Introduction to Scikit-Learn (sklearn)

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

What we're going to cover:

In [1]:
```
# Let's listify the contents
what_were_covering = [
    "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 [2]:
```
what_were_covering
```

Out[2]:
```
['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 [3]:
```
# 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 [4]:
```
# 1. Get the data ready
import pandas as pd
heart_disease = pd.read_csv("../data/heart-disease.csv")
heart_disease
```

Out[4]:

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

303 rows × 14 columns

In [5]:
```
# Create X (features matrix)
X = heart_disease.drop("target", axis=1)

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

In [6]:
```python
# 2. Choose the right model and hyperparameters
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100)

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

Out[6]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [7]:
```python
# 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 [8]:
```python
clf.fit(X_train, y_train);
```

In [9]:
```python
X_train
```

Out[9]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
170	56	1	2	130	256	1	0	142	1	0.6	1	1	1
44	39	1	2	140	321	0	0	182	0	0.0	2	0	2
145	70	1	1	156	245	0	0	143	0	0.0	2	0	2
175	40	1	0	110	167	0	0	114	1	2.0	1	0	3
61	54	1	1	108	309	0	1	156	0	0.0	2	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
190	51	0	0	130	305	0	1	142	1	1.2	1	0	3
194	60	1	2	140	185	0	0	155	0	3.0	1	0	2
178	43	1	0	120	177	0	0	120	1	2.5	1	0	3
75	55	0	1	135	250	0	0	161	0	1.4	1	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

242 rows × 13 columns

In [10]:
```python
# make a prediction
y_label = clf.predict(np.array([0, 2, 3, 4]))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-7cea9660990e> in <module>
      1 # make a prediction
----> 2 y_label = clf.predict(np.array([0, 2, 3, 4]))

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in pred
ict(self, X)
    610             The predicted classes.
    611         """
--> 612         proba = self.predict_proba(X)
    613 
    614         if self.n_outputs_ == 1:

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in pred
ict_proba(self, X)
    654         check_is_fitted(self)
    655         # Check data
--> 656         X = self._validate_X_predict(X)
    657 
    658         # Assign chunk of trees to jobs

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in _val
idate_X_predict(self, X)
    410         check_is_fitted(self)
    411 
--> 412         return self.estimators_[0]._validate_X_predict(X, check_input=True)
    413 
    414     @property

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/tree/_classes.py in _valida
te_X_predict(self, X, check_input)
    378         """Validate X whenever one tries to predict, apply, predict_proba"""
    379         if check_input:
--> 380             X = check_array(X, dtype=DTYPE, accept_sparse="csr")
    381             if issparse(X) and (X.indices.dtype != np.intc or
    382                                 X.indptr.dtype != np.intc):

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/utils/validation.py in chec
k_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_
nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    554                     "Reshape your data either using array.reshape(-1, 1) if "
    555                     "your data has a single feature or array.reshape(1, -1) "
--> 556                     "if it contains a single sample.".format(array))
    557 
    558         # in the future np.flexible dtypes will be handled like object dtypes

ValueError: Expected 2D array, got 1D array instead:
array=[0. 2. 3. 4.].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1,
 -1) if it contains a single sample.
```

In [11]:
```python
y_preds = clf.predict(X_test)
y_preds
```

Out[11]:
```
array([1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1])
```

In [12]:
```python
y_test
```

Out[12]:
```
50     1
83     1
295    0
120    1
40     1
      ..
25     1
246    0
205    0
3      1
7      1
Name: target, Length: 61, dtype: int64
```

In [13]:
```python
# 4. Evaluate the model on the training data and test data
clf.score(X_train, y_train)
```

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

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

Out[14]:
```
0.7540983606557377
```

In [15]:
```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

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

```
              precision    recall  f1-score   support

           0       0.71      0.79      0.75        28
           1       0.80      0.73      0.76        33

    accuracy                           0.75        61
   macro avg       0.75      0.76      0.75        61
weighted avg       0.76      0.75      0.75        61
```

In [16]:
```python
confusion_matrix(y_test, y_preds)
```

Out[16]:
```
array([[22,  6],
       [ 9, 24]])
```

In [17]:
```python
accuracy_score(y_test, y_preds)
```

Out[17]:
```
0.7540983606557377
```

In [18]:
```python
# 5. Improve a model
# Try different amount of n_estimators
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    clf = RandomForestClassifier(n_estimators=i).fit(X_train, y_train)
    print(f"Model accuracy on test set: {clf.score(X_test, y_test) * 100:.2f}%")
    print("")
```

```
Trying model with 10 estimators...
Model accuracy on test set: 75.41%

Trying model with 20 estimators...
Model accuracy on test set: 78.69%

Trying model with 30 estimators...
Model accuracy on test set: 77.05%

Trying model with 40 estimators...
Model accuracy on test set: 80.33%

Trying model with 50 estimators...
Model accuracy on test set: 80.33%

Trying model with 60 estimators...
Model accuracy on test set: 80.33%

Trying model with 70 estimators...
Model accuracy on test set: 81.97%

Trying model with 80 estimators...
Model accuracy on test set: 78.69%

Trying model with 90 estimators...
Model accuracy on test set: 80.33%
```

In [19]:
```python
# 6. Save a model and load it
import pickle

pickle.dump(clf, open("random_forst_model_1.pkl", "wb"))
```

In [20]:
```
loaded_model = pickle.load(open("random_forst_model_1.pkl", "rb"))
loaded_model.score(X_test, y_test)
```

Out[20]: 0.8032786885245902

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

Three main things we have to do: 1. Split the data into features and labels (usually `X` & `y`) 2. Filling (also called imputing) or disregarding missing values 3. Converting non-numerical values to numerical values (also called feature encoding)

In [21]:
```
heart_disease.head()
```

Out[21]:

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

In [22]:
```
X = heart_disease.drop("target", axis=1)
X.head()
```

Out[22]:

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

In [23]:
```
y = heart_disease["target"]
y.head()
```

Out[23]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [24]:
```
# 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=0.3)
```

In [25]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[25]: ((212, 13), (91, 13), (212,), (91,))

In [26]:
```
X.shape[0] * 0.8
```

Out[26]: 242.4

In [27]:
```
242 + 61
```

Out[27]: 303

In [28]:
```
len(heart_disease)
```

Out[28]: 303

1.1 Make sure it's all numerical

In [29]:
```python
car_sales = pd.read_csv("../data/car-sales-extended.csv")
car_sales.head()
```

Out[29]:

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

In [30]:
```python
car_sales["Doors"].value_counts()
```

Out[30]:
```
4    856
5     79
3     65
Name: Doors, dtype: int64
```

In [31]:
```python
len(car_sales)
```

Out[31]:
```
1000
```

In [32]:
```python
car_sales.dtypes
```

Out[32]:
```
Make             object
Colour           object
Odometer (KM)     int64
Doors             int64
Price             int64
dtype: object
```

In [33]:
```python
# Split into X/y
X = car_sales.drop("Price", axis=1)
y = car_sales["Price"]

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

In [34]:
```
# Build machine learning model
from sklearn.ensemble import RandomForestRegressor

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-2eeea2d0b490> in <module>
      3 
      4 model = RandomForestRegressor()
----> 5 model.fit(X_train, y_train)
      6 model.score(X_test, y_test)

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in fit
(self, X, y, sample_weight)
    293         """
    294         # Validate or convert input data
--> 295         X = check_array(X, accept_sparse="csc", dtype=DTYPE)
    296         y = check_array(y, accept_sparse='csc', ensure_2d=False, dtype=None)
    297         if sample_weight is not None:

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/utils/validation.py in chec
k_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_
nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    529                     array = array.astype(dtype, casting="unsafe", copy=False)
    530                 else:
--> 531                     array = np.asarray(array, order=order, dtype=dtype)
    532             except ComplexWarning:
    533                 raise ValueError("Complex data not supported\n"

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/numpy/core/_asarray.py in asarray
(a, dtype, order)
     83 
     84     """
---> 85     return array(a, dtype, copy=False, order=order)
     86 
     87 

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

In [35]:
```
X.head()
```

Out[35]:

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

In [36]:
```
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

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

transformed_X = transformer.fit_transform(X)
transformed_X
```

Out[36]:
```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 3.54310e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 1.92714e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 8.47140e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 6.66040e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.15883e+05],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.48360e+05]])
```

In [37]:
```
X.head()
```

Out[37]:

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

In [38]:
```
pd.DataFrame(transformed_X)
```

Out[38]:

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

1000 rows × 13 columns

In [39]:
```python
# Another way to do it with pd.dummies...
dummies = pd.get_dummies(car_sales[["Make", "Colour", "Doors"]])
dummies
```

Out[39]:

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

1000 rows × 10 columns

In [40]:
```python
# Let's refit the model
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(transformed_X,
                                                    y,
                                                    test_size=0.2)

model.fit(X_train, y_train)
```

Out[40]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [41]:
```python
X.head()
```

Out[41]:

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

In [42]:
```python
model.score(X_test, y_test)
```

Out[42]:
```
0.3235867221569877
```

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 [43]:
```
# Import car sales missing data
car_sales_missing = pd.read_csv("../data/car-sales-extended-missing-data.csv")
car_sales_missing.head()
```

Out[43]:

	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 [44]:
```
car_sales_missing.isna().sum()
```

Out[44]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [45]:
```
# Create X & y
X = car_sales_missing.drop("Price", axis=1)
y = car_sales_missing["Price"]
```

In [46]:
```
# Let's try and convert our data to numbers
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

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

transformed_X = transformer.fit_transform(X)
transformed_X
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-f532939289ac> in <module>
     11                                   remainder="passthrough")
     12
---> 13 transformed_X = transformer.fit_transform(X)
     14 transformed_X

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/compose/_column_transforme
r.py in fit_transform(self, X, y)
    516         self._validate_remainder(X)
    517
--> 518         result = self._fit_transform(X, y, _fit_transform_one)
    519
    520         if not result:

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/compose/_column_transforme
r.py in _fit_transform(self, X, y, func, fitted)
    455                     message=self._log_message(name, idx, len(transformers)))
```

In [47]:
```
car_sales_missing
```

Out[47]:

	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 [48]:
```
car_sales_missing["Doors"].value_counts()
```

Out[48]:
```
4.0    811
5.0     75
3.0     64
Name: Doors, dtype: int64
```

Option 1: Fill missing data with Pandas

In [49]:
```
# Fill the "Make" column
car_sales_missing["Make"].fillna("missing", inplace=True)

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

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

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

In [50]:
```
# Check our dataframe again
car_sales_missing.isna().sum()
```

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

In [51]:
```
# Remove rows with missing Price value
car_sales_missing.dropna(inplace=True)
```

In [52]:
```
car_sales_missing.isna().sum()
```

Out[52]:
```
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price            0
dtype: int64
```

In [53]:
```
len(car_sales_missing)
```

Out[53]:
```
950
```

In [54]:
```
X = car_sales_missing.drop("Price", axis=1)
y = car_sales_missing["Price"]
```

In [55]:
```
# Let's try and convert our data to numbers
# Turn the categories into numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

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

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

Out[55]:
```
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: Filling missing data and transforming categorical data with Scikit-Learn

Note: This section is different to the video. The video shows filling and transforming the entire dataset (`X`) and although the techniques are correct, it's best to fill and transform training and test sets separately (as shown in the code below).

The main takeaways:

- Split your data first (into train/test)
- Fill/transform the training set and test sets separately

Thank you Robert [for pointing this out (https://www.udemy.com/course/complete-machine-learning-and-data-science-zero-to-mastery/learn/#questions/9506426)](https://www.udemy.com/course/complete-machine-learning-and-data-science-zero-to-mastery/learn/#questions/9506426).

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

Out[56]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0

In [57]:
```
car_sales_missing.isna().sum()
```

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

In [58]:
```python
# Drop the rows with no labels
car_sales_missing.dropna(subset=["Price"], inplace=True)
car_sales_missing.isna().sum()
```

Out[58]:
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
Price             0
dtype: int64
```

In [59]:
```python
# Split into X & y
X = car_sales_missing.drop("Price", axis=1)
y = car_sales_missing["Price"]

# Split data into train and test
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2)
```

In [60]:
```python
# Check missing values
X.isna().sum()
```

Out[60]:
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
dtype: int64
```

In [61]:
```python
# 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_feature = ["Doors"]
num_features = ["Odometer (KM)"]

# Create an imputer (something that fills missing data)
imputer = ColumnTransformer([
    ("cat_imputer", cat_imputer, cat_features),
    ("door_imputer", door_imputer, door_feature),
    ("num_imputer", num_imputer, num_features)
])

# Fill train and test values separately
filled_X_train = imputer.fit_transform(X_train)
filled_X_test = imputer.transform(X_test)

# Check filled X_train
filled_X_train
```

Out[61]:
```
array([['Honda', 'White', 4.0, 71934.0],
       ['Toyota', 'Red', 4.0, 162665.0],
       ['Honda', 'White', 4.0, 42844.0],
       ...,
       ['Toyota', 'White', 4.0, 196225.0],
       ['Honda', 'Blue', 4.0, 133117.0],
       ['Honda', 'missing', 4.0, 150582.0]], dtype=object)
```

In [62]:
```python
# Get our transformed data array's back into DataFrame's
car_sales_filled_train = pd.DataFrame(filled_X_train,
                                      columns=["Make", "Colour", "Doors", "Odometer (KM)"])

car_sales_filled_test = pd.DataFrame(filled_X_test,
                                     columns=["Make", "Colour", "Doors", "Odometer (KM)"])

# Check missing data in training set
car_sales_filled_train.isna().sum()
```

Out[62]:
```
Make             0
Colour           0
Doors            0
Odometer (KM)    0
dtype: int64
```

In [63]:
```python
# Check to see the original... still missing values
car_sales_missing.isna().sum()
```

Out[63]:
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
Price             0
dtype: int64
```

In [64]:
```python
# Now let's one hot encode the features with the same code as before
categorical_features = ["Make", "Colour", "Doors"]
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                  remainder="passthrough")

# Fill train and test values separately
transformed_X_train = transformer.fit_transform(car_sales_filled_train)
transformed_X_test = transformer.transform(car_sales_filled_test)

# Check transformed and filled X_train
transformed_X_train.toarray()
```

Out[64]:
```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 7.19340e+04],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 1.62665e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 4.28440e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 1.96225e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 1.33117e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 1.50582e+05]])
```

In [65]:
```python
# Now we've transformed X, let's see if we can fit a model
np.random.seed(42)
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()

# Make sure to use transformed (filled and one-hot encoded X data)
model.fit(transformed_X_train, y_train)
model.score(transformed_X_test, y_test)
```

Out[65]:
```
0.21229043336119102
```

In [66]:
```python
# Check length of transformed data (filled and one-hot encoded)
# vs. length of original data
len(transformed_X_train.toarray())+len(transformed_X_test.toarray()), len(car_sales)
```

Out[66]:
```
(950, 1000)
```

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/algorithm for our problem

Scikit-Learn uses estimator as another term for machine learning model or algorithm.

- Classification - predicting whether a sample is one thing or another
- Regression - predicting a number

Step 1 - Check the Scikit-Learn machine learning map... [https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

2.1 Picking a machine learning model for a regression problem

In [67]:
```
# Import Boston housing dataset
from sklearn.datasets import load_boston
boston = load_boston()
boston;
```

In [68]:
```
boston_df = pd.DataFrame(boston["data"], columns=boston["feature_names"])
boston_df["target"] = pd.Series(boston["target"])
boston_df.head()
```

Out[68]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [69]:
```
# How many samples?
len(boston_df)
```

Out[69]: 506

In [70]:
```
# Let's try the Ridge Regression model
from sklearn.linear_model import Ridge

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

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

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

# Instantiate Ridge model
model = Ridge()
model.fit(X_train, y_train)

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

Out[70]: 0.6662221670168518

How do we improve this score?

What if Ridge wasn't working?

Let's refer back to the map... [https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

In [71]:
```
# Let's try the Random Forst Regressor
from sklearn.ensemble import RandomForestRegressor

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

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

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

# Instatiate Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)

# Evaluate the Random Forest Regressor
rf.score(X_test, y_test)
```

Out[71]: 0.8896648705127477

In [72]:
```
# Check the Ridge model again
model.score(X_test, y_test)
```

Out[72]: 0.6662221670168518

2.2 Choosing an estimator for a classification problem

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

```
In [73]: heart_disease = pd.read_csv("data/heart-disease.csv")
         heart_disease.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-73-44f78f3704d4> in <module>
----> 1 heart_disease = pd.read_csv("data/heart-disease.csv")
      2 heart_disease.head()

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/pandas/io/parsers.py in parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    674         )
    675 
--> 676         return _read(filepath_or_buffer, kwds)
    677 
    678     parser_f.__name__ = name

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer, kwds)
    446 
    447     # Create the parser.
--> 448     parser = TextFileReader(fp_or_buf, **kwds)
    449 
    450     if chunksize or iterator:

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/pandas/io/parsers.py in __init__(self, f, engine, **kwds)
    878             self.options["has_index_names"] = kwds["has_index_names"]
    879 
--> 880         self._make_engine(self.engine)
    881 
    882     def close(self):

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/pandas/io/parsers.py in _make_engine(self, engine)
   1112     def _make_engine(self, engine="c"):
   1113         if engine == "c":
-> 1114             self._engine = CParserWrapper(self.f, **self.options)
   1115         else:
   1116             if engine == "python":

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/pandas/io/parsers.py in __init__(self, src, **kwds)
   1889         kwds["usecols"] = self.usecols
   1890 
-> 1891         self._reader = parsers.TextReader(src, **kwds)
   1892         self.unnamed_cols = self._reader.unnamed_cols
   1893 

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source()

FileNotFoundError: [Errno 2] File data/heart-disease.csv does not exist: 'data/heart-disease.csv'
```

```
In [74]: len(heart_disease)
```

```
Out[74]: 303
```

Consulting the map and it says to try `LinearSVC`.

In [75]:
```python
# 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=10000)
clf.fit(X_train, y_train)

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

```
/Users/daniel/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/svm/_base.py:94
7: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[75]:
```
0.47540983606557374
```

In [76]:
```python
heart_disease["target"].value_counts()
```

Out[76]:
```
1    165
0    138
Name: target, dtype: int64
```

In [77]:
```python
# 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[77]:
```
0.8524590163934426
```

Tidbit:

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

In [78]:
```
heart_disease
```

Out[78]:

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

303 rows × 14 columns

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 [79]:
```
# 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 learned)
clf.score(X_test, y_test)
```

Out[79]: 0.8524590163934426

In [80]:
```
X.head()
```

Out[80]:

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

In [81]:
```
y.tail()
```

Out[81]:
```
298    0
299    0
300    0
301    0
302    0
Name: target, dtype: int64
```

Random Forest model deep dive

These resources will help you understand what's happening inside the Random Forest models we've been using.

- Random Forest Wikipedia (https://en.wikipedia.org/wiki/Random_forest)
- Random Forest Wikipedia (simple version) (https://simple.wikipedia.org/wiki/Random_forest)
- Random Forests in Python (http://blog.yhat.com/posts/random-forests-in-python.html) by yhat
- An Implementation and Explanation of the Random Forest in Python (https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76) by Will Koehrsen

3.2 Make predictions using a machine learning model

2 ways to make predictions:

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

In [82]:
```python
# Use a trained model to make predictions
clf.predict(np.array([1, 7, 8, 3, 4])) # this doesn't work...
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-82-5908053f578c> in <module>
      1 # Use a trained model to make predictions
----> 2 clf.predict(np.array([1, 7, 8, 3, 4])) # this doesn't work...

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in pred
ict(self, X)
    610             The predicted classes.
    611         """
--> 612         proba = self.predict_proba(X)
    613 
    614         if self.n_outputs_ == 1:

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in pred
ict_proba(self, X)
    654         check_is_fitted(self)
    655         # Check data
--> 656         X = self._validate_X_predict(X)
    657 
    658         # Assign chunk of trees to jobs

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/ensemble/_forest.py in _val
idate_X_predict(self, X)
    410         check_is_fitted(self)
    411 
--> 412         return self.estimators_[0]._validate_X_predict(X, check_input=True)
    413 
    414     @property

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/tree/_classes.py in _valida
te_X_predict(self, X, check_input)
    378         """Validate X whenever one tries to predict, apply, predict_proba"""
    379         if check_input:
--> 380             X = check_array(X, dtype=DTYPE, accept_sparse="csr")
    381             if issparse(X) and (X.indices.dtype != np.intc or
    382                                 X.indptr.dtype != np.intc):

~/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/utils/validation.py in chec
k_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_
nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    554                     "Reshape your data either using array.reshape(-1, 1) if "
    555                     "your data has a single feature or array.reshape(1, -1) "
--> 556                     "if it contains a single sample.".format(array))
    557 
    558         # in the future np.flexible dtypes will be handled like object dtypes

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

In [83]:
```python
X_test.head()
```

Out[83]:

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

In [84]:
```python
clf.predict(X_test)
```

Out[84]:
```
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 [85]:
```
np.array(y_test)
```

Out[85]:
```
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 [86]:
```
# Compare predictions to truth labels to evaluate the model
y_preds = clf.predict(X_test)
np.mean(y_preds == y_test)
```

Out[86]:
```
0.8524590163934426
```

In [87]:
```
clf.score(X_test, y_test)
```

Out[87]:
```
0.8524590163934426
```

In [88]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_preds)
```

Out[88]:
```
0.8524590163934426
```

Make predictions with `predict_proba()` - use this if someone asks you "what's the probability your model is assigning to each prediction?"

In [89]:
```
# predict_proba() returns probabilities of a classification label
clf.predict_proba(X_test[:5])
```

Out[89]:
```
array([[0.89, 0.11],
       [0.49, 0.51],
       [0.43, 0.57],
       [0.84, 0.16],
       [0.18, 0.82]])
```

In [90]:
```
# Let's predict() on the same data...
clf.predict(X_test[:5])
```

Out[90]:
```
array([0, 1, 1, 0, 1])
```

In [91]:
```
X_test[:5]
```

Out[91]:

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

In [92]:
```
heart_disease["target"].value_counts()
```

Out[92]:
```
1    165
0    138
Name: target, dtype: int64
```

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

In [93]:
```
boston_df.head()
```

Out[93]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [94]:
```python
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

# Create the data
X = boston_df.drop("target", axis=1)
y = boston_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)

# Instantiate and fit model
model = RandomForestRegressor(n_estimators=100).fit(X_train, y_train)

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

In [95]:
```python
y_preds[:10]
```

Out[95]:
```
array([23.002, 30.826, 16.734, 23.467, 16.853, 21.725, 19.232, 15.239,
       21.067, 20.738])
```

In [96]:
```python
np.array(y_test[:10])
```

Out[96]:
```
array([23.6, 32.4, 13.6, 22.8, 16.1, 20. , 17.8, 14. , 19.6, 16.8])
```

In [97]:
```python
# Compare the predictions to the truth
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_preds)
```

Out[97]:
```
2.1226372549019623
```

4. Evaluating a machine learning model

Three ways to evaluate Scikit-Learn models/esitmators:

1. Estimator `score` method
2. The `scoring` parameter
3. Problem-specific metric functions.

4.1 Evaluating a model with the `score` method

In [98]:
```python
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

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

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

clf = RandomForestClassifier()

clf.fit(X_train, y_train)
```

Out[98]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [99]:
```python
clf.score(X_train, y_train)
```

Out[99]:
```
1.0
```

In [100]:
```python
clf.score(X_test, y_test)
```

Out[100]:
```
0.8524590163934426
```

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

In [101]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

# Create the data
X = boston_df.drop("target", axis=1)
y = boston_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)

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

In [102]:
```
model.score(X_test, y_test)
```

Out[102]:
```
0.873969014117403
```

4.2 Evaluating a model using the `scoring` parameter

In [103]:
```
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"]

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

clf = RandomForestClassifier(n_estimators=100)

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

In [104]:
```
clf.score(X_test, y_test)
```

Out[104]:
```
0.8524590163934426
```

In [105]:
```
cross_val_score(clf, X, y, cv=5)
```

Out[105]:
```
array([0.81967213, 0.86885246, 0.81967213, 0.78333333, 0.76666667])
```

In [106]:
```
cross_val_score(clf, X, y, cv=10)
```

Out[106]:
```
array([0.90322581, 0.80645161, 0.87096774, 0.9       , 0.86666667,
       0.8       , 0.73333333, 0.86666667, 0.73333333, 0.8       ])
```

In [107]:
```
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[107]:
```
(0.8524590163934426, 0.8248087431693989)
```

In [108]:
```
# Default scoring parameter of classifier = mean accuracy
clf.score()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-108-cca012993b3a> in <module>
      1 # Default scoring parameter of classifier = mean accuracy
----> 2 clf.score()

TypeError: score() missing 2 required positional arguments: 'X' and 'y'
```

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

4.2.1 Classification model evaluation metrics

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

Accuracy

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

In [ ]:
```
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)
```

In [ ]:
```
np.mean(cross_val_score)
```

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

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

- Area under curve (AUC)
- ROC curve

ROC curves are a comparison of a model's true postive 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 predicts 0 when truth is 0
- False negative = model predicts 0 when truth is 1

In [109]:
```
# Create X_test... etc
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [110]:
```
from sklearn.metrics import roc_curve

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

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

y_probs[:10], len(y_probs)
```

Out[110]:
```
(array([[0.51, 0.49],
        [0.17, 0.83],
        [0.51, 0.49],
        [0.72, 0.28],
        [0.43, 0.57],
        [0.12, 0.88],
        [0.3 , 0.7 ],
        [0.97, 0.03],
        [0.15, 0.85],
        [0.4 , 0.6 ]]),
 61)
```

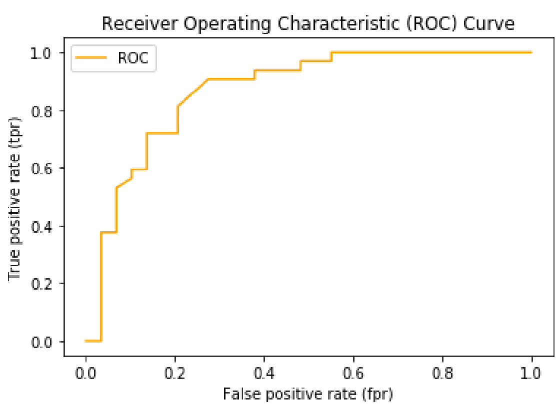
In [111]:
```
y_probs_positive = y_probs[:, 1]
y_probs_positive[:10]
```

Out[111]:
```
array([0.49, 0.83, 0.49, 0.28, 0.57, 0.88, 0.7 , 0.03, 0.85, 0.6 ])
```

In [112]:
```
# Caculate fpr, tpr and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)

# Check the false positive rates
fpr
```

Out[112]:
```
array([0.        , 0.03448276, 0.03448276, 0.03448276, 0.03448276,
       0.03448276, 0.03448276, 0.06896552, 0.06896552, 0.06896552,
       0.10344828, 0.10344828, 0.13793103, 0.13793103, 0.13793103,
       0.20689655, 0.20689655, 0.20689655, 0.27586207, 0.37931034,
       0.37931034, 0.48275862, 0.48275862, 0.55172414, 0.55172414,
       1.        ])
```

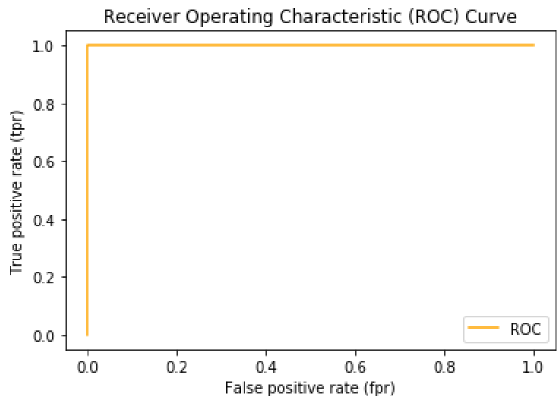
In [113]:
```
# Create a function for plotting ROC curves
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):
    """
    Plots a ROC curve given the false positive rate (fpr)
    and true positive rate (tpr) of a model.
    """
    # Plot roc curve
    plt.plot(fpr, tpr, color="orange", label="ROC")
    # Plot line with no predictive power (baseline)
    #plt.plot([0, 1], [0, 1], color="darkblue", linestyle="--", label="Guessing")

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

plot_roc_curve(fpr, tpr)
```



In [114]:
```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_probs_positive)
```

Out[114]:
```
0.8669181034482759
```

In [115]:
```
# Plot perfect ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_test)
plot_roc_curve(fpr, tpr)
```



In [116]:
```
# Perfect AUC score
roc_auc_score(y_test, y_test)
```

Out[116]: `1.0`

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 idea of where the model is getting confused.

In [117]:
```
from sklearn.metrics import confusion_matrix

y_preds = clf.predict(X_test)

confusion_matrix(y_test, y_preds)
```

Out[117]:
```
array([[23,  6],
       [ 6, 26]])
```

In [118]:
```
# Visualize confusion matrix with pd.crosstab()
pd.crosstab(y_test,
            y_preds,
            rownames=["Actual Labels"],
            colnames=["Predicted Labels"])
```

Out[118]:

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

In [119]:
```
22 + 7 + 8 + 24
```

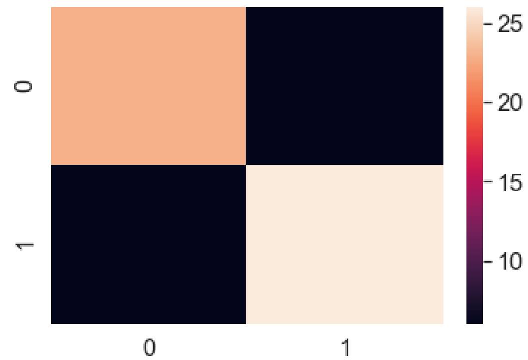
Out[119]: `61`

In [120]:
```
len(X_test)
```

Out[120]: `61`

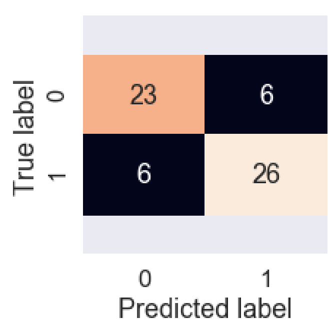
In [121]:
```
# # How install a conda package into the current envrionment from a Jupyter Notebook
# import sys
# !conda install --yes --prefix {sys.prefix} seaborn
```

In [122]:
```python
# Make our confusion matrix more visual with Seaborn's heatmap()
import seaborn as sns

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

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

# Plot it using Seaborn
sns.heatmap(conf_mat);
```



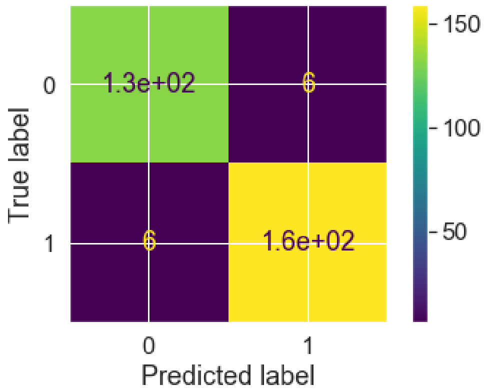
Note: In the original notebook, the function below had the `"True label"` as the x-axis label and the `"Predicted label"` as the y-axis label. But due to the way `confusion_matrix()` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html) outputs values, these should be swapped around. The code below has been corrected.

In [123]:
```python
def plot_conf_mat(conf_mat):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3,3))
    ax = sns.heatmap(conf_mat,
                     annot=True, # Annotate the boxes with conf_mat info
                     cbar=False)
    plt.xlabel("Predicted label")
    plt.ylabel("True label")

    # Fix the broken annotations (this happened in Matplotlib 3.1.1)
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top-0.5);

plot_conf_mat(conf_mat)
```



In [124]:
```python
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(clf, X, y)
```

Out[124]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff630bc2d10>
```



Classification Report

In [125]:
```python
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

           0       0.79      0.79      0.79        29
           1       0.81      0.81      0.81        32

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

In [126]:
```python
# Where precision and recall become valuable
disease_true = np.zeros(10000)
disease_true[0] = 1 # only one positive case

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

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

```
/Users/daniel/Desktop/ml-course/zero-to-mastery-ml/env/lib/python3.7/site-packages/sklearn/metrics/_classi
fication.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in la
bels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[126]:

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

To summarize 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 important 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.
- **F1-score** is a combination of precision and recall.

4.2.2 Regression model evaluation metrics

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

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

R^2

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

In [127]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

X = boston_df.drop("target", axis=1)
y = boston_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);
```

In [128]:
```
model.score(X_test, y_test)
```

Out[128]: 0.873969014117403

In [129]:
```
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 [130]:
```
y_test.mean()
```

Out[130]: 21.488235294117644

In [131]:
```
# Model only predicting the mean gets an R^2 score of 0
r2_score(y_test, y_test_mean)
```

Out[131]: 0.0

In [132]:
```
# Model predicting perfectly the correct values gets an R^2 score of 1
r2_score(y_test, y_test)
```

Out[132]: 1.0

Mean absolue error (MAE)

MAE is the average of the aboslute differences between predictions and actual values. It gives you an idea of how wrong your models predictions are.

In [133]:
```
# Mean absolute error
from sklearn.metrics import mean_absolute_error

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

Out[133]: 2.1226372549019623

In [134]:
```
df = pd.DataFrame(data={"actual values": y_test,
                        "predicted values": y_preds})
df["differences"] = df["predicted values"] - df["actual values"]
df
```

Out[134]:

	actual values	predicted values	differences
173	23.6	23.002	-0.598
274	32.4	30.826	-1.574
491	13.6	16.734	3.134
72	22.8	23.467	0.667
452	16.1	16.853	0.753
...	...	...	...
412	17.9	13.030	-4.870
436	9.6	12.490	2.890
411	17.2	13.406	-3.794
86	22.5	20.219	-2.281
75	21.4	23.898	2.498

102 rows × 3 columns

Mean squared error (MSE)

In [135]:
```
# 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[135]: 9.242328990196082

In [136]:
```
# Calculate MSE by hand
squared = np.square(df["differences"])
squared.mean()
```

Out[136]: 9.242328990196082

4.2.3 Finally using the `scoring` parameter

In [137]:
```
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 [138]:
```
np.random.seed(42)
cv_acc = cross_val_score(clf, X, y, cv=5, scoring=None)
cv_acc
```

Out[138]: array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])

In [139]:
```
# Cross-validated accuracy
print(f'The cross-validated accuracy is: {np.mean(cv_acc)*100:.2f}%')
```

The cross-validated accuracy is: 82.48%

In [140]:
```python
np.random.seed(42)
cv_acc = cross_val_score(clf, X, y, cv=5, scoring="accuracy")
print(f'The cross-validated accuracy is: {np.mean(cv_acc)*100:.2f}%')
```

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

In [141]:
```python
# Precision
cv_precision = cross_val_score(clf, X, y, cv=5, scoring="precision")
np.mean(cv_precision)
```

Out[141]:
```
0.8085601538512754
```

In [142]:
```python
# Recall
cv_recall = cross_val_score(clf, X, y, cv=5, scoring="recall")
np.mean(cv_recall)
```

Out[142]:
```
0.8424242424242424
```

In [143]:
```python
cv_f1 = cross_val_score(clf, X, y, cv=5, scoring="f1")
np.mean(cv_f1)
```

Out[143]:
```
0.841476533416832
```

How about our regression model?

In [144]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

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

model = RandomForestRegressor(n_estimators=100)
```

In [145]:
```python
np.random.seed(42)
cv_r2 = cross_val_score(model, X, y, cv=5, scoring=None)
np.mean(cv_r2)
```

Out[145]:
```
0.622375083951403
```

In [146]:
```python
np.random.seed(42)
cv_r2 = cross_val_score(model, X, y, cv=5, scoring="r2")
cv_r2
```

Out[146]:
```
array([0.76861165, 0.85851765, 0.74941131, 0.47891315, 0.25642166])
```

In [147]:
```python
# Mean absolute error
cv_mae = cross_val_score(model, X, y, cv=5, scoring="neg_mean_absolute_error")
cv_mae
```

Out[147]:
```
array([-2.12751961, -2.53956436, -3.42026733, -3.82432673, -3.06893069])
```

In [148]:
```python
# Mean squared error
cv_mse = cross_val_score(model, X, y, cv=5, scoring="neg_mean_squared_error")
np.mean(cv_mse)
```

Out[148]:
```
-21.02253826604542
```

4.3 Using different evalution metrics as Scikit-Learn functions

Classification evaluation functions

In [149]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

np.random.seed(42)

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

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

clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)

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

# Evaluate the classifier
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)}")
print(f"Recall: {recall_score(y_test, y_preds)}")
print(f"F1: {f1_score(y_test, y_preds)}")
```

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

Regression evaluation functions

In [150]:
```python
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

np.random.seed(42)

X = boston_df.drop("target", axis=1)
y = boston_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)

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

# Evaluate the regression model
print("Regression model metrics on the test set")
print(f"R^2: {r2_score(y_test, y_preds)}")
print(f"MAE: {mean_absolute_error(y_test, y_preds)}")
print(f"MSE: {mean_squared_error(y_test, y_preds)}")
```

```
Regression model metrics on the test set
R^2: 0.8739690141174031
MAE: 2.1226372549019623
MSE: 9.242328990196082
```

5. Improving a model

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

From a data perspective:

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

From a model perspective:

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

Hyperparameters vs. Parameters

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

Three ways to adjust hyperparameters:

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

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

clf = RandomForestClassifier(n_estimator=100)
```

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

5.1 Tuning hyperparameters by hand

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

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

We're going to try and adjust:

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

In [ ]:
```
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(precision, 2),
                   "recall": round(recall, 2),
                   "f1": round(f1, 2)}
    print(f"Acc: {accuracy * 100:.2f}%")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 score: {f1:.2f}")

    return metric_dict
```

In [ ]:
```python
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)) # 15% of data
X_train, y_train = X[:train_split], y[:train_split]
X_valid, y_valid = X[train_split:valid_split], y[train_split:valid_split]
X_test, y_test = X[valid_split:], y[:valid_split]

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
```

In [ ]:
```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)

# Evalute the 2nd classsifier
clf_2_metrics = evaluate_preds(y_valid, y_preds_2)
```

5.2 Hyperparameter tuning with RandomizedSearchCV

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

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

np.random.seed(42)

# Split 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);
```

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

In [ ]:
```python
# 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)
```

5.3 Hyperparameter tuning with GridSearchCV

In [ ]:
```python
grid
```

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

In [ ]:
```python
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 GridSearchCV
# gs_clf = GridSearchCV(estimator=clf,
#                       param_grid=grid_2,
#                       cv=5,
#                       verbose=2)

# Fit the GridSearchCV version of clf
#gs_clf.fit(X_train, y_train);
```

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

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

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

Let's compare our different models metrics.

In [ ]:
```python
compare_metrics = pd.DataFrame({"baseline": baseline_metrics,
                                "clf_2": clf_2_metrics,
                                "random search": rs_metrics,
                                "grid search": gs_metrics})

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

6. Saving and loading trained machine learning models

Two ways to save and load machine learning models:

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

Pickle

In [ ]:
```python
import pickle

# Save an extisting model to file
pickle.dump(gs_clf, open("gs_random_random_forest_model_1.pkl", "wb"))
```

In [ ]:
```python
# Load a saved model
loaded_pickle_model = pickle.load(open("gs_random_random_forest_model_1.pkl", "rb"))
```

In [ ]:
```python
# Make some predictions
pickle_y_preds = loaded_pickle_model.predict(X_test)
evaluate_preds(y_test, pickle_y_preds)
```

Joblib

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

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

In [ ]:
```python
# Import a saved joblib model
loaded_joblib_model = load(filename="gs_random_forest_model_1.joblib")
```

In [ ]:
```python
# Make and evaluate joblib predictions
joblib_y_preds = loaded_joblib_model.predict(X_test)
evaluate_preds(y_test, joblib_y_preds)
```

7. Putting it all together!

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

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

In [ ]:
```python
data.isna().sum()
```

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

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

In [ ]:

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

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

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

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

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

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

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

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

# 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)
```

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

In [ ]:

```python
# Use GridSearchCV with our regression Pipeline
from sklearn.model_selection import GridSearchCV

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

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

In [ ]:

```python
gs_model.score(X_test, y_test)
```

In [ ]:

```python
what_were_covering
```