In [46]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Get the data ready

In [47]:
```
# Import dataset
heart_disease = pd.read_csv('heart.csv')

# View the data
heart_disease.head()
```

Out[47]:

	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

With this example, we're going to use all of the columns except the target column to predict the targert column.

In other words, using a patient's medical and demographic data to predict whether or not they have heart disease.

In [15]:
```
# Create X (all the feature columns)
X = heart_disease.drop("target", axis=1)

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

In [48]:
```
# 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)

# View the data shapes
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[48]:
```
((227, 13), (76, 13), (227,), (76,))
```

1. Choose the model/estimator You can do this using the Scikit-Learn machine learning map.

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

1. Fit the model to the data and use it to make a prediction

A model will (attempt to) learn the patterns in a dataset by calling the fit() function on it and passing it the data.

In [50]:
```
model.fit(X_train, y_train)
```

Out[50]:
```
RandomForestClassifier()
```

Once a model has learned patterns in data, you can use them to make a prediction with the predict() function.

In [19]:
```
# Make predictions
y_preds = model.predict(X_test)
```

In [20]:
```
# This will be in the same format as y_test
y_preds
```

Out[20]:
```
array([0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0], dtype=int64)
```

In [54]:
```
X_test.loc[200]
```

Out[54]:
```
age          44.0
sex           1.0
cp            0.0
trestbps    110.0
chol        197.0
fbs           0.0
restecg       0.0
thalach     177.0
exang         0.0
oldpeak       0.0
slope         2.0
ca            1.0
thal          2.0
Name: 200, dtype: float64
```

In [55]:
```
heart_disease.loc[200]
```

Out[55]:
```
age          44.0
sex           1.0
cp            0.0
trestbps    110.0
chol        197.0
fbs           0.0
restecg       0.0
thalach     177.0
exang         0.0
oldpeak       0.0
slope         2.0
ca            1.0
thal          2.0
target        0.0
Name: 200, dtype: float64
```

In [57]:
```
# Make a prediction on a single sample (has to be array)
model.predict(np.array(X_test.loc[200]).reshape(1, -1))
```

Out[57]:
```
array([1], dtype=int64)
```

1. Evaluate the model

A trained model/estimator can be evaluated by calling the score() function and passing it a collection of data.

In [58]:
```
# On the training set
model.score(X_train, y_train)
```

Out[58]:
```
1.0
```

In [59]:
```
# On the test set (unseen)
model.score(X_test, y_test)
```

Out[59]:
```
0.8157894736842105
```

1. Experiment to improve (hyperparameter tuning)

A model's first evaluation metrics aren't always its last. One way to improve a models predictions is with hyperparameter tuning.

In [60]:
```
# Try different numbers of estimators (n_estimators is a hyperparameter you can change)
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    model = RandomForestClassifier(n_estimators=i).fit(X_train, y_train)
    print(f"Model accruacy on test set: {model.score(X_test, y_test)}")
    print("")
```

```
Trying model with 10 estimators...
Model accruacy on test set: 0.8026315789473685

Trying model with 20 estimators...
Model accruacy on test set: 0.8026315789473685

Trying model with 30 estimators...
Model accruacy on test set: 0.8289473684210527

Trying model with 40 estimators...
Model accruacy on test set: 0.8026315789473685

Trying model with 50 estimators...
Model accruacy on test set: 0.8026315789473685

Trying model with 60 estimators...
Model accruacy on test set: 0.8421052631578947

Trying model with 70 estimators...
Model accruacy on test set: 0.8157894736842105

Trying model with 80 estimators...
Model accruacy on test set: 0.8421052631578947

Trying model with 90 estimators...
Model accruacy on test set: 0.8157894736842105
```

Note: It's best practice to test different hyperparameters with a validation set or cross-validation.

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

# Try different numbers of estimators with cross-validation and no cross-validation
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    model = RandomForestClassifier(n_estimators=i).fit(X_train, y_train)
    print(f"Model accruacy on test set: {model.score(X_test, y_test)}")
    print(f"Cross-validation score: {np.mean(cross_val_score(model, X, y, cv=5)) * 100}%")
    print("")
```

```
Trying model with 10 estimators...
Model accruacy on test set: 0.8026315789473685
Cross-validation score: 78.53551912568305%

Trying model with 20 estimators...
Model accruacy on test set: 0.8026315789473685
Cross-validation score: 79.84699453551912%

Trying model with 30 estimators...
Model accruacy on test set: 0.8289473684210527
Cross-validation score: 80.50819672131148%

Trying model with 40 estimators...
Model accruacy on test set: 0.8421052631578947
Cross-validation score: 82.15300546448088%

Trying model with 50 estimators...
Model accruacy on test set: 0.7894736842105263
Cross-validation score: 81.16393442622295%

Trying model with 60 estimators...
Model accruacy on test set: 0.8157894736842105
Cross-validation score: 83.47540983606557%

Trying model with 70 estimators...
Model accruacy on test set: 0.8289473684210527
Cross-validation score: 81.83060109289617%

Trying model with 80 estimators...
Model accruacy on test set: 0.8289473684210527
Cross-validation score: 82.81420765027322%

Trying model with 90 estimators...
Model accruacy on test set: 0.8157894736842105
Cross-validation score: 82.81967213114754%
```

1. Save a model for later use

A trained model can be exported and saved so it can be imported and used later. One way to save a model is using Python's pickle module.

In [62]:
```
import pickle

# Save trained model to file
pickle.dump(model, open("random_forest_model_1.pkl", "wb"))
```

In [64]:
```
# Load a saved model and make a prediction on a single example
loaded_model = pickle.load(open("random_forest_model_1.pkl", "rb"))
loaded_model.predict(np.array(X_test.loc[200]).reshape(1, -1))
```

Out[64]:
```
array([1], dtype=int64)
```

In [ ]:
```

```