales_filled)
transformed_X
```

Out[212…

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 3800 stored elements and shape (950, 15)>
```

In [213…

```
# Now we've got our data as numbers and filled (no missing values)
# Let's fit a model
np.random.seed(42)
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

X_train,X_test, y_train, y_test = train_test_split(transformed_X, y, test

model = RandomForestRegressor(n_estimators= 100)
```

```
model.fit (X_train,y_train)
model.score(X_test, y_test)
```

Out[213…
```
0.21990196728583944
```

In [214…
```
len(car_sales_filled),len(car_sales)
```

Out[214…
```
(950, 1000)
```

car_sales_filled has lower value than previous (car_sales) because it owns less samples. We had removed some labels. Note: The 50 less values in the transformed data is because we dropped the rows (50 total) with missing values in the Price column.

2. Choosing the right estimator/algorith for your problem

Some things to note:

- Sklearn refers to machine learning models, algorithms as estimators.
- Classification problem - predicting a category (heart disease or not) *Sometimes you'll see `clf` (short for classifier) used as a classification estimator
- Regression problem - predicting a number (selling price of a car)

If you're working on a machine learning problem and looking to use Sklearn and not sure what model you should use, refer to the sklearn machine learning map: https://scikit-learn.org/stable/machine_learning_map.html

In [215…
```
what_iam_covering
```

Out[215…
```
['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 dat
a',
 '4. Evaluating a model',
 '5. Improve a model',
 '6. Save and load a trained model',
 '7. Putting it all together!']
```

2.1 Picking a machine learning model for a regression problem

Let's use the California dataset - https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_h

In [216…
```
# Get California Housing dataset
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
housing
```

Out[216…

```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,    2.
55555556,
           37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, ...,    2.1098418
3,
           37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, ...,    2.8022598
9,
           37.85      , -122.24      ],
        ...,
        [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351
,
           39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, ...,    2.1232091
7,
           39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, ...,    2.6169811
3,
           39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing dataset
\n--------------------------\n\n**Data Set Characteristics:**\n\n:Number
of Instances: 20640\n\n:Number of Attributes: 8 numeric, predictive attr
ibutes and the target\n\n:Attribute Information:\n    - MedInc        me
dian income in block group\n    - HouseAge      median house age in bloc
k group\n    - AveRooms      average number of rooms per household\n    
- AveBedrms     average number of bedrooms per household\n    - Populati
on    block group population\n    - AveOccup      average number of hous
ehold members\n    - Latitude      block group latitude\n    - Longitude
block group longitude\n\n:Missing Attribute Values: None\n\nThis dataset
was obtained from the StatLib repository.\nhttps://www.dcc.fc.up.pt/~lto
rgo/Regression/cal_housing.html\n\nThe target variable is the median hou
se value for California districts,\nexpressed in hundreds of thousands o
f dollars ($100,000).\n\nThis dataset was derived from the 1990 U.S. cen
sus, using one row per census\nblock group. A block group is the smalles
t geographical unit for which the U.S.\nCensus Bureau publishes sample d
ata (a block group typically has a population\nof 600 to 3,000 peopl
e).\n\nA household is a group of people residing within a home. Since th
e average\nnumber of rooms and bedrooms in this dataset are provided per
household, these\ncolumns may take surprisingly large values for block g
roups with few households\nand many empty houses, such as vacation resor
ts.\n\nIt can be downloaded/loaded using the\n:func:`sklearn.datasets.fe
tch_california_housing` function.\n\n.. rubric:: References\n\n- Pace,
R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,\n  Statistic
s and Probability Letters, 33 (1997) 291-297\n'}
```

In [217…

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

Out[217…

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

In [218…

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

Out[218…

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

In [ ]:

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

In [ ]:

```
# Import algorithm/estimator
from sklearn.linear_model import Ridge

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

# Create the data
X = housing_df.drop("target", axis = 1)
y = housing_df["target"] # median house price in $ 100,000s

print(X.shape)
print(y.shape)

# Split into train and test sets
X_train, X_test, y_train,y_test = train_test_split(X,y, test_size = 0.2)

# Instantiate and fit the model (on the training set)
model = Ridge()
model.fit(X_train,y_train)

# Check the score of the mode
model.score(X_test,y_test)
```

In [220…

```
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

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

```
y = housing_df["target"]

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Model
model = HistGradientBoostingRegressor(max_iter=100)
model.fit(X_train, y_train)

# Evaluation
y_pred = model.predict(X_test)
score = model.score(X_test, y_test) # R²
score
```

Out[220… 0.8342922792113371

In [221…
```
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Setup
X = housing_df.drop("target", axis=1)
y = housing_df["target"]

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Model
model = HistGradientBoostingRegressor(max_iter=100)
model.fit(X_train, y_train)

# Evaluation
y_pred = model.predict(X_test)
score = model.score(X_test, y_test) # R²
score
```

Out[221… 0.8343975028147513

What if `Ridge` did not work or the score didn't fit our needs?

Well, we could always try a different model...

How about we try an ensemble model (an ensemble is combination of smaller models to try make better predictions than just a single model?)

Sklearn's ensemble models can be found here: https://scikit-learn.org/stable/modules/ensemble.html

In [222…
```
# Import the RandomForestRegressor model class from the ensemble module
from sklearn.ensemble import RandomForestRegressor

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

# Create the data
X = housing_df.drop("target", axis = 1)
y = housing_df["target"]
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size = 0.2

#Create random forest model
model = RandomForestRegressor()
model.fit (X_train,y_train)

#Check the score of the model (on the test set)
model.score(X_test, y_test)
```

Out[222…

```
0.8066196804802649
```

2.2 Picking a machine learning for a classification problem

Let's go to the map... https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

In [225…

```
heart_disease = pd.read_csv("/home/user/Downloads/heart-disease.csv")
heart_disease.head()
```

Out[225…

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

In [224…

```
len(heart_disease)
```

Out[224…

```
303
```

Consulting the map and it says to try `LinearSVC`

In [226…

```
# Import the LinearSVC estimator class
from sklearn.svm import LinearSVC

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

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

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size= 0.2)

# Instantiate LinearSVC
clf = LinearSVC(max_iter = 1000)
clf.fit(X_train,y_train)
```

```
#Evaluate the LinearSVC
clf.score(X_test,y_test)
```

Out[226… 0.8688524590163934

In [227…
```
heart_disease["target"].value_counts
```

Out[227…
```
<bound method IndexOpsMixin.value_counts of 0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64>
```

In [228…
```
## Import the RandomForestClassifier estimator class
from sklearn.ensemble import RandomForestClassifier
# Setup random seed
np.random.seed(42)

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

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size= 0.2)

# Instantiate Random Forest Classifier
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train,y_train)

#Evaluate the Random Forest Classifier
clf.score(X_test,y_test)
```

Out[228… 0.8524590163934426

Tid bit:

1. If you have structured data (DataFrame table), use ensemble methods
2. If you have unstructured data (images, audio, text), use deep learning or transfer learning.

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

3.1 Fitting the model to the data

Different names for:

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

In [229…

```
# Import the RandomForestClassifier estimator class
from sklearn.ensemble import RandomForestClassifier
# Setup random seed
np.random.seed(42)

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

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size= 0.2)

# Instantiate Random Forest Classifier
clf = RandomForestClassifier(n_estimators = 100)

# Fit the model to the data (training the machine learning model)
clf.fit(X_train,y_train)

#Evaluate the Random Forest Classifier (use the patterns the model has us
clf.score(X_test,y_test)
```

Out[229…

```
0.8524590163934426
```

In [230…

```
X.head()
```

Out[230…

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

In [231…

```
y.tail()
```

Out[231…

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

3.2 Make predictions using a machine learning model

2 ways to make predictions:

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

In [ ]:

```
# Use a trained model to make predictions
#clf.predict(np.array([1, 7,8, 3,4]))
```

In [232…
```
clf.predict(X_test)
```

Out[232…
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0])
```

In [233…
```
np.array(y_test)
```

Out[233…
```
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])
```

In [234…
```
# Compare predictions to truth labels to evaluate the model
y_preds = clf.predict(X_test)
np.mean(y_preds == y_test)
```

Out[234…
```
np.float64(0.8524590163934426)
```

In [235…
```
clf.score(X_test,y_test)
```

Out[235…
```
0.8524590163934426
```

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

Make predictions with `predict_proba()`

In [236…
```
# predict_proba () returns probabilities of a classification label
clf.predict_proba(X_test[:5])
```

Out[236…
```
array([[0.89, 0.11],
       [0.49, 0.51],
       [0.43, 0.57],
       [0.84, 0.16],
       [0.18, 0.82]])
```

In [237…
```
# Let's predict () on the same data...
clf.predict(X_test[:5])
```

Out[237…
```
array([0, 1, 1, 0, 1])
```

In [238…
```
X_test[:5]
```

Out[238…

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

`predict()` can be used for regression models

In [239…
```
housing_df.head()
```

Out[239…

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

In [250…
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

# Create the data

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

# Split into training and test sets
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)

# Create model instance
model = RandomForestRegressor()

#Fit the model to the data
model.fit(X_train, y_train)

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

In [241…
```
y_preds[:10]
```

Out[241…
```
array([0.49384  , 0.75494  , 4.9285964, 2.54029  , 2.33176  , 1.6549701,
       2.34323  , 1.66182  , 2.47489  , 4.8344779])
```

In [242…
```
np.array(y_test[:10])
```

Out[242…
```
array([0.477  , 0.458  , 5.00001, 2.186  , 2.78   , 1.587  , 1.982  ,
       1.575  , 3.4    , 4.466  ])
```

In [243…
```
# Compare the predictions to the truth
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,y_preds)
```

Out[243…
```
0.3265721842781009
```

In [252…
```
len(y_preds), len(y_test)
```

Out[252…
```
(4128, 4128)
```

In [246…
```
housing_df["target"]
```

Out[246…

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

4. Evaluating a machine learning model

Three ways to evaluate Scickit-Learn models/estimators:

1. Estimator's built-in `score()` method
2. The `scoring` parameter
3. Problem-specific metric functions

You can read more these here: https://scikit-learn.org/stable/modules/model_evaluation.html

4.1 Evaluating a model with the `score` method

In [248…

```
from sklearn.ensemble import RandomForestClassifier
# Setup random seed
np.random.seed(42)

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

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size= 0.2)

# Instantiate Random Forest Classifier
clf = RandomForestClassifier(n_estimators = 500)

# Fit the model to the data (training the machine learning model)
clf.fit(X_train,y_train)
```

Out[248…

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=500)
```

In [249…

```
# The highest value for the.score() method is 1.0, the lowest is 0.0
clf.score(X_train, y_train)
```

Out[249… 1.0

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

Let's use the `score()` on our regression problem...

In [254...
```
from sklearn.ensemble import RandomForestClassifier
# Setup random seed
np.random.seed(42)

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

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size= 0.2)

# Instantiate Random Forest Classifier
model = RandomForestClassifier(n_estimators= 100)

# Fit the model to the data (training the machine learning model)
model.fit(X_train,y_train)
```

Out[254...
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [255...
```
# The default score () evaluation metric is r_squared for regression algo
# Highest = 1.0/ lowest = 0.0
model.score(X_test,y_test)
```

Out[255...
```
0.8524590163934426
```

In [256...
```
model.score(X_test,y_test)
```

Out[256...
```
0.8524590163934426
```

In [257...
```
housing_df.head()
```

Out[257...

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

In [258...
```
y_test.mean()
```

Out[258...
```
np.float64(0.5245901639344263)
```

4.2 Evaluating a model using the scoring parameter

In [259…

```
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier
# Setup random seed
np.random.seed(42)

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

# Split the data
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size= 0.2)

# Instantiate Random Forest Classifier
clf = RandomForestClassifier(n_estimators = 100)

# Fit the model to the data (training the machine learning model)
clf.fit(X_train,y_train)
```

Out[259…

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [260…

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

Out[260…

```
0.8524590163934426
```

In [261…

```
cross_val_score(clf,X,y,cv=5)
```

Out[261…

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

In [262…

```
np.random.seed(42)

# Single_training and test split score
clf_single_score = clf.score(X_test,y_test)

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

# Compare the two
clf_single_score, clf_cross_val_score
```

Out[262…

```
(0.8524590163934426, np.float64(0.8248087431693989))
```

In [ ]:

```
# Default scoring parameter of classifier = mean accuracy
clf.score()
```

In [263…

```
# Scoring parameter set to None by default
cross_val_score (clf,X,y, cv = 5, scoring = None)
```

Out[263…

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

4.2.1 Classification model evaluation metrics

1. Accurancy

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

In [264…
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

np. random.seed(42)

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

clf = RandomForestClassifier(n_estimators = 100)
cross_val_score = cross_val_score(clf,X,y,cv= 5)
cross_val_score
```

Out[264…
```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

In [265…
```
np.mean(cross_val_score)
```

Out[265…
```
np.float64(0.8248087431693989)
```

In [266…
```
print(f"Heart Disease Classifier Cross-Validated Accurancy : {np.mean(cro
```

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

Area under the receiver operating characterisctic curve (AUC/ROC)

- Area under curve(AUC)
- ROC curve

ROC curve are a comparison of a model's true positive rate (tpr) versus a models 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 predict 0 when truth is 0
- False negative = model predicts 0 when truth is 1

In [269…
```
# Create X_test...etc
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
```

In [270…
```
from sklearn.metrics import roc_curve

# Fit the classifier
clf.fit(X_train, y_train)

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

y_probs[:10]
```

Out[270…
```
array([[0.16, 0.84],
       [0.09, 0.91],
       [0.26, 0.74],
       [0.71, 0.29],
       [0.28, 0.72],
       [0.08, 0.92],
       [0.97, 0.03],
       [0.05, 0.95],
       [0.14, 0.86],
       [0.  , 1.  ]])
```

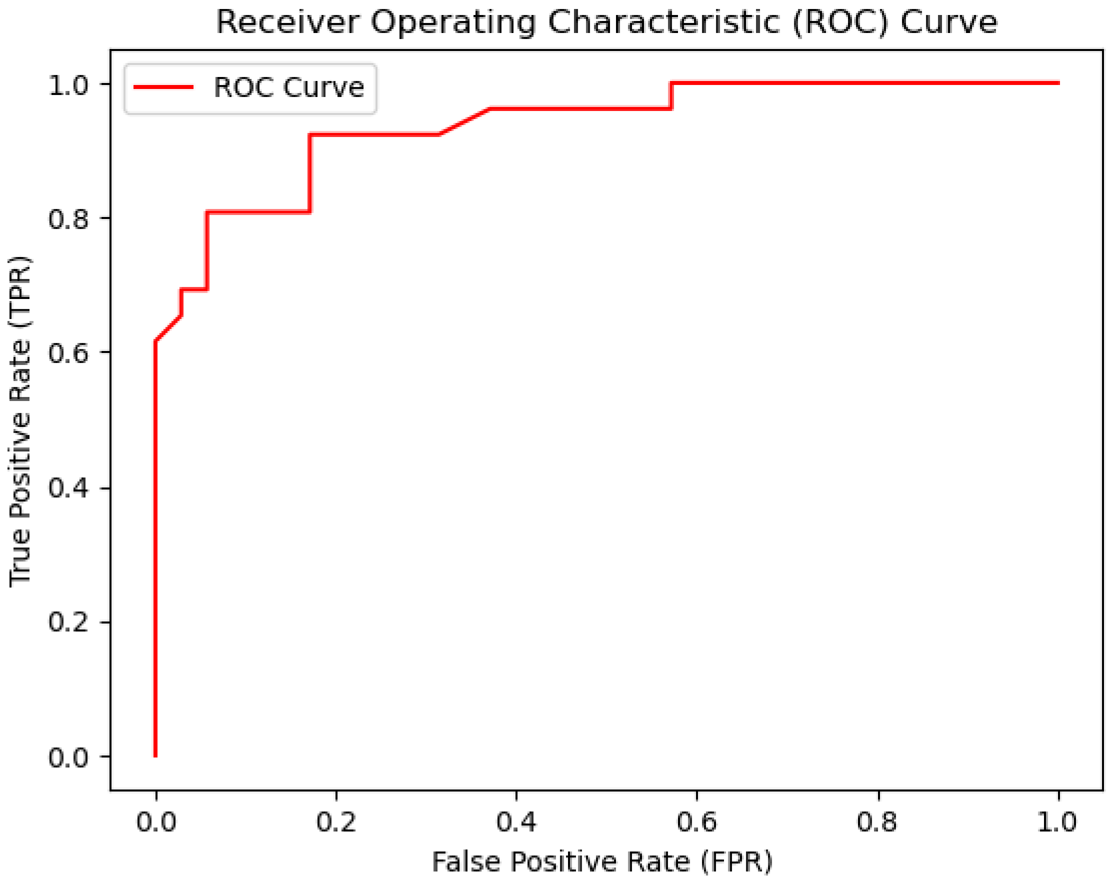
In [271…
```
y_probs_positive = y_probs[:,1]
y_probs_positive[:10]
```

Out[271…
```
array([0.84, 0.91, 0.74, 0.29, 0.72, 0.92, 0.03, 0.95, 0.86, 1.  ])
```

In [272…
```
# Calculate fpr, tpr and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)

# Check the false positive rates
fpr
```

Out[272…
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.02857143,
       0.02857143, 0.05714286, 0.05714286, 0.11428571, 0.17142857,
       0.17142857, 0.31428571, 0.37142857, 0.42857143, 0.48571429,
       0.57142857, 0.57142857, 0.62857143, 0.74285714, 0.82857143,
       1.        ])
```

In [273…
```
# Create a function for plotting ROC curves

import matplotlib.pyplot as plt

def plot_roc_curve(fpr,tpr):
    """
    Plots a ROC curve given the false positive rate (fpr) and true positi

# Plot roc curve

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

# Customize the plot
    plt.xlabel("False Positive Rate (FPR) ")
    plt.ylabel("True Positive Rate (TPR)")
    plt.title("Receiver Operating Characteristic (ROC) Curve")
    plt.legend()
    plt.show()


plot_roc_curve(fpr,tpr)
```



In [274…

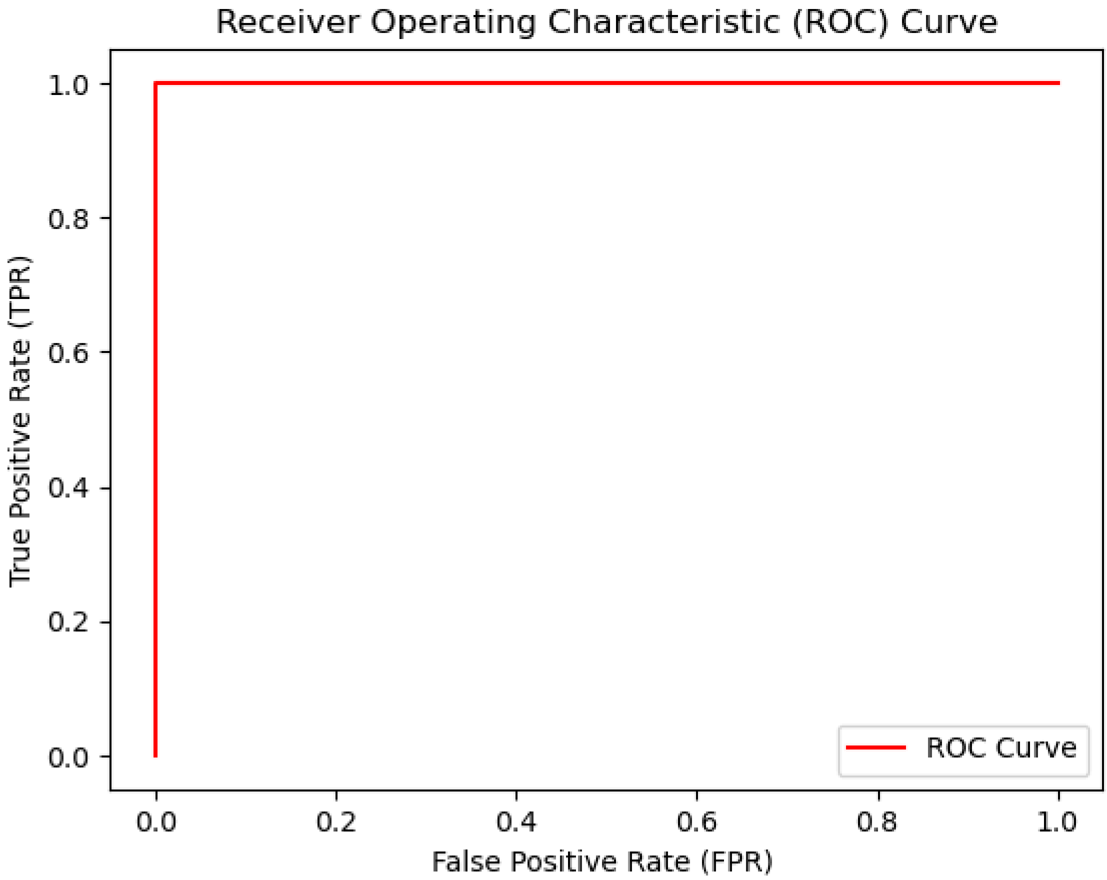
```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_probs_positive)
```

Out[274…

```
np.float64(0.9368131868131868)
```

In [275…

```
#Plot perfect ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test,y_test)
plot_roc_curve (fpr,tpr)
```



In [276…
```
# Perfect AUC score
roc_auc_score(y_test,y_test)
```

Out[276…
```
np.float64(1.0)
```

Confusion Matrix

The next way to evaluate a classification model is by using a confusion 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 ideia of where the model is getting confused.

In [277…
```
import sys
!conda install --yes --prefix {sys.prefix} seaborn
```

```
Channels:
 - defaults
Platform: linux-64
doneecting package metadata (repodata.json): -
doneing environment: -

# All requested packages already installed.
```

In [280…
```
from sklearn.metrics import confusion_matrix


y_preds = clf.predict(X_test)
```

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

Out[280…

```
array([[27,  8],
       [ 2, 24]])
```

In [279…

```
# Visualize confusion matrix wit pd.crosstab()
pd.crosstab(y_test,y_preds,rownames = ["Actual Label"], colnames =["Predi
```

Out[279…

Predicted Labels	0	1
Actual Label		
0	27	8
1	2	24

In [ ]:

```
22+3+5+31
```

In [281…

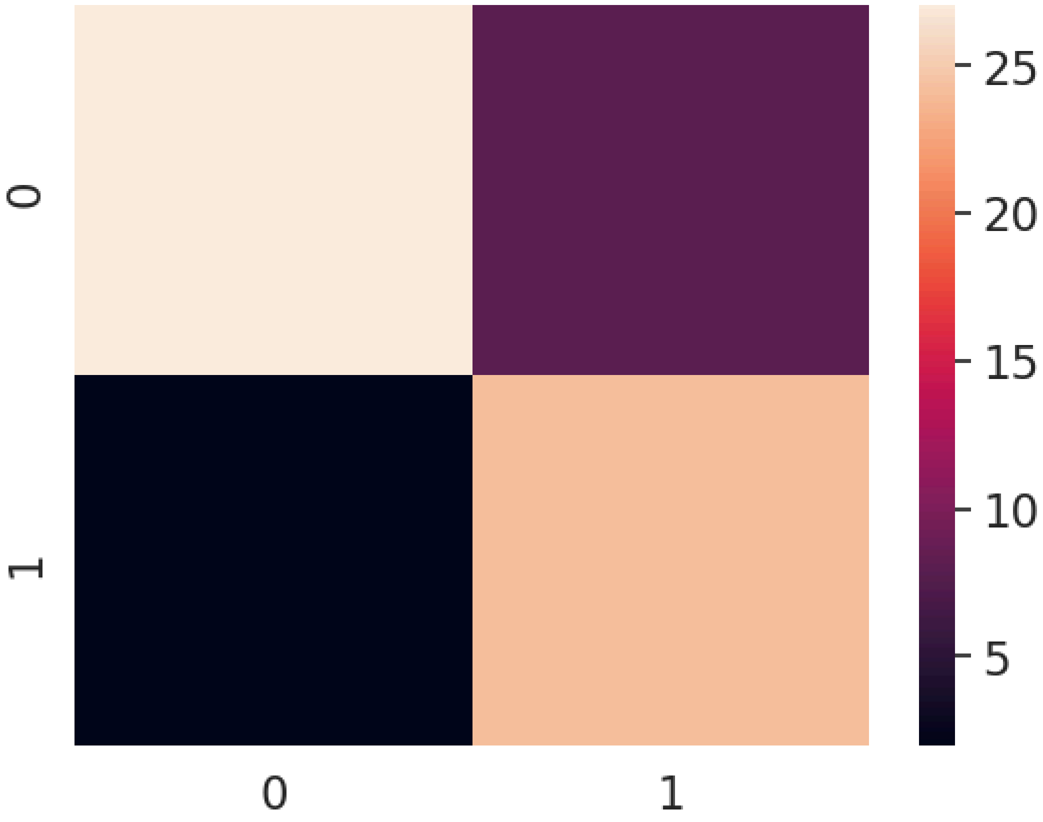
```
len(X_test)
```

Out[281…

```
61
```

In [282…

```
# Make our confusion matrix more visual with Seaborn's heatmap
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Visualization Set
sns.set(font_scale=1.5)

# Creation of confusion matrix
conf_mat = confusion_matrix(y_test, y_preds)

# Heatmap generation
sns.heatmap(conf_mat)

# Show the graph
plt.show()
```



Creating a confusion matrix using Scikit-Learn

To use new methods of creating a confusion matrix with Scikit-Learn you will need sklearn version 1.0+

In [283…
```
import sklearn
sklearn.__version__
```

Out[283… '1.6.1'

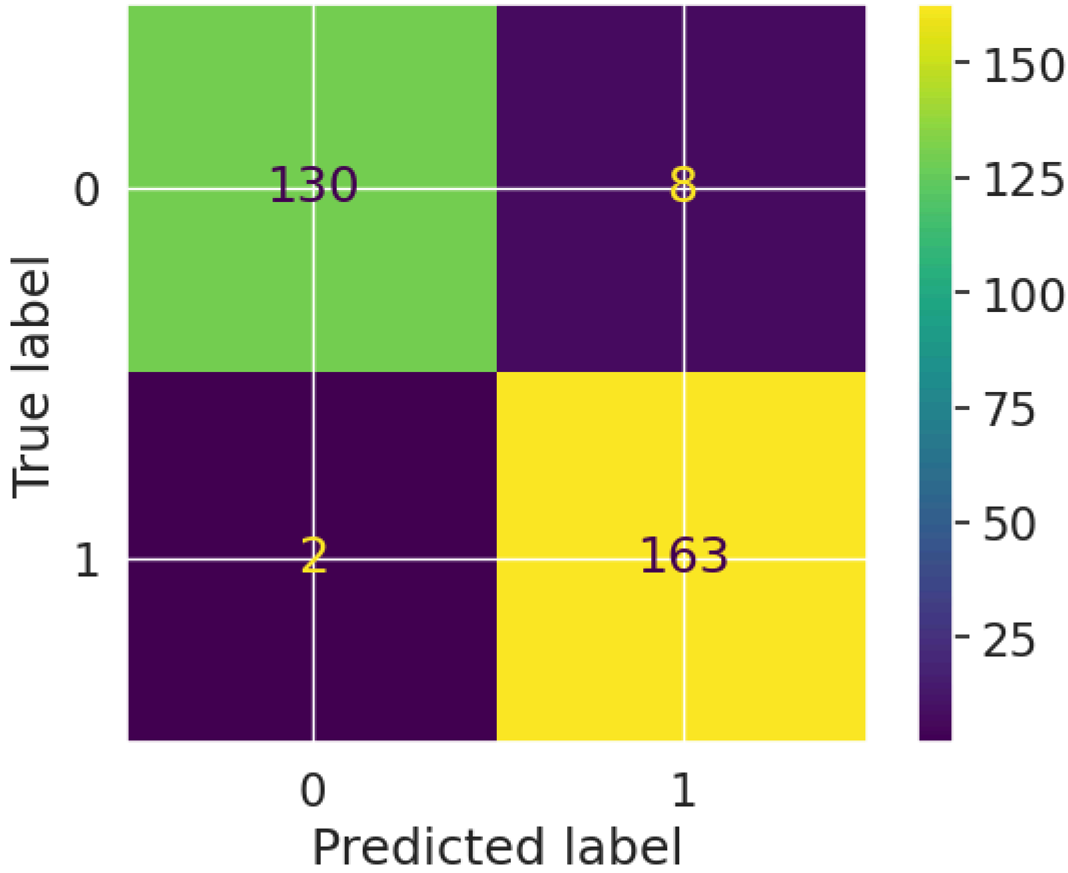
In [284…
```
clf
```

Out[284…

▾ RandomForestClassifier

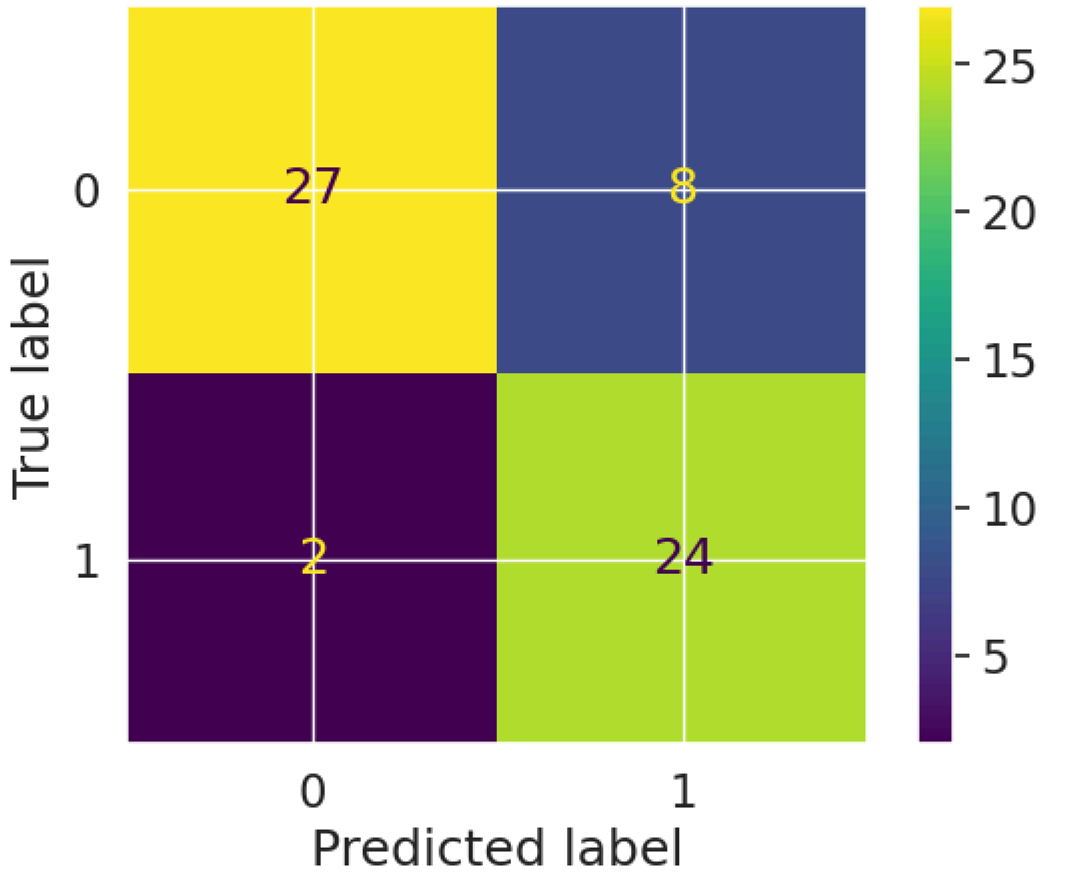
```
RandomForestClassifier()
```

In [285…
```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(estimator = clf, X=X, y=y)
plt.show();
```



In [286…

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



In [287…
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_preds))
```
```
              precision    recall  f1-score   support

           0       0.93      0.77      0.84        35
           1       0.75      0.92      0.83        26

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

In [288…
```
# Where precision and recall become valuable
disease_true = np.zeros (1000)
disease_true[0] = 1 # only positive case

disease_preds = np.zeros(1000) # model predicts every case as 0

pd.DataFrame(classification_report(disease_true, disease_preds, output_di
```

Out[288…

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.9990	0.0	0.999	0.49950	0.998001
recall	1.0000	0.0	0.999	0.50000	0.999000
f1-score	0.9995	0.0	0.999	0.49975	0.998500
support	999.0000	1.0	0.999	1000.00000	1000.000000

To summarize classification 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 importante when classes are imbalanced.
- If false positive predictions are worse than false negatives, aim for higher precision.
- If false negative predictions are worse than false positives, aim for higher recall.

4.2.2. Regression model evaluation metrics

Model evaluation metrics documentation - https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

The ones I'm going cover are:

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

In [302…
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)
```

```
X = housing_df.drop("target",axis= 1)
y = housing_df["target"]

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)

model = RandomForestRegressor(n_estimators = 100)
model.fit(X_train,y_train)
```

Out[302…

```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [299…

```
RandomForestRegressor()
model.score(X_test,y_test)
```

Out[299…

```
0.8066196804802649
```

In [301…

```
housing_df.head()
```

Out[301…

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

In [296…

```
y_test.mean()
```

Out[296…

```
np.float64(2.0550030959302323)
```

In [303…

```
y_test
```

Out[303…

```
20046    0.47700
3024     0.45800
15663    5.00001
20484    2.18600
9814     2.78000
          ...
15362    2.63300
16623    2.66800
18086    5.00001
2144     0.72300
3665     1.51500
Name: target, Length: 4128, dtype: float64
```

In [304…

```
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 [306…

```
y_test_mean[:10]
```

Out[306…
```
array([2.0550031, 2.0550031, 2.0550031, 2.0550031, 2.0550031, 2.0550031,
       2.0550031, 2.0550031, 2.0550031, 2.0550031])
```

r2_score(y_true = y_test, y_pred = y_test) r2_score

Mean absolute error (MAE)

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

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

In [305…
```
from sklearn.metrics import mean_absolute_error

y_preds = model.predict(X_test)
mae = mean_absolute_error(y_test,y_preds)
mae
```

Out[305…
```
0.3265721842781009
```

In [307…
```
y_preds = model.predict(X_test)
y_proba = model.predict(X_test)
#y_proba = model.predict_proba(X_test)[:, 1]  # Probability of class 1
df = pd.DataFrame({
    "actual values": y_test,
    "predicted probabilities": y_preds,
    "differences": y_proba - y_test
})
mae = mean_absolute_error(y_test, y_proba)
mae
df.head(10)
```

Out[307…

	actual values	predicted probabilities	differences
20046	0.47700	0.493840	0.016840
3024	0.45800	0.754940	0.296940
15663	5.00001	4.928596	-0.071414
20484	2.18600	2.540290	0.354290
9814	2.78000	2.331760	-0.448240
13311	1.58700	1.654970	0.067970
7113	1.98200	2.343230	0.361230
7668	1.57500	1.661820	0.086820
18246	3.40000	2.474890	-0.925110
5723	4.46600	4.834478	0.368478

In [308…
```
#MAE using formulas and differences
np.abs (df ["differences"]).mean()
```

Out[308…
```
np.float64(0.3265721842781009)
```

Mean squared error (MSE)

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

In [309…
```
# 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[309… 0.2534073069137548

In [310…
```
df["squared_differences"] = np.square(df["differences"])
df.head()
```

Out[310…

	actual values	predicted probabilities	differences	squared_differences
20046	0.47700	0.493840	0.016840	0.000284
3024	0.45800	0.754940	0.296940	0.088173
15663	5.00001	4.928596	-0.071414	0.005100
20484	2.18600	2.540290	0.354290	0.125521
9814	2.78000	2.331760	-0.448240	0.200919

In [311…
```
# Calculate MSE by hand
squared = np.square(df["differences"])
squared.mean()
```

Out[311… np.float64(0.2534073069137548)

In [312…
```
df_large_error = df.copy()
df_large_error.at[df_large_error.index[0], "squared_differences"] = 16
```

In [313…
```
df_large_error.head()
```

Out[313…

	actual values	predicted probabilities	differences	squared_differences
20046	0.47700	0.493840	0.016840	16.000000
3024	0.45800	0.754940	0.296940	0.088173
15663	5.00001	4.928596	-0.071414	0.005100
20484	2.18600	2.540290	0.354290	0.125521
9814	2.78000	2.331760	-0.448240	0.200919

In [314…
```
# Calculate MSE with large error
df_large_error["squared_differences"].mean()
```

Out[314… np.float64(0.25728320720794084)

In [316…
```
df_large_error.iloc[1:100] = 20
df_large_error.head()
```

Out[316…

	actual values	predicted probabilities	differences	squared_differences
20046	0.477	0.49384	0.01684	16.0
3024	20.000	20.00000	20.00000	20.0
15663	20.000	20.00000	20.00000	20.0
20484	20.000	20.00000	20.00000	20.0
9814	20.000	20.00000	20.00000	20.0

4.2.3. Finally using the scoring parameter

In [317…

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

np.random.seed(42)

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

clf = RandomForestClassifier(n_estimators = 100)
```

In [321…

```
np.random.seed(42)

# Cross-validation accuracy
cv_acc = cross_val_score(clf,X,y,cv = 5, scoring = None) # if  scoring= N
cv_acc
```

Out[321…

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

In [320…

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

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

In [322…

```
np.random.seed(42)

cv_acc = cross_val_score(clf,X,y,cv = 5, scoring = "accuracy")
cv_acc
```

Out[322…

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

In [323…

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

Out[323…

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

In [324…

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

Out[324…

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

In [325…

```
# Cross-validated precision
print(f"The cross-validated precision is: {np.mean(cv_recall)}")
```

```
The cross-validated precision is: 0.8545454545454545
```

Let's see the `scoring` parameter being using for a regression problem.

In [341…

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

np.random.seed(42)

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

model = RandomForestRegressor(n_estimators = 100)
```

In [ ]:

```
np.random.seed(42)
cv_r2 = cross_val_score(model, X,y, cv = 3, scoring = None)
np.mean(cv_r2)
```

In [ ]:

```
# Mean squared error
cv_mse = cross_val_score(model, X, y, cv = 3, scoring = "neg_mean_squared
np.mean(cv_mse)
```

In [ ]:

```
# Mean absolute error
cv_mae = cross_val_score(model, X,y, cv = 3, scoring = "neg_mean_absolute
np.mean(cv_mae)
```

4.3 Using different evaluation metrics as Scikit-Learn functions

The 3rd way to evaluate scikit-learn machine learning models/estimators is to using the `sklearn.metrics` module - https://scikit-learn.org/stable/modules/model_evaluation.html#module-sklearn.metrics

In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score,recall_score,
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

np.random.seed(42)

# Create X & y
X = heart_disease.drop("target", axis = 1)
y = heart_disease["target"]

# Split data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)

# Create model
clf = RandomForestClassifier()

# Fit
clf.fit(X_train,y_train)
```

```python
# Make predictions
y_preds = clf.predict(X_test)

# Evaluate model using evaluation functions
print("Classifier metrics on the test set")
print(f"Accuracy: {accuracy_score(y_test, y_preds)*100:.2f}%")
print(f"Precision: {precision_score(y_test, y_preds)*100:.2f}%")
print(f"Recall: {recall_score(y_test, y_preds)*100:.2f}%")
print(f"F1: {f1_score(y_test, y_preds)*100:.2f}%")
```

In [ ]:

```python
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_e
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

np.random.seed(42)

#Create X & y
X = housing_df.drop("target",axis =1)
y = housing_df["target"]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)

# Create model
model = RandomForestRegressor()

# Fit model
model.fit(X_train,y_train)

# Make predictions
y_preds = model.predict(X_test)

# Evaluate model using evaluation functions
print("Regression metrics on the test set")
print(f"R2 score: {r2_score(y_test,y_preds)}")
print(f"MAE : {mean_absolute_error(y_test,y_preds)}")
print(f"MSE: {mean_squared_error(y_test,y_preds)}")
```

In [ ]:

```python
what_iam_covering
```

5. Improving a model

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

From a data perspective:

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

From a model perspective:

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

Hyperparameters vs Parameters

*Parameters = model find these pattterns in data *Hyperparameters = settings on a model you can adjust to (potentially) improve its abiltiy to find patterns

Three ways to adjust hyperparameteres:

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

Let's make 3 sets, training, validation and test.

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

clf = RandomForestClassifier()
clf.get_params()
```

We're going to try and adjust:

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

In [66]:
```
def evaluate_preds(y_true, y_preds):
    """
    Performs evaluation comparison on y_true labels vs. y_pred labels
    on a classification.
    """
    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(prec
    print(f"Acc: {accuracy*100:.2f}%")
    print(f"Precision: {precision*100:.2f}%")
    print(f"Recall: {recall*100:.2f}%")
    print(f"F1 score: {f1*100:.2f}%")

    return metric_dict
```

In [67]:
```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

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

# Split into X & y
X = heart_disease_shuffled.drop("target", axis= 1)
y= heart_disease_shuffled["target"]

# Split the data into train validation & test sets
train_split = round(0.7 * len(heart_disease_shuffled)) # 70% of data
valid_split = round(train_split + 0.15 * len(heart_disease_shuffled)) # 1
```

```python
X_train, y_train = X[:train_split], y[:train_split]
X_valid, y_valid = X[train_split:valid_split],y[train_split:valid_split]
X_test, y_test = X[valid_split:],y[:valid_split]

len(X_train), len(X_valid), len(X_test)


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

# Make baseline predictions
y_preds = clf.predict(X_valid)

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

```
Acc: 82.22%
Precision: 81.48%
Recall: 88.00%
F1 score: 84.62%
```

Out[67]:
```
{'accuracy': 0.82, 'precision': 0.81, 'recalll': 0.88, 'f1': 0.85}
```

In [68]:
```python
np.random.seed(42)

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

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

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

```
Acc: 82.22%
Precision: 84.00%
Recall: 84.00%
F1 score: 84.00%
```

5.2 Hyperparameter tuning with RandomizedSearchCV

In [69]:
```python
from sklearn.model_selection import RandomizedSearchCV

grid = {"n_estimators": [10,100,200,500,1000,1200], "max_depth": [None, 5

np.random.seed(42)

# Split into X & y
X = heart_disease_shuffled.drop("target", axis =1)
y= heart_disease_shuffled["target"]

# Split into train and test sets
X_train,X_test, y_train, y_test = train_test_split( X,y, test_size = 0.2)

# Instantiate RandomForestClassifier
clf = RandomForestClassifier(n_jobs = 1)

# Setup RandomizedSearchCV
```

```
rs_clf = RandomizedSearchCV(estimator = clf,
                            param_distributions = grid, n_iter = 10, # numbe
                            cv = 5,
                            verbose = 2)

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

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=1200; total time=   4.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=1200; total time=   4.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=1200; total time=   4.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=1200; total time=   4.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=1200; total time=   4.1s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.1s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
```

```
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   1.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.9s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.9s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.5s
```

```
/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklearn/model_
selection/_validation.py:528: FitFailedWarning:
20 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/utils/_param_validation.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf),
a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got
'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklearn/model_
selection/_search.py:1108: UserWarning: One or more of the test scores are
non-finite: [0.82244898        nan 0.80620748        nan 0.80595238
nan
 0.81428571 0.83886054        nan 0.81428571]
  warnings.warn(
```

In [71]:
```
rs_clf.best_params_
```

Out[71]:
```
{'n_estimators': 200,
 'min_samples_split': 6,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': None}
```

In [72]:
```
# Make predictions with the best hyperparameters
rs_y_preds = rs_clf.predict(X_test)


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

```
Acc: 81.97%
Precision: 77.42%
Recall: 85.71%
F1 score: 81.36%
```

5.2 Hyperparameter tuning with RandomizedSearchCV

In [79]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier


grid = {"n_estimators": [10,100,200, 500, 1000, 2000],
        "max_depth": [None, 5, 10,20,30],
"max_features": ["auto","sqrt"], "min_samples_split": [2,4,6], "min_sampl

np.random.seed(42)

# Split into X & y
X = heart_disease_shuffled.drop("target", axis = 1)
y = heart_disease_shuffled["target"]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)

#Instantiate RandomForestClassifier
clf = RandomForestClassifier(n_jobs = 1)

#Setup RandomizedSearchCV
rs_clf = RandomizedSearchCV(estimator = clf,
                            param_distributions = grid,
                            n_iter = 10, # number of models to try
                            cv = 5,
                            verbose = 2)
# Fit the RandomizedSearchCV version of clf
rs_clf.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=2000; total time=   7.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=2000; total time=   8.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=2000; total time=   7.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=2000; total time=   7.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_s
plit=6, n_estimators=2000; total time=   6.9s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=100; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=200; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_s
plit=4, n_estimators=10; total time=   0.1s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_
```

```
split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   2.0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=4, n_estimators=200; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=4, n_estimators=1000; total time=   3.5s
```

```
/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklearn/model_
selection/_validation.py:528: FitFailedWarning:
20 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/utils/_param_validation.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf),
a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got
'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklearn/model_
selection/_search.py:1108: UserWarning: One or more of the test scores are
non-finite: [0.81420068        nan 0.81828231        nan 0.78103741
nan
 0.81420068 0.82670068        nan 0.82253401]
  warnings.warn(
```

In [81]:

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

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

```
Acc: 83.61%
Precision: 78.12%
Recall: 89.29%
F1 score: 83.33%
```

5.3. Hyperparameter tuning with GridSearchCV

In [82]:

```
grid
```

Out[82]:

```
{'n_estimators': [10, 100, 200, 500, 1000, 2000],
 'max_depth': [None, 5, 10, 20, 30],
 'max_features': ['auto', 'sqrt'],
 'min_samples_split': [2, 4, 6],
 'min_samples_leaf': [1, 2, 4]}
```

In [86]:
```
grid_2 = {'n_estimators': [ 100, 200, 500],
 'max_depth': [None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_split': [6],
 'min_samples_leaf': [1, 2]}
```

In [87]:
```
from sklearn.model_selection import GridSearchCV, train_test_split

np.random.seed(42)

# Split into X & y
X = heart_disease_shuffled.drop("target", axis = 1)
y = heart_disease_shuffled["target"]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)

#Instantiate RandomForestClassifier
clf = RandomForestClassifier(n_jobs = 1)

#Setup GriddSearchCV
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);
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
```

```
s_split=6, n_estimators=500; total time=   0.0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   2.0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   2.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=100; total time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=200; total time=   0.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.8s
```

```
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=6, n_estimators=500; total time=   1.9s
```

```
/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklearn/model_
selection/_validation.py:528: FitFailedWarning:
30 fits failed out of a total of 60.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting e
rror_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklear
n/utils/_param_validation.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf),
a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got
'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/home/user/miniconda3/envs/env/lib/python3.12/site-packages/sklearn/model_
selection/_search.py:1108: UserWarning: One or more of the test scores are
non-finite: [       nan        nan        nan        nan        nan
nan
 0.82270408 0.81811224 0.82244898 0.82253401 0.82236395 0.81011905]
  warnings.warn(
```

In [89]:
```python
gs_clf.best_params_
```

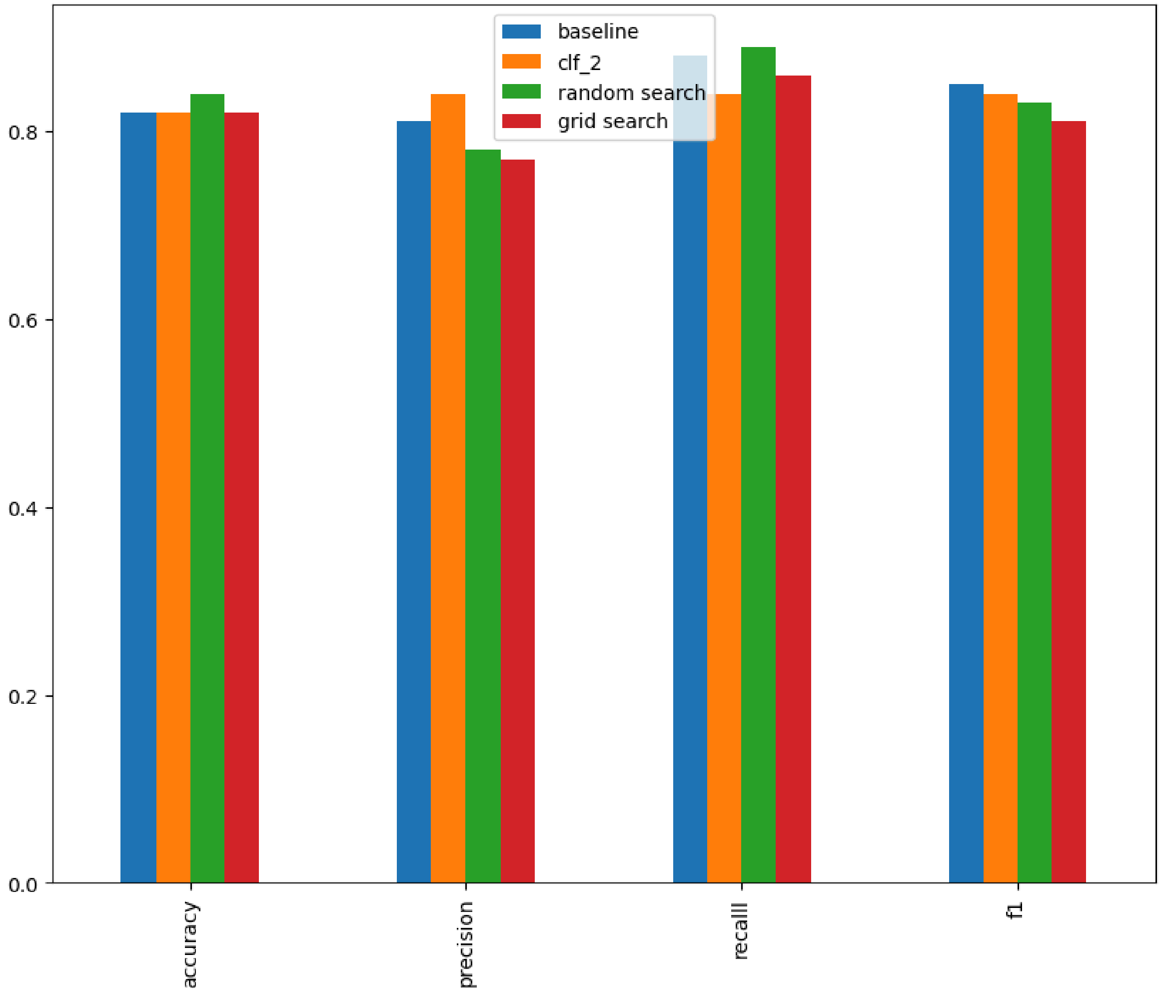
Out[89]:
```
{'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'n_estimators': 100}
```

In [91]:
```python
gs_y_preds = gs_clf.predict(X_test)

# evaluate the predictions
gs_metrics = evaluate_preds(y_test, gs_y_preds)
```

```
Acc: 81.97%
Precision: 77.42%
Recall: 85.71%
F1 score: 81.36%
```

Lets's compare our different models metrics.

In [99]:
```
compare_metrics = pd.DataFrame({"baseline" : baseline_metrics, "clf_2": c
compare_metrics.plot.bar(figsize = (10, 8));
plt.show();
```



In [100…
```
what_iam_covering
```

Out[100…
```
['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 dat
a',
 '4. Evaluating a model',
 '5. Improve a model',
 '6. Save and load a trained model',
 '7. Putting it all together!']
```

6. Saving and loading trained machine learning models

Two ways to save and load machine learning models:

1. With Pyhon's `pickle` module
2. With the `joblib` module

In [101…
```
import pickle
```

```
#Save as existing model to file
pickle.dump(gs_clf, open("gs_random_forest_model1_1.pkl","wb"))
```

In [102…

```
# Load a saved model
loaded_pickled_model = pickle.load(open("gs_random_forest_model1_1.pkl",
```

In [103…

```
# Make some predictions
pickle_y_preds = loaded_pickled_model.predict(X_test)
evaluate_preds(y_test, pickle_y_preds)
```

```
Acc: 81.97%
Precision: 77.42%
Recall: 85.71%
F1 score: 81.36%
```

Out[103…

```
{'accuracy': 0.82, 'precision': 0.77, 'recalll': 0.86, 'f1': 0.81}
```

Joblib

In [105…

```
from joblib import dump, load

# save model to file
dump(gs_clf, filename = "gs_random_forest_model1_1.joblib")
```

Out[105…

```
['gs_random_forest_model1_1.joblib']
```

In [109…

```
# Import a saved joblib model
loaded_joblib_model = load(filename = "gs_random_forest_model1_1.joblib")
```

In [110…

```
# Make and evaluate joblib predictions
joblib_y_preds = loaded_joblib_model.predict(X_test)
evaluate_preds(y_test, joblib_y_preds)
```

```
Acc: 81.97%
Precision: 77.42%
Recall: 85.71%
F1 score: 81.36%
```

Out[110…

```
{'accuracy': 0.82, 'precision': 0.77, 'recalll': 0.86, 'f1': 0.81}
```

In [111…

```
what_iam_covering
```

Out[111…

```
['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 dat
a',
 '4. Evaluating a model',
 '5. Improve a model',
 '6. Save and load a trained model',
 '7. Putting it all together!']
```

7 Putting it all together!

In [114…

```
data = pd.read_csv("/home/user/Downloads/car-sales-extended-missing-data.
data
```

Out[114...

	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 [115...

```
data.dtypes
```

Out[115...

```
Make              object
Colour            object
Odometer (KM)    float64
Doors            float64
Price            float64
dtype: object
```

In [117...

```
data.isna().sum()
```

Out[117...

```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

Steps we want to do (all in one cell):

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

In [134...

```
# 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
from sklearn.model_selection import RandomizedSearchCV

# 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("/home/user/Downloads/car-sales-extended-missing-data.
data. dropna(subset = ["Price"],inplace = True)

#Define different features and transformer pipeline
categorical_features = ["Make", "Colour"]
categorical_transformer = Pipeline(steps = [("imputer", SimpleImputer(str
                                             ("onehot", OneHotEncoder(han

door_feature = ["Doors"]
door_transformer = Pipeline(steps = [("imputer", SimpleImputer(strategy =

numeric_features = ["Odometer (KM)"]
numeric_transformer = Pipeline(steps = [("imputer", SimpleImputer(strateg

# Setup preprocessing steps (fill missing values, then convert to numbers
preprocessor = ColumnTransformer(transformers = [("cat", categorical_tran
                                                 ("door", door_transforme
                                                  ("num", numeric_transfo



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

# Split data
X = data.drop("Price", axis = 1)
y = data["Price"]
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[134…

```
0.22188417408787875
```

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

In [ ]: