scikit_learn_intoduction

January 18, 2026

1 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:

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

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

[1]:
```
import warnings
warnings.filterwarnings("default")
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
```

[3]:
```
# 1. Get the data ready
heart_disease = pd.read_csv("./data/heart-disease.csv")
heart_disease
```

[3]:

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

```
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     ...  ..   ...     ...
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 14 columns]
```

[4]:
```python
# Create X (features matrix)
X = heart_disease.drop("target", axis = 1)

#Create Y (Labels)
Y = heart_disease["target"]
```

[5]:
```python
# 2. Choose the right model and hyperparameters
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators = 100) # no need of the oarameter

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

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

[6]:
```
# 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)
```

[7]:
```
clf.fit(X_Train, Y_Train)
```

[7]:
```
RandomForestClassifier()
```

[8]:
```
# make a prediction
y_preds = clf.predict(X_Test)
```

[9]:
```
y_preds
```

[9]:
```
array([0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0])
```

[10]:
```
# 4. Evaluate the model
clf.score(X_Train, Y_Train)
```

[10]:
```
1.0
```

[11]:
```
clf.score(X_Test, Y_Test)
```

[11]:
```
0.8524590163934426
```

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

print(classification_report(Y_Test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.76      0.86      0.81        22
           1       0.92      0.85      0.88        39

    accuracy                           0.85        61
   macro avg       0.84      0.85      0.84        61
weighted avg       0.86      0.85      0.85        61
```

[13]:
```
confusion_matrix(Y_Test, y_preds)
```

[13]:
```
array([[19,  3],
       [ 6, 33]])
```

[14]:
```
accuracy_score(Y_Test,y_preds)
```

[14]:
```
0.8524590163934426
```

[15]:
```
# 5. Improve a model
# Try diffrent 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: 70.49%

Trying model with 20 estimators...
Model accuracy on test set: 83.61%

Trying model with 30 estimators...
Model accuracy on test set: 83.61%

Trying model with 40 estimators...
Model accuracy on test set: 86.89%

Trying model with 50 estimators...
Model accuracy on test set: 81.97%

Trying model with 60 estimators...
Model accuracy on test set: 88.52%

Trying model with 70 estimators...
Model accuracy on test set: 85.25%

Trying model with 80 estimators...
Model accuracy on test set: 86.89%

Trying model with 90 estimators...
Model accuracy on test set: 86.89%
```

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

with open("random_forest_model_1.pkl", "wb") as f:
```

```
    pickle.dump(clf, f)
```

[17]:
```
with open("random_forest_model_1.pkl", "rb") as f:
    loaded_model = pickle.load(f)


loaded_model.score(X_Test, Y_Test)
```

[17]:
```
0.8688524590163934
```

[18]:
```
# Let's listify the contents

whats_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"]
```

[19]:
```
whats_were_covering
```

[19]:
```
['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']
```

2 1. Getting the data ready

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

[20]:
```
heart_disease.head()
```

[20]:

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

```
   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

[21]:
```
X = heart_disease.drop("target", axis = 1)
X.head()
```

[21]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal
0   0     1
1   0     2
2   0     2
3   0     2
4   0     2
```

[22]:
```
y = heart_disease["target"]
y.head()
```

[22]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

[23]:
```
# Train and Test Sets
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(X,y, test_size = 0.2)
```

[24]:
```
X_train.shape, X_test.shape, y_test.shape, y_train.shape
```

[24]:
```
((242, 13), (61, 13), (61,), (242,))
```

[25]:
```
X.shape[0]*0.8
```

[25]:
```
242.4
```

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

[26]:
```
303
```

2.1 1.1 Make sure it's all numerical

[27]:
```
car_sales = pd.read_csv ("./data/car-sales-extended.csv")
car_sales.head()
```

[27]:

	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

[28]:
```
len(car_sales)
```

[28]:
```
1000
```

[29]:
```
car_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Make            1000 non-null   object
 1   Colour          1000 non-null   object
 2   Odometer (KM)   1000 non-null   int64 
 3   Doors           1000 non-null   int64 
 4   Price           1000 non-null   int64 
dtypes: int64(3), object(2)
memory usage: 39.2+ KB
```

[30]:
```
#Split into X and 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)
```

[31]:
```
# Build ML model
from sklearn.ensemble import RandomForestRegressor # Predict a number

model = RandomForestRegressor()
# model.fit(X_train, y_train) # throw a error because the model cant handle␣
 ↪strings
# model.score(X_Test,y_test)
```

[32]:
```
from sklearn.preprocessing import OneHotEncoder # Turn the categories into␣
↪numbers
from sklearn.compose import ColumnTransformer

categorical_features = ["Make", "Colour","Doors"] # Doors because there a only 3␣
↪typen (5,4 and 3) it is also categorical
one_hot = OneHotEncoder()
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                remainder = "passthrough")
transformed_X = transformer.fit_transform(X)

transformed_X
```

[32]:
```
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]], shape=(1000, 13))
```

[33]:
```
pd.DataFrame(transformed_X) # the odometer ha not changed because it is not␣
↪categorical
```

[33]:

	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 x 13 columns]

[34]:
```
# PREPROCESSING: One-Hot-Encoding for categorical columns (and passthrough for␣
 ↪numeric columns)
#
# Motivation:
# Most ML algorithms in scikit-learn expect purely numerical input (a matrix in␣
 ↪R^n).
# Raw categorical values such as "Toyota" or "Red" cannot be used directly.
# Also, we must NOT simply map categories to integers (e.g., Toyota=1, BMW=2),
# because that would create an artificial order and distance between categories.
#
# OneHotEncoder:
# - Learns the set of unique categories in each selected column during `fit()`.
# - During `transform()`, it converts each category into a binary vector:
#   Example for Colour = {Red, Blue, Black}:
#     Red   -> [1, 0, 0]
#     Blue  -> [0, 1, 0]
#     Black -> [0, 0, 1]
# - This produces "orthogonal" features (no implied ranking), which is the␣
 ↪correct
#   mathematical representation for nominal variables.
# - By default, the output is a sparse matrix to save memory (because most␣
 ↪entries are 0).
#
# Why "Doors" is treated as categorical:
# Even though Doors is numeric-looking (3, 4, 5), the values represent discrete␣
 ↪types,
# not a continuous measurement. The difference between 3 and 4 doors is not a␣
 ↪linear
# quantity in the same way as "Odometer (KM)". Treating it as categorical avoids
# misleading linear assumptions (especially important for linear models).
#
# ColumnTransformer:
# - Applies transformations to specific columns only.
# - Here: apply OneHotEncoder to ["Make", "Colour", "Doors"].
# - `remainder="passthrough"` ensures that all other columns (e.g., Odometer,␣
 ↪Price, etc.)
#   remain in the dataset unchanged. Without this, those columns would be␣
 ↪dropped.
#
# fit_transform(X):
# - `fit`: discovers categories in the selected columns (learns the encoding␣
 ↪schema).
# - `transform`: outputs a final numerical design matrix that concatenates:
#     (a) one-hot encoded categorical columns
#     (b) untouched numerical columns (passthrough)
```

```
# - The resulting matrix `transformed_X` is ready for model training in␣
 ↪scikit-learn.
#
# Best practice note:
# In a full ML workflow, you typically fit the transformer ONLY on training data
# (fit on X_train, transform X_train and X_test) to avoid data leakage.
```

[35]:
```
# another way
dummies = pd.get_dummies(car_sales[["Make", "Colour","Doors"]]) # dont work in␣
 ↪int columns
dummies
```

[35]:
```
     Doors  Make_BMW  Make_Honda  Make_Nissan  Make_Toyota  Colour_Black  \
0        4     False        True        False        False         False
1        5      True       False        False        False         False
2        4     False        True        False        False         False
3        4     False       False        False         True         False
4        3     False       False         True        False         False
..     ...       ...         ...          ...          ...           ...
995      4     False       False        False         True          True
996      3     False       False         True        False         False
997      4     False       False         True        False         False
998      4     False        True        False        False         False
999      4     False       False        False         True         False

     Colour_Blue  Colour_Green  Colour_Red  Colour_White
0          False         False       False          True
1           True         False       False         False
2          False         False       False          True
3          False         False       False          True
4           True         False       False         False
..           ...           ...         ...           ...
995        False         False       False         False
996        False         False       False          True
997         True         False       False         False
998        False         False       False          True
999         True         False       False         False

[1000 rows x 10 columns]
```

[36]:
```
# 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)
```

[36]:
```
RandomForestRegressor()
```

[37]:
```
model.score(X_test, y_test) # to less data try it in real life
```

[37]:
```
0.3235867221569877
```

2.1.1 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

[38]:
```
# import car_sales missing data
car_sales_missing = pd.read_csv("data/car-sales-extended-missing-data.csv")
car_sales_missing.head()
```

[38]:

	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

[39]:
```
car_sales_missing.isna().sum() #how many missing values
```

[39]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

[40]:
```
# Create X & y
X = car_sales_missing.drop("Price", axis = 1)
y = car_sales_missing["Price"]
```

[41]:
```
# Lets try and convert our data to numbers

from sklearn.preprocessing import OneHotEncoder # Turn the categories into␣
 ↪numbers
from sklearn.compose import ColumnTransformer

categorical_features = ["Make", "Colour","Doors"] # Doors because there a only 3␣
 ↪typen (5,4 and 3) it is also categorical
one_hot = OneHotEncoder(sparse_output=False) # getting a array not a sparse␣
 ↪Matrix
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
```

```
                                remainder = "passthrough")
transformed_X = transformer.fit_transform(X)

transformed_X
```

[41]:
```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        0.00000e+00, 3.54310e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 1.92714e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        0.00000e+00, 8.47140e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 0.00000e+00,
        0.00000e+00, 6.66040e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        0.00000e+00, 2.15883e+05],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        0.00000e+00, 2.48360e+05]], shape=(1000, 16))
```

[42]:
```
pd.DataFrame(transformed_X).head()
```

[42]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	35431.0
1	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	192714.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	84714.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	154365.0
4	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	181577.0

[43]:
```
# Example 1: Standardization (Z-Score Scaling)
# This transforms each numerical feature so that it has a mean of 0
# and a standard deviation of 1.
# The resulting values can be negative or positive and are not bounded.
# Standardization is commonly used for distance-based or gradient-based
# algorithms such as Logistic Regression, SVMs, and KNN.

from sklearn.preprocessing import StandardScaler
import pandas as pd

# Beispiel-Datensatz
data = {
```

```
    "Odometer_KM": [6000, 45000, 120000, 250000, 345000],
    "Repair_Cost": [100, 300, 700, 1200, 1700]
}

df = pd.DataFrame(data)

# Initialisiere StandardScaler
scaler = StandardScaler()

# Fit + Transform
df_scaled = pd.DataFrame(
    scaler.fit_transform(df),
    columns=df.columns
)

df_scaled
```

[43]:

	Odometer_KM	Repair_Cost
0	-1.159128	-1.193490
1	-0.852022	-0.852493
2	-0.261434	-0.170499
3	0.762253	0.681994
4	1.510332	1.534487

[44]:
```
# Example 2: Normalization (Min-Max Scaling)
# This rescales each numerical feature to a fixed range between 0 and 1.
# The minimum value of a feature becomes 0, the maximum value becomes 1,
# and all other values are scaled proportionally in between.
# Min-Max scaling is often used when features have known bounds or when
# training neural networks, where a consistent input range is beneficial.

from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Beispiel-Datensatz
data = {
    "Odometer_KM": [6000, 45000, 120000, 250000, 345000],
    "Repair_Cost": [100, 300, 700, 1200, 1700]
}

df = pd.DataFrame(data)

# Initialisiere Min-Max-Scaler
scaler = MinMaxScaler()

# Fit + Transform (nur für Demonstration an Gesamtdaten)
df_scaled = pd.DataFrame(
```

```
    scaler.fit_transform(df),
    columns=df.columns
)

df_scaled
```

[44]:

	Odometer_KM	Repair_Cost
0	0.000000	0.0000
1	0.115044	0.1250
2	0.336283	0.3750
3	0.719764	0.6875
4	1.000000	1.0000

2.1.2 Option 2: Fill missing values with Scikit-Learn

[45]:
```
car_sales_missing = pd.read_csv("data/car-sales-extended-missing-data.csv")
car_sales_missing.head()
```

[45]:

	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

[46]:
```
car_sales_missing.isna().sum()
```

[46]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

[47]:
```
#drop the rows with no labels
car_sales_missing.dropna(subset = ["Price"], inplace = True)
car_sales_missing.isna().sum()
```

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

[48]:
```
# Split into X & y
X = car_sales_missing.drop("Price", axis = 1)
y = car_sales_missing["Price"]
```

[49]:
```
# Fill missing values with scikit Learn (better way before Encoding into numbers)
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 a Imputer (something that fills missing data)
imputer = ColumnTransformer([
    ("cat_imputer", cat_imputer, cat_features),
    ("door_imputer",door_imputer,door_features),
    ("num:imputer", num_imputer, num_features)
])

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

[49]:
```
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]], shape=(950, 4), dtype=object)
```

[50]:
```
car_sales_filled = pd.DataFrame(filled_X,␣
 ↪columns=["Make","Colour","Doors","Odometer (KM)"])
```

[51]:
```
car_sales_filled
```

[51]:

	Make	Colour	Doors	Odometer (KM)
0	Honda	White	4.0	35431.0
1	BMW	Blue	5.0	192714.0
2	Honda	White	4.0	84714.0
3	Toyota	White	4.0	154365.0
4	Nissan	Blue	3.0	181577.0
..	...	...	...	...
945	Toyota	Black	4.0	35820.0
946	missing	White	3.0	155144.0

```
947   Nissan   Blue   4.0       66604.0
948    Honda  White   4.0      215883.0
949   Toyota   Blue   4.0      248360.0

[950 rows x 4 columns]
```

[52]:
```
car_sales_filled.isna().sum()
```

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

[53]:
```
# Split into X & y don't change
X = car_sales_filled
```

[54]:
```
# Lets try and convert our data to numbers

from sklearn.preprocessing import OneHotEncoder # Turn the categories into␣
 ↪numbers
from sklearn.compose import ColumnTransformer

categorical_features = ["Make", "Colour","Doors"] # Doors because there a only 3␣
 ↪typen (5,4 and 3) it is also categorical
one_hot = OneHotEncoder(sparse_output=False) # getting a array not a sparse␣
 ↪Matrix
transformer = ColumnTransformer([("one_hot",
                                  one_hot,
                                  categorical_features)],
                                 remainder = "passthrough") # dont change other␣
 ↪columns
transformed_X = transformer.fit_transform(X)

transformed_X
```

[54]:
```
array([[0.0, 1.0, 0.0, ..., 1.0, 0.0, 35431.0],
       [1.0, 0.0, 0.0, ..., 0.0, 1.0, 192714.0],
       [0.0, 1.0, 0.0, ..., 1.0, 0.0, 84714.0],
       ...,
       [0.0, 0.0, 1.0, ..., 1.0, 0.0, 66604.0],
       [0.0, 1.0, 0.0, ..., 1.0, 0.0, 215883.0],
       [0.0, 0.0, 0.0, ..., 1.0, 0.0, 248360.0]],
      shape=(950, 15), dtype=object)
```

[55]:
```
pd.DataFrame(transformed_X)
```

[55]:
```
       0    1    2    3    4    5    6    7    8    9    10   11   12   13  \
0    0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0
1    1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
2    0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0
3    0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0
4    0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
..   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
945  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
946  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0
947  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
948  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0
949  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0

            14
0     35431.0
1    192714.0
2     84714.0
3    154365.0
4    181577.0
..        ...
945   35820.0
946  155144.0
947   66604.0
948  215883.0
949  248360.0

[950 rows x 15 columns]
```

[56]:
```
# Now we've got our data as numbers and filled ( no missing data)
#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_size=0.2)

model = RandomForestRegressor()
model.fit(X_train,y_train)
model.score(X_test,y_test) # worse score maybe wrong model (or we need more␣
 ↪samples)
```

[56]:
```
0.21990196728583944
```

[57]:
```
whats_were_covering
```

[57]:
```
['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']
```

3 2. choose the right estimator/algorithm for our problems

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

3.0.1 2.1 picking a ML model for regression problem

Let's use the california housing toy dataset

[58]:
```
from sklearn.datasets import fetch_california_housing
```

[59]:
```
df = fetch_california_housing()
```

[60]:
```
df
```

[60]:
```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,
2.55555556,
           37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, ...,    2.10984183,
           37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, ...,    2.80225989,
           37.85      , -122.24      ],
        ...,
        [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351 ,
           39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, ...,    2.12320917,
           39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, ...,    2.61698113,
           39.37      , -121.24      ]], shape=(20640, 8)),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894],
shape=(20640,)),
 '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 attributes
and the target\n\n:Attribute Information:\n    - MedInc        median income in
block group\n    - HouseAge      median house age in block group\n    - AveRooms
average number of rooms per household\n    - AveBedrms     average number of
bedrooms per household\n    - Population    block group population\n    -
AveOccup      average number of household 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/~ltorgo/Regression/cal_housing.html\n\nThe
target variable is the median house value for California districts,\nexpressed
in hundreds of thousands of dollars ($100,000).\n\nThis dataset was derived from
the 1990 U.S. census, using one row per census\nblock group. A block group is
the smallest geographical unit for which the U.S.\nCensus Bureau publishes
sample data (a block group typically has a population\nof 600 to 3,000
people).\n\nA household is a group of people residing within a home. Since the
average\nnumber of rooms and bedrooms in this dataset are provided per
household, these\ncolumns may take surprisingly large values for block groups
with few households\nand many empty houses, such as vacation resorts.\n\nIt can
be downloaded/loaded using
the\n:func:`sklearn.datasets.fetch_california_housing` function.\n\n.. rubric::
References\n\n- Pace, R. Kelley and Ronald Barry, Sparse Spatial
Autoregressions,\n  Statistics and Probability Letters, 33:291-297, 1997.\n'}
```

[61]:
```
housing_df = pd.DataFrame(df["data"])
housing_df
```

[61]:

	0	1	2	3	4	5	6	7
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

[20640 rows x 8 columns]

[62]:
```
housing_df = pd.DataFrame(df["data"], columns = df["feature_names"])
housing_df
```

[62]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25
...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24

[20640 rows x 8 columns]

[63]:
```
housing_df["target"]= df ["target"]
housing_df
```

[63]:

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

```
       Longitude  target
0         -122.23   4.526
1         -122.22   3.585
2         -122.24   3.521
3         -122.25   3.413
4         -122.25   3.422
...           ...     ...
20635     -121.09   0.781
20636     -121.21   0.771
20637     -121.22   0.923
20638     -121.32   0.847
20639     -121.24   0.894

[20640 rows x 9 columns]
```

[64]:
```
housing_df = housing_df.drop("target", axis = 1)
```

[65]:
```
#import algorithm
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
#setup random seed
np.random.seed(42)

# Create the data
X = housing_df
y = df["target"]

# Split into train and test data

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

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

model.score(X_test,y_test)
```

[65]:
```
0.5911128551684496
```

[66]:
```
# Varianz beschreibt die vertikale Streuung der Zielwerte um einen Referenzwert.
#
# - Gesamtvarianz (SST):
#   Abstand der echten Werte y zum Mittelwert y_mean.
#   Sie zeigt, wie stark die Daten insgesamt streuen,
#   also wie viel "Information" bzw. Unordnung in den Zielwerten steckt.
#
# - Nicht erklärte Varianz (SSE):
```

```
#   Abstand der echten Werte y zu den Modellvorhersagen y_pred.
#   Sie zeigt, wie viel dieser ursprünglichen Streuung nach dem Modell
#   noch übrig bleibt und vom Modell nicht erklärt werden kann.
#
# - Zusammenhang zu R²:
#   R² = 1 - (nicht erklärte Varianz / Gesamtvarianz)
#   → R² misst, welcher Anteil der ursprünglichen Streuung
#     durch das Modell reduziert bzw. erklärt wurde.
#
# Grafische Vorstellung:
#   • Abstand Punkt → Mittelwert  = Gesamtvarianz
#   • Abstand Punkt → Modelllinie = nicht erklärte Varianz
```

What if Ridge didn't work or the score didn't fit our needs

Try a Ensemble Model

[67]:
```
# Ensemble-Methoden kombinieren mehrere einzelne Modelle (Base Learners)
# zu einem gemeinsamen Gesamtmodell, um stabilere und genauere Vorhersagen
# zu erzielen als mit einem einzelnen Modell.
#
# Motivation:
# Einzelmodelle leiden häufig unter hohem Bias (Underfitting) oder
# hoher Varianz (Overfitting). Ensembles wirken diesem Problem entgegen.
#
# Grundprinzip:
# - Mehrere Modelle werden trainiert
# - Jedes Modell macht leicht unterschiedliche Fehler
# - Durch Mittelung oder Gewichtung heben sich Fehler teilweise auf
#
# Hauptarten von Ensemble-Methoden:
#
# 1) Bagging (z. B. Random Forest):
#    - Modelle werden parallel auf zufälligen Stichproben trainiert
#    - Reduziert hauptsächlich die Varianz
#    - Besonders effektiv bei instabilen Modellen (z. B. Entscheidungsbäume)
#
# 2) Boosting (z. B. Gradient Boosting, XGBoost):
#    - Modelle werden sequenziell trainiert
#    - Jedes neue Modell fokussiert sich auf die Fehler der vorherigen
#    - Reduziert hauptsächlich den Bias
#
# 3) Stacking:
#    - Mehrere unterschiedliche Modelle werden kombiniert
#    - Ein Meta-Modell lernt, wie die Vorhersagen optimal gemischt werden
#
# Zusammenhang zu Bias-Varianz-Tradeoff:
# Ensemble-Methoden reduzieren Varianz, Bias oder beides und senken dadurch
```

```
# die nicht erklärte Varianz, was zu besserer Generalisierung führt.
```

[68]:
```
# import the RandomForestRegressor model class
from sklearn.ensemble import RandomForestRegressor

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

# model
rfg = RandomForestRegressor()

# data
# Create the data
X = housing_df
y = df["target"]

# Split into train and test data

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

rfg.fit(X_train, y_train)

rfg.score(X_test,y_test)
```

[68]:
```
0.806410393875331
```

3.1 2.2 Picking a ML model for a classification Problem

[69]:
```
heart_disease = pd.read_csv("data/heart-disease.csv")
heart_disease.head()
```

[69]:

	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

[70]:
```
len(heart_disease)
```

[70]: 303

[71]:
```
# import the LinearSVC
from sklearn.svm import LinearSVC

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

# getting data ready
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)

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

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

[71]: 0.881578947368421

[72]:
```
#Works good! LinearSVC
```

[73]:
```
# import the RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

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

# getting data ready
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)

#Instantiate RandomForest
clf = RandomForestClassifier()
#fit the model to the data (training)
clf.fit(X_train,y_train)

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

[73]:
```
0.8289473684210527
```

[74]:
```
whats_were_covering
```

[74]:
```
['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']
```

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

3.2.1 3.1 Fitting a model to the data

X = features, features variables, data

y = labels , targets, target variables

[75]:
```
# import the RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

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

# getting data ready
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)

#Instantiate RandomForest
clf = RandomForestClassifier()
#fit the model to the data (training)
clf.fit(X_train,y_train)

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

[75]:
```
0.8289473684210527
```

[76]:
```
X.head()
```

[76]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	

```
2   41    0   1       130   204    0       0       172      0      1.4      2
3   56    1   1       120   236    0       1       178      0      0.8      2
4   57    0   0       120   354    0       1       163      1      0.6      2

   ca  thal
0   0     1
1   0     2
2   0     2
3   0     2
4   0     2
```

[77]:
```
y.head()
```

[77]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

[78]:
```
# the fit method find patterns when the target is 1 and when target is 0
```

3.2.2 3.2 Make predictions using a ML model

2 ways to make predictions

1. predict()
2. predict_proba()

[79]:
```
# Use a trained model to make predictions with predict()
```

[80]:
```
#clf.predict(np.array[1,6,7,8,5])# this doesnt work
```

[81]:
```
clf.predict(X_test)
```

[81]:
```
array([0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 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, 1, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1])
```

[82]:
```
np.array(y_test)
```

[82]:
```
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, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1])
```

[83]:
```
# compare predictions to truth labels to eval. the model
y_preds = clf.predict(X_test)
np.mean(y_preds == y_test) # this is what the method score does the accuracy
```

[83]:
```
np.float64(0.8289473684210527)
```

[84]:
```
# another way for the accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_preds,y_test)
```

[84]:
```
0.8289473684210527
```

Make predictions with predict_proba()

[85]:
```
# predict_proba returns probabilities of a classification label
```

[86]:
```
clf.predict_proba(X_test[:5])
```

[86]:
```
array([[0.91, 0.09],
       [0.45, 0.55],
       [0.51, 0.49],
       [0.85, 0.15],
       [0.25, 0.75]])
```

[87]:
```
# Lets predict on the same data
clf.predict(X_test[:5])
```

[87]:
```
array([0, 1, 0, 0, 1])
```

predict () can also be used for regression models

[88]:
```
housing_df.head()
```

[88]:

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

[89]:
```
from sklearn.ensemble import RandomForestRegressor
```

```
np.random.seed(42)

X = housing_df
y = y = pd.Series(fetch_california_housing()["target"])

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

model = RandomForestRegressor()

model.fit(X_train, y_train)

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

[90]:
```
y_preds
```

[90]:
```
array([0.4939   , 0.75494  , 4.9285964, ..., 4.8363785, 0.71782  ,
       1.67781  ], shape=(4128,))
```

[91]:
```
np.array(y_test[:10])
```

[91]:
```
array([0.477  , 0.458  , 5.00001, 2.186  , 2.78   , 1.587  , 1.982  ,
       1.575  , 3.4    , 4.466  ])
```

[92]:
```
# COmpare the predictions to the truth
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,y_preds)

# Mean Absolute Error (MAE):
# Der MAE gibt an, um wie viele Einheiten ein Modell im Durchschnitt␣
 ↪danebenliegt.
# Dazu wird für jede Vorhersage die absolute Abweichung zum echten Wert berechnet
# (Minuszeichen werden ignoriert) und anschließend der Mittelwert dieser␣
 ↪Abweichungen gebildet.
# Der MAE hat dieselbe Einheit wie die Zielvariable und ist leicht␣
 ↪interpretierbar.
```

[92]:
```
0.32656738464147306
```

[93]:
```
whats_were_covering
```

[93]:
```
['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']
```

3.3 4. Evaluating a model

three ways to evaluate Scikit-Learn models/estimators:

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

3.3.1 4.1 Evaluating a model with the score method

[94]:
```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

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

#Create train test
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.2)

#Create Model
rfc = RandomForestClassifier()

# Fit model
rfc.fit(X_train, y_train)
```

[94]:
```
RandomForestClassifier()
```

[95]:
```
rfc.score(X_train,y_train) # the highest vakue for the score method is 1.0 and␣
 ↪lowest is 0.0 the return value is the accuracy
```

[95]:
```
1.0
```

[96]:
```
rfc.score(X_test,y_test)
```

[96]:
```
0.8524590163934426
```

Let's use the data on a regression model

[97]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

# Create X & y
```

```
X = pd.DataFrame(fetch_california_housing()["data"], columns =␣
 ↪fetch_california_housing()["feature_names"])
y = fetch_california_housing()["target"]

#Create train test
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.2)

#Create Model
rfc = RandomForestRegressor()

# Fit model
rfc.fit(X_train, y_train)
```

[97]:
```
RandomForestRegressor()
```

[98]:
```
rfc.score(X_test,y_test) # the default score (evaluation metric is r_squared for␣
 ↪regression algorithms highest = 1.0, lowest 0.0
```

[98]:
```
0.806652667101436
```

3.3.2 4.2 Evaluating a model using the scoring parameter

[99]:
```
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

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

#Create train test
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.2)

#Create Model
rfc = RandomForestClassifier()

# Fit model
rfc.fit(X_train, y_train);
```

[100]:
```
rfc.score(X_test,y_test)
```

[100]:
```
0.8524590163934426
```

[101]:
```
cross_val_score(clf,X,y, cv = 5) # returns a array
```

[101]:
```
array([0.81967213, 0.86885246, 0.81967213, 0.78333333, 0.76666667])
```

[102]:
```
np.random.seed(42)
#SIngle training and test split score
clf_single_score = clf.score(X_test,y_test)

# take the mean of 5fold cv score
clf_cross_val_score = np.mean (cross_val_score(clf,X,y))

#compare the two
clf_single_score, clf_cross_val_score
```

[102]:
```
(0.8360655737704918, np.float64(0.8248087431693989))
```

[103]:
```
# Scoring parameter set to None by default
cross_val_score(clf,X,y,scoring = None)
```

[103]:
```
array([0.78688525, 0.86885246, 0.80327869, 0.78333333, 0.76666667])
```

[104]:
```
# Cross-Validation wird verwendet, um die tatsächliche Leistungsfähigkeit eines␣
 ↪Modells
# zuverlässig zu bewerten. Im Gegensatz zu einem einzelnen Train/Test-Split␣
 ↪mittelt sie
# die Performance über mehrere Datenaufteilungen und reduziert damit␣
 ↪Zufallseinflüsse.
# Sie wird standardmäßig bei Modellvergleich, Hyperparameter-Tuning und in␣
 ↪professionellen
# ML-Projekten eingesetzt, während ein Single Split nur für schnelle Tests␣
 ↪geeignet ist.
```

3.3.3 4.2 Classification model evaluation metrics

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

3.3.4 Accuracy

[105]:
```
heart_disease.head()
```

[105]:

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

```
   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

[106]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

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

clf = RandomForestClassifier()

cross_val_score(clf,X,y,cv =5) # return value => mean accuracy
```

[106]:
```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

[107]:
```
cross_val_score = cross_val_score(clf,X,y,cv =5)
np.mean(cross_val_score)
```

[107]:
```
np.float64(0.8018032786885245)
```

[108]:
```
print(f"Heart Disease Classifier Cross-Validated Accuracy: {np.
↪mean(cross_val_score)*100}")
```

```
Heart Disease Classifier Cross-Validated Accuracy: 80.18032786885246
```

3.3.5 Area under the reciever operating charateristic curve (AUC/ROC)

- Area under curve (AUC)
- ROC

ROC curves are a comparison of a model's true positive rate (tpr) versus a model false positive rate (fpr)

- True positive = model predicts 1 when truth is 1
- False positive = model predicts 1 when truth is 0
- True negative = model predicts 0 when truth is 0
- False negative = model predicts 0 when truth is 1

[109]:
```
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.2)

clf.fit(X_train,y_train)
```

[109]:
```
RandomForestClassifier()
```

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

# Make Predictions with probalilities
y_probs = clf.predict_proba(X_test)
y_probs[:10]
```

[110]:
```
array([[0.11, 0.89],
       [0.53, 0.47],
       [0.09, 0.91],
       [0.39, 0.61],
       [0.23, 0.77],
       [0.01, 0.99],
       [0.61, 0.39],
       [0.45, 0.55],
       [0.11, 0.89],
       [0.87, 0.13]])
```

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

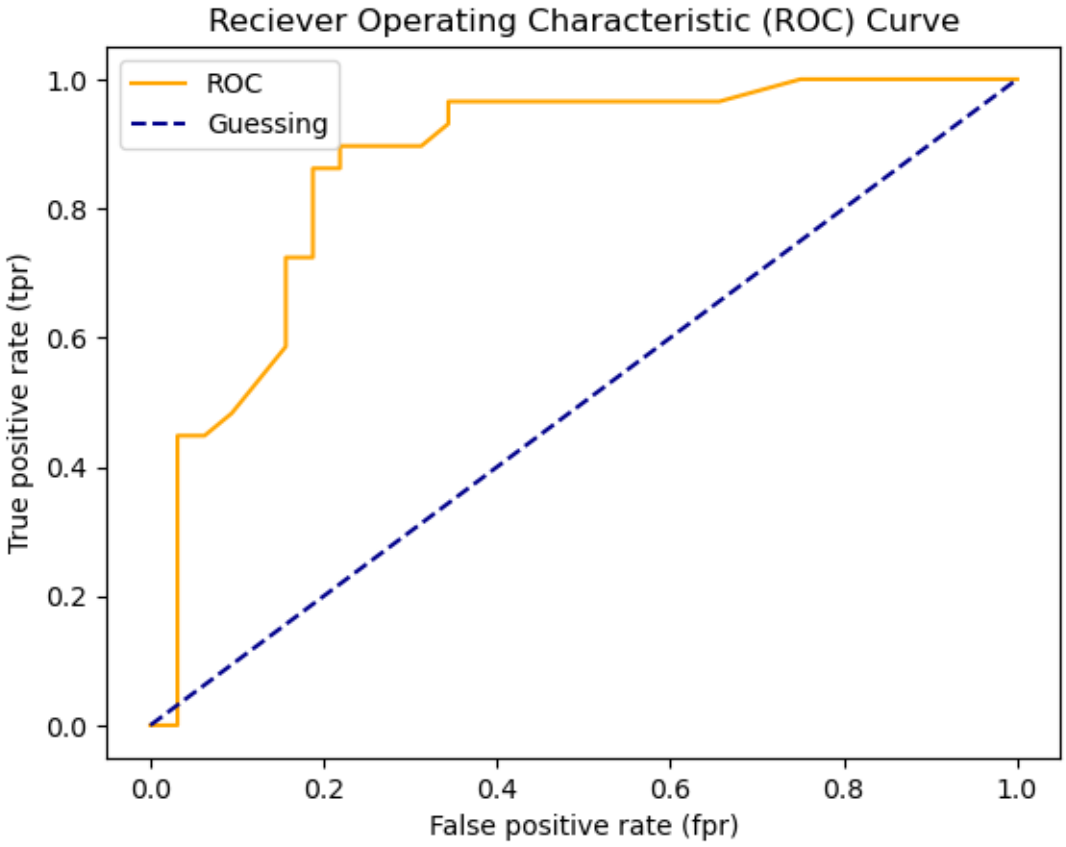
[111]:
```
array([0.89, 0.47, 0.91, 0.61, 0.77, 0.99, 0.39, 0.55, 0.89, 0.13])
```

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

# Check the false positive rates
fpr
```

[112]:
```
array([0.     , 0.03125, 0.03125, 0.03125, 0.03125, 0.03125, 0.0625 ,
       0.09375, 0.15625, 0.15625, 0.1875 , 0.1875 , 0.21875, 0.21875,
       0.3125 , 0.34375, 0.34375, 0.40625, 0.59375, 0.65625, 0.75   ,
       0.9375 , 1.     ])
```

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

def plot_roc_curve (fpr, tpr):
    """
    Plots a ROC Curve give the fpr and tpr of a model
    """
    #plot the 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")

    #Customizing the plot
    plt.xlabel("False positive rate (fpr)")
    plt.ylabel("True positive rate (tpr)")
```

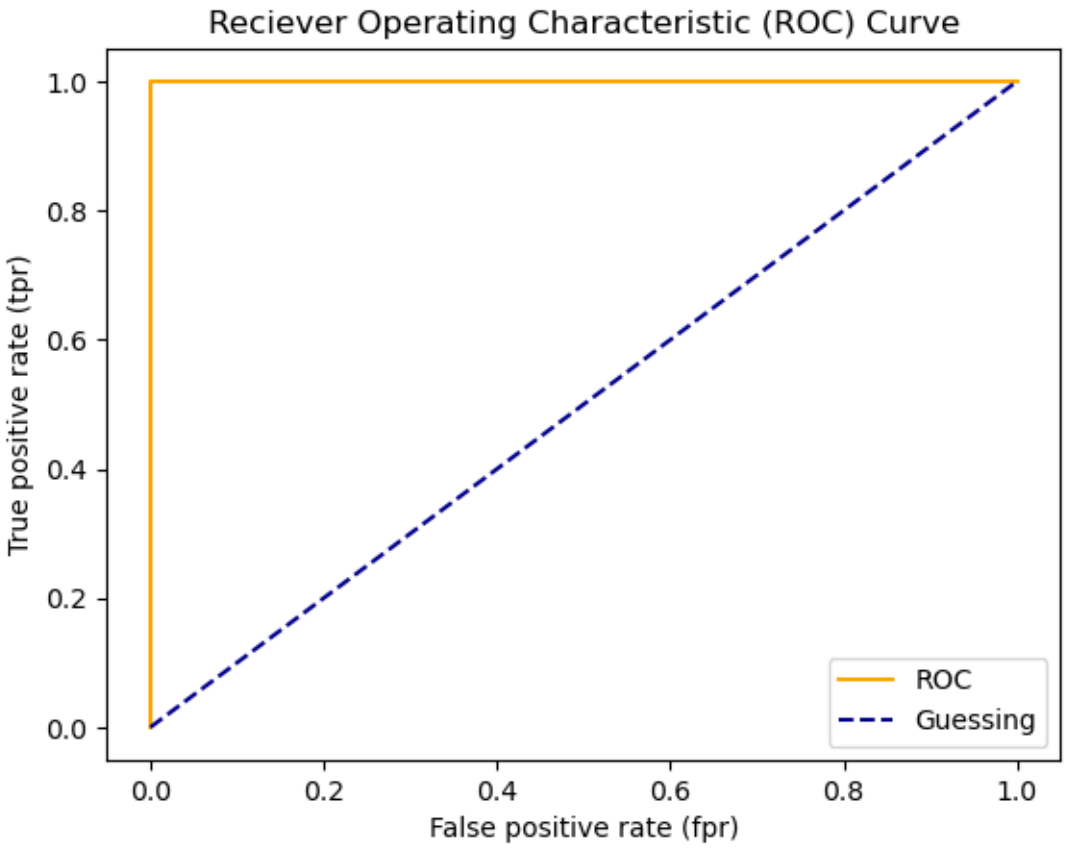
```
    plt.title ("Reciever Operating Characteristic (ROC) Curve")
    plt.legend()
    plt.show()

plot_roc_curve(fpr,tpr)
```



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

roc_auc_score(y_test,y_probs_positive)
```

[114]:
```
0.8679956896551724
```

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



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

[116]: 1.0

[117]:
```
# ================================================================================
# ROC & AUC - Kurzfassung
# ================================================================================
#
# ROC (Receiver Operating Characteristic):
# Zeigt, wie sich ein Klassifikationsmodell bei allen möglichen Schwellenwerten
# verhält. Dargestellt wird der Zusammenhang zwischen:
# - True Positive Rate  (wie viele echte Positive erkannt werden)
# - False Positive Rate (wie viele Negative fälschlich als positiv erkannt␣
 ↪werden)
#
# AUC (Area Under the Curve):
# Eine einzelne Zahl, die die ROC-Kurve zusammenfasst.
# Sie misst die Trennfähigkeit des Modells.
#
```

```
# Intuition:
# AUC = Wahrscheinlichkeit, dass ein zufällig positives Beispiel
# einen höheren Score erhält als ein zufällig negatives.
#
# Interpretation:
# AUC = 1.0 -> perfekte Trennung
# AUC = 0.5 -> Zufall
#
# Merksatz:
# ROC/AUC bewerten nicht eine konkrete Entscheidung,
# sondern wie gut das Modell positive von negativen Fällen trennt.
# ========================================================================
```

3.3.6 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

[118]:
```
from sklearn.metrics import confusion_matrix

y_preds = clf.predict(X_test)

confusion_matrix(y_test,y_preds)
```

[118]:
```
array([[24,  8],
       [ 3, 26]])
```

[119]:
```
# Visualize confusion matrix with pd.crosstab

pd.crosstab(y_test,y_preds, rownames = ["Actual Labels"], colnames = ["Predicted␣
 ↪Labels"])
```

[119]:
```
Predicted Labels   0   1
Actual Labels
0                 24   8
1                  3  26
```

[120]:
```
24+8+3+26
```

[120]:
```
61
```

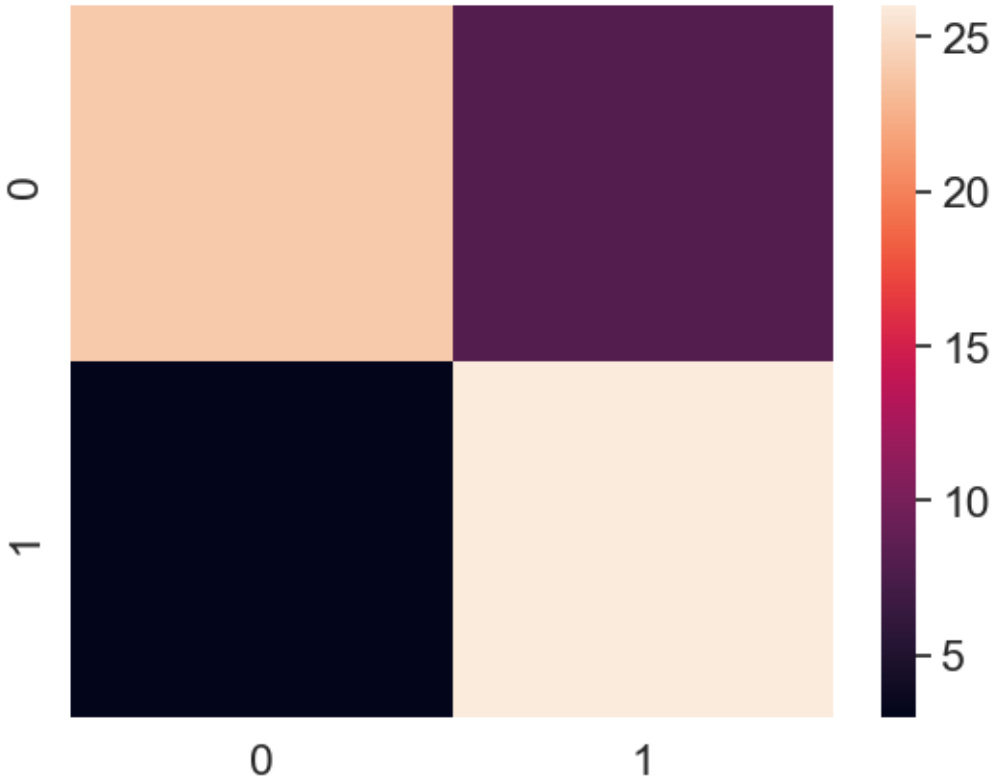
[121]:
```
len(y_preds)
```

[121]:
```
61
```

[122]:
```
import sys  # How to install a new package in JN
#!conda install --yes --prefix {sys.prefix} seaborn  already installed in the␣
 ↪environment
```

[123]:
```
# Make our Confusion matrix more visual with seaborn 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 is using Seaborn
sns.heatmap(conf_mat)
```

[123]: <Axes: >



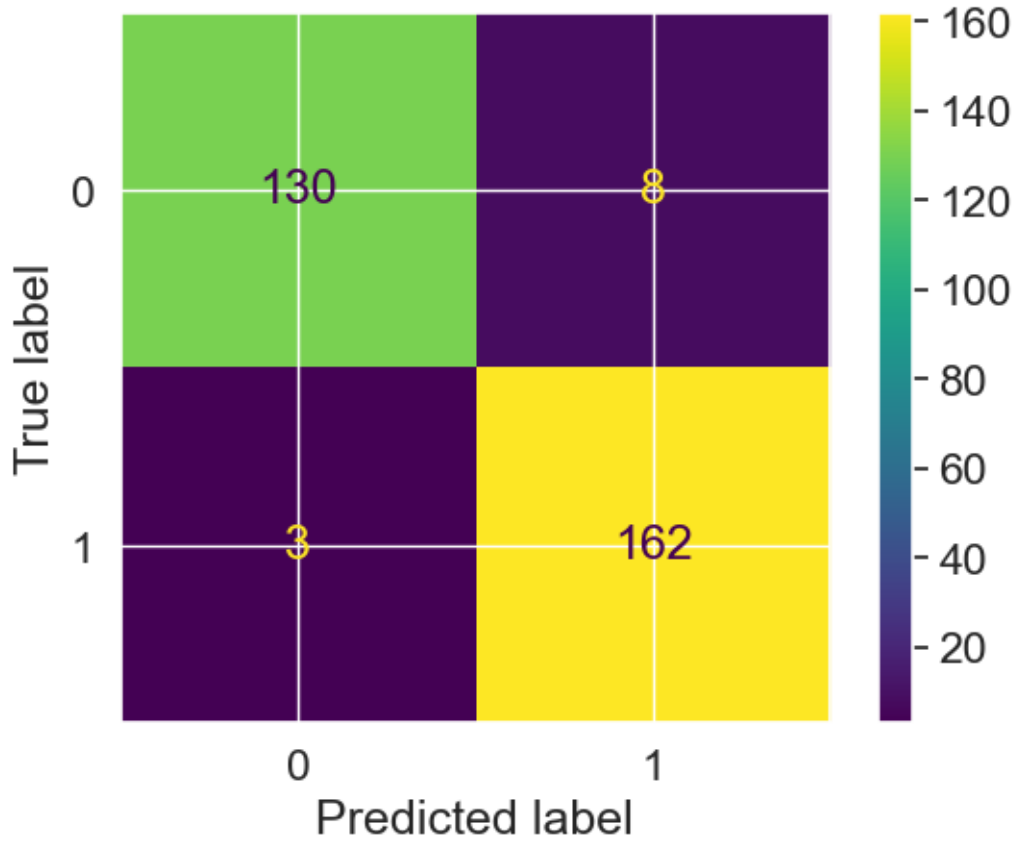
3.3.7 Creating a confusion matrix using scikit Learn

[124]:
```
import sklearn
sklearn.__version__
```

[124]: '1.8.0'

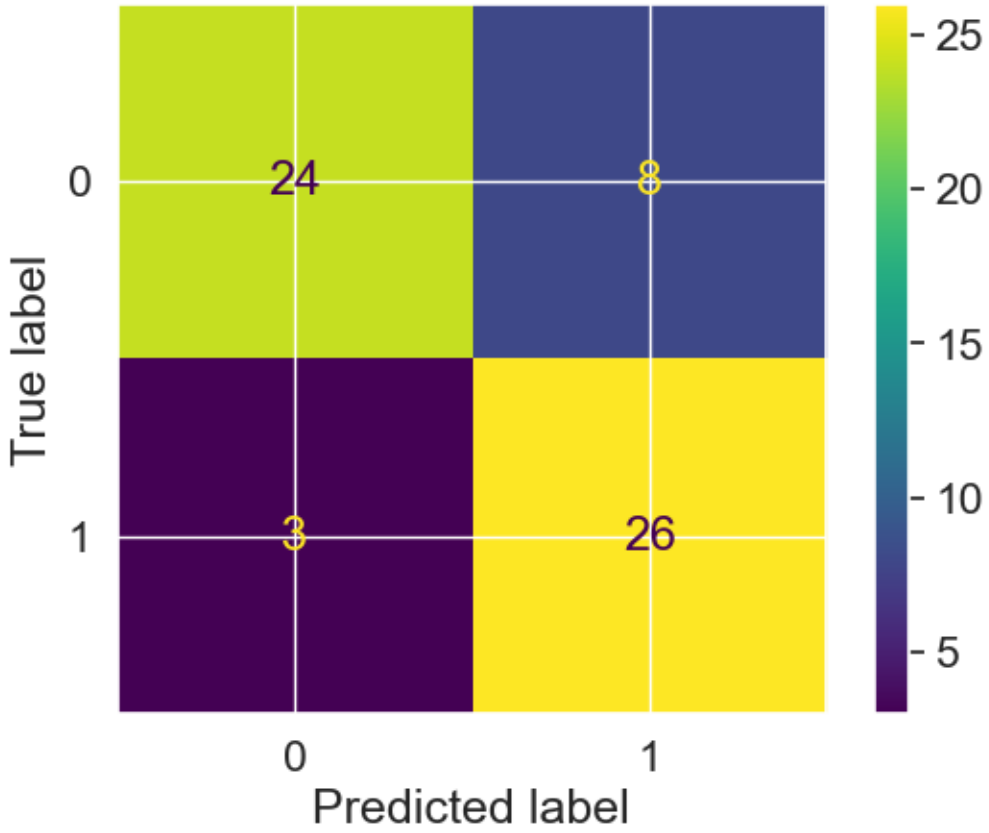
[125]:
```
from sklearn.metrics import ConfusionMatrixDisplay

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

[125]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ddb0f5e510>



[126]:
```
ConfusionMatrixDisplay.from_predictions(y_true = y_test, y_pred= y_preds)
```

[126]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ddda07afd0>



3.3.8 Classification Report

[127]:
```
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

           0       0.89      0.75      0.81        32
           1       0.76      0.90      0.83        29

    accuracy                           0.82        61
   macro avg       0.83      0.82      0.82        61
weighted avg       0.83      0.82      0.82        61
```

[128]:
```
# where precision and recall become vlauable

disease_true = np.zeros(10000)
disease_true[0] = 1 # only one possitive
```

```python
disease_preds = np.zeros(10000) # model predictions

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

```
C:\Users\RomaM\Documents\ZeroToMastery\Machine_Learning\scikit_learn_introductio
n\env\Lib\site-packages\sklearn\metrics\_classification.py:1833:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\RomaM\Documents\ZeroToMastery\Machine_Learning\scikit_learn_introductio
n\env\Lib\site-packages\sklearn\metrics\_classification.py:1833:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\RomaM\Documents\ZeroToMastery\Machine_Learning\scikit_learn_introductio
n\env\Lib\site-packages\sklearn\metrics\_classification.py:1833:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

[128]:

	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

- Accuray 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 higer precision
- If false negative predictions are worse than false positives, aim for higher recall.
- f1-score is a combination of precision and recall.

3.3.9 4.2.2 Regression model evaluation metrics

Model evaluation metrics documentation - https://scikit-learn.org/0.15/modules/model_evaluation.html#regressic metrics

The ones we are going to cover are:

1. R^2 or coefficient of determination
2. Mean Absolute Error MAE
3. Mean squared Error MSE

[129]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

X = housing_df
y = df["target"]

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

model = RandomForestRegressor()

model.fit(X_train,y_train)
```

[129]:
```
RandomForestRegressor()
```

[130]:
```
model.score(X_test,y_test)
```

[130]:
```
0.806652667101436
```

[131]:
```
y_test
```

[131]:
```
array([0.477  , 0.458  , 5.00001, ..., 5.00001, 0.723  , 1.515  ],
      shape=(4128,))
```

[132]:
```
y_test.mean()
```

[132]:
```
np.float64(2.0550030959302323)
```

[133]:
```
from sklearn.metrics import r2_score

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

[134]:
```
y_test_mean[:10]
```

[134]:
```
array([2.0550031, 2.0550031, 2.0550031, 2.0550031, 2.0550031, 2.0550031,
       2.0550031, 2.0550031, 2.0550031, 2.0550031])
```

[135]:
```
r2_score(y_true = y_test,
         y_pred = y_test_mean)
```

[135]:
```
0.0
```

[136]:
```
r2_score(y_true = y_test,
         y_pred = y_test)
```

[136]:
```
1.0
```

3.3.10 Mean absolute error

MAE is the average of the absolute diffrences between predictions and actual values It gives you an idea of how wrong your model predictions are

[137]:
```
# MAE
from sklearn.metrics import mean_absolute_error

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

[137]:
```
0.32656738464147306
```

[138]:
```
y_preds
```

[138]:
```
array([0.4939   , 0.75494  , 4.9285964, ..., 4.8363785, 0.71782  ,
       1.67781  ], shape=(4128,))
```

[139]:
```
y_test
```

[139]:
```
array([0.477  , 0.458  , 5.00001, ..., 5.00001, 0.723  , 1.515  ],
      shape=(4128,))
```

[140]:
```
df2 = pd.DataFrame( data = {"actual values": y_test, "predicted values":␣
↪y_preds})
df2["diffrences"]= df2["predicted values"] - df2["actual values"]
df2.head(10)
```

[140]:

	actual values	predicted values	diffrences
0	0.47700	0.493900	0.016900
1	0.45800	0.754940	0.296940
2	5.00001	4.928596	-0.071414
3	2.18600	2.540240	0.354240
4	2.78000	2.331760	-0.448240
5	1.58700	1.660220	0.073220
6	1.98200	2.343100	0.361100
7	1.57500	1.663110	0.088110
8	3.40000	2.474890	-0.925110
9	4.46600	4.834478	0.368478

[141]:
```
# MAE using formulas and diffrences
np.abs(df2["diffrences"]).mean() # MAE
```

[141]:
```
np.float64(0.32656738464147306)
```

3.3.11 Mean squared error

MSE is the mean of the square of the errors between actual and predictied values

[142]:
```
# 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
```

[142]:
```
0.25336408094921037
```

[143]:
```
df2["squared_diffrences"] = np.square(df2["diffrences"])
df2.head(10)
```

[143]:

	actual values	predicted values	diffrences	squared_diffrences
0	0.47700	0.493900	0.016900	0.000286
1	0.45800	0.754940	0.296940	0.088173
2	5.00001	4.928596	-0.071414	0.005100
3	2.18600	2.540240	0.354240	0.125486
4	2.78000	2.331760	-0.448240	0.200919
5	1.58700	1.660220	0.073220	0.005361
6	1.98200	2.343100	0.361100	0.130393
7	1.57500	1.663110	0.088110	0.007763
8	3.40000	2.474890	-0.925110	0.855829
9	4.46600	4.834478	0.368478	0.135776

[144]:
```
# Calculate MSE by hand
squared = np.square(df2["diffrences"])
squared.mean()
```

[144]:
```
np.float64(0.25336408094921037)
```

[145]:
```
df_large_error = df2.copy()
df_large_error.loc[0, "squared_diffrences"] = 16
```

[146]:
```
df_large_error.head()
```

[146]:

	actual values	predicted values	diffrences	squared_diffrences
0	0.47700	0.493900	0.016900	16.000000
1	0.45800	0.754940	0.296940	0.088173
2	5.00001	4.928596	-0.071414	0.005100
3	2.18600	2.540240	0.354240	0.125486
4	2.78000	2.331760	-0.448240	0.200919

[147]:
```
# Calculate MSE with large error
df_large_error["squared_diffrences"].mean()
```

[147]:
```
np.float64(0.25723998075298943)
```

[148]:
```
df_large_error.loc[1:100,"squared_diffrences"] = 20
```

[149]:
```
df_large_error.head()
```

[149]:

	actual values	predicted values	diffrences	squared_diffrences
0	0.47700	0.493900	0.016900	16.0
1	0.45800	0.754940	0.296940	20.0
2	5.00001	4.928596	-0.071414	20.0
3	2.18600	2.540240	0.354240	20.0
4	2.78000	2.331760	-0.448240	20.0

[150]:
```
# Calculate MSE with large error
df_large_error["squared_diffrences"].mean()
```

[150]:
```
np.float64(0.738094852122935)
```

3.3.12 4.2.3 Finally using the scoring parameter

[151]:
```
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()
```

[152]:
```
np.random.seed(42)

# Cross Validation accuracy

cv_acc = cross_val_score(clf, X,y, cv = 5, scoring = None) # if scoring = None, estimators default scoring evaluation metric is used (accuracy for classification problem)
cv_acc
```

[152]:
```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

[153]:
```
# Cross-validated accuracy
print(f"The cross-calidated accuracy is: {np.mean(cv_acc)*100:2f}%")
```

```
The cross-calidated accuracy is: 82.480874%
```

[154]:
```
np.random.seed(42)

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

[154]:
```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

[155]:
```
# Cross-validated accuracy
print(f"The cross-calidated accuracy is: {np.mean(cv_acc)*100:2f}%")
```

```
The cross-calidated accuracy is: 82.480874%
```

[156]:
```
#Precision
np.random.seed(42)
cv_precision = cross_val_score(clf, X,y, cv = 5, scoring = "precision")
cv_precision
```

[156]:
```
array([0.82352941, 0.93548387, 0.84848485, 0.79411765, 0.76315789])
```

[157]:
```
# Cross-validated precision
print(f"The cross-calidated precision is: {np.mean(cv_precision)*100:2f}")
```

```
The cross-calidated precision is: 83.295473
```

[158]:
```
# Recall
np.random.seed(42)
cv_recall = cross_val_score(clf, X,y, cv = 5, scoring = "recall")
cv_recall
```

[158]:
```
array([0.84848485, 0.87878788, 0.84848485, 0.81818182, 0.87878788])
```

[159]:
```
# Cross-validated recall
print(f"The cross-calidated recall is: {np.mean(cv_recall)*100:2f}")
```

```
The cross-calidated recall is: 85.454545
```

Let's see the scoring parameter bein used for a regression problem. . . .

[160]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

X = housing_df
y = df["target"]

model = RandomForestRegressor()
```

[161]:
```
np.random.seed(42)

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

[161]:
```
np.float64(0.6545660727379677)
```

[162]:
```
# MAE
cv_mae = cross_val_score(model,X,y, cv = 3, scoring = "neg_mean_absolute_error")
np.mean(cv_mae)
```

[162]:
```
np.float64(-0.48487437131782957)
```

[163]:
```
cv_mae
```

[163]:
```
array([-0.52261147, -0.42493192, -0.50707973])
```

[164]:
```
#MSE
cv_mse = cross_val_score(model,X,y, cv = 3, scoring = "neg_mean_squared_error")
np.mean(cv_mse)
```

[164]:
```
np.float64(-0.4612145271352917)
```

[165]:
```
cv_mse
```

[165]:
```
array([-0.51324679, -0.32921673, -0.54118007])
```

3.3.13 4.3 Using diffrent evaluation metrics as Scikit-Learn functions

The 3rd way to evaluate scikit learn machine learning models is to using sklearn.metrics module

[166]:
```
from sklearn.metrics import accuracy_score, precision_score,recall_score,f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

np.random.seed(42)

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

# Split

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

# Create model
clf = RandomForestClassifier()

# Fit model
clf.fit(X_train,y_train)

y_preds = clf.predict(X_test)

accuracy_score(y_test,y_preds), precision_score(y_test,y_preds),␣
↪recall_score(y_test,y_preds), f1_score(y_test,y_preds)
```

[166]:
```
(0.8524590163934426, 0.8484848484848485, 0.875, 0.8615384615384616)
```

[167]:
```
clf.score(X_test,y_test)
```

[167]:
```
0.8524590163934426
```

[168]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

np.random.seed(42)

# Create X and y
X = housing_df
y = df["target"]

# Split

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

# Create model
clf = RandomForestRegressor()

# Fit model
clf.fit(X_train,y_train)

y_preds = clf.predict(X_test)

r2_score(y_test,y_preds), mean_absolute_error(y_test,y_preds),␣
↪mean_squared_error(y_test,y_preds)
```

[168]:
```
(0.806652667101436, 0.32656738464147306, 0.25336408094921037)
```

[169]:
```
clf.score(X_test,y_test)
```

[169]:
```
0.806652667101436
```

[170]:
```
whats_were_covering
```

[170]:
```
['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']
```

3.4 5. Improve a model

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

From a data perspective: * Could we collect more data? (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 it's ability to find patterns.

Three ways to adjust hyperparameters: 1. By hand 2. Randomly with RandomSearchCV 3. Exhaustively with GridSearchCV

[171]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
```

[172]:
```
#how to get the Hyperparameters

clf.get_params()
```

[172]:
```
{'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}
```

3.4.1 5.1 Tuning Hyperparameter by hand

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

[173]:
```
clf.get_params() # look scikit learn doc they suggest
```

[173]:
```
{'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}
```

We're going try and adjust:

- max_depth
- max_features
- min_samples_leaf
- min_samples_split
- n_estimators

[174]:
```
def evaluate_preds(y_true, y_preds):
    """
    Performs evaluation comparison on y_true labels vs. y_pred labels on a␣
↪classification model.
    """
    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"Accuracy: {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
```

[175]:
```
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))
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)
```

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

[176]:
```
np.random.seed(42)

# create a second classifier with diffrent hyperparameters
clf_2 = RandomForestClassifier(n_estimators=1000)
clf_2.fit(X_train,y_train)
# Make  predictions with different HP
y_preds_2 = clf_2.predict(X_valid)

# Evaluate the 2nd classifier on validation set
metrics = evaluate_preds(y_valid,y_preds_2)
```

```
Accuracy: 82.22%
```

```
Precision: 81.48
Recall: 88.00
F1 score: 84.62
```

[177]:
```
np.random.seed(42)

# create a third classifier with diffrent hyperparameters
clf_3 = RandomForestClassifier( n_estimators=500,
    max_features="sqrt",
    min_samples_leaf=5,
    min_samples_split=10)


clf_3.fit(X_train,y_train)
# Make  predictions with different HP
y_preds_3 = clf_3.predict(X_valid)

# Evaluate the 3rd classifier on validation set
metrics_2 = evaluate_preds(y_valid,y_preds_3)
```

```
Accuracy: 84.44%
Precision: 82.14
Recall: 92.00
F1 score: 86.79
```

3.4.2 5.2 Hyperparameter tuning with RandomizedSearchCV

[178]:
```
from sklearn.model_selection import RandomizedSearchCV

grid = {"n_estimators":[10,100,200,500,1000,1200],
        "max_depth":[None, 5,10,20,30],
        "max_features": ["log2","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 ) # How much of the processor we␣
↪dedicate to the model
```

```
#Setup RandomizedSearchCV
rs_clf = RandomizedSearchCV(estimator=clf,
                            param_distributions=grid,
                            n_iter=10, # number of models to try
                            cv = 5, #5 fold cross validation
                            verbose = 2 # the detail of the output 0-3
                           )
# Fit the RSCV version of clf
rs_clf.fit(X_train,y_train); # automatically will make a valid set
```

```
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_split=6, n_estimators=1200; total time=   4.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=1200; total time=   4.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=1200; total time=   4.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=1200; total time=   4.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=1200; total time=   4.1s
[CV] END max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=100; total time=   0.2s
[CV] END max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=100; total time=   0.3s
[CV] END max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=100; total time=   0.2s
[CV] END max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=100; total time=   0.3s
[CV] END max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=200; total time=   0.6s
[CV] END max_depth=20, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=20, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=20, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=20, max_features=log2, min_samples_leaf=1,
```

```
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=20, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=4, n_estimators=10; total time=   0.0s
[CV] END max_depth=10, max_features=log2, 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_samples_split=6, n_estimators=500; total time=   1.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=4,
min_samples_split=4, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=4,
min_samples_split=4, n_estimators=200; total time=   0.5s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=4,
```

```
min_samples_split=4, n_estimators=200; total time=   0.5s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=4,
min_samples_split=4, n_estimators=200; total time=   0.6s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=4,
min_samples_split=4, n_estimators=200; total time=   0.6s
[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=   3.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2,
min_samples_split=4, n_estimators=1000; total time=   3.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2,
min_samples_split=4, n_estimators=1000; total time=   3.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2,
min_samples_split=4, n_estimators=1000; total time=   3.4s
```

[179]:
```
rs_clf.best_params_
```

[179]:
```
{'n_estimators': 200,
 'min_samples_split': 6,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': None}
```

[180]:
```
# Make predictions  with the best parameters
rs_y_preds = rs_clf.predict(X_test)

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

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

3.4.3 5.2 Hyperparameter tuning with GridSearchCV

[181]:
```
grid
```

[181]:
```
{'n_estimators': [10, 100, 200, 500, 1000, 1200],
 'max_depth': [None, 5, 10, 20, 30],
 'max_features': ['log2', 'sqrt'],
 'min_samples_split': [2, 4, 6],
 'min_samples_leaf': [1, 2, 4]}
```

[182]:
```
## Grid Search is kind of Brute Force Search

6*5*2*3*3*5  # amount of diffrent models the last 5 because of CV
```

[182]: 2700

[183]:
```
grid_2 = {'n_estimators': [ 100, 200, 500],
 'max_depth': [None],
 'max_features': ['log2', 'sqrt'], # auto is not supported
 'min_samples_split': [6],
 'min_samples_leaf': [1, 2]} # We reduced the combinations of Hyperparameters␣
 ↪based on the result of the RSCV
```

[184]:
```
3*1*2*1*2*5
```

[184]: 60

[185]:
```
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 ) # How much of the processor we␣
 ↪dedicate to the model

#Setup GridSearchCV
gs_clf = GridSearchCV(estimator=clf,
                           param_grid=grid_2,
                           cv = 5, #5 fold cross validation
                           verbose = 2 # the detail of the output 0-3
                          )
# Fit the RSCV version of clf
gs_clf.fit(X_train,y_train); # automatically will make a valid set
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
```

```
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.9s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.4s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
```

```
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=log2, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=200; total time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=6, n_estimators=500; total time=   1.7s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=100; total time=   0.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
```

```
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=200; total time=   0.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2,
min_samples_split=6, n_estimators=500; total time=   1.5s
```

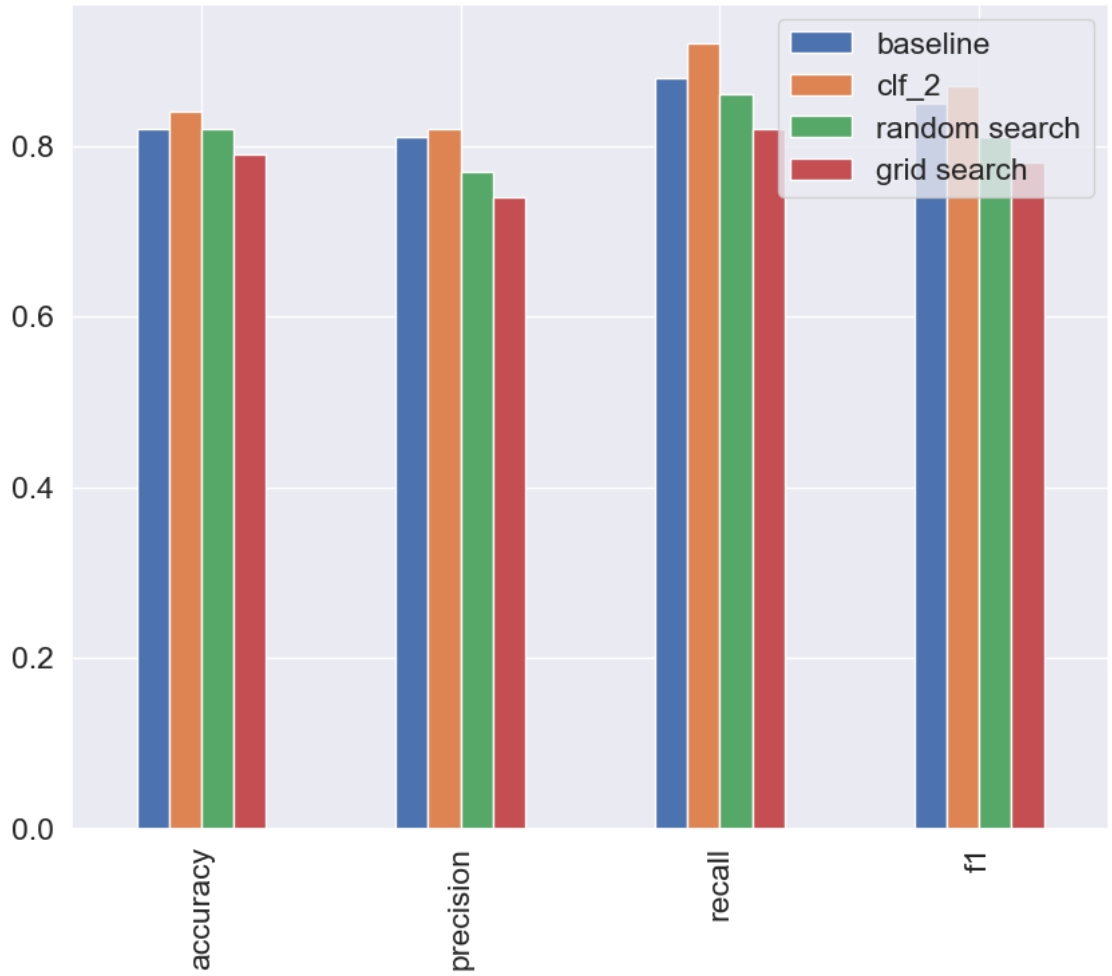
[186]:
```
gs_y_preds = gs_clf.predict(X_test)

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

```
Accuracy: 78.69%
Precision: 74.19
Recall: 82.14
F1 score: 77.97
```

Let's compare our diffrent models metrics

[187]:
```
compare_metrics = pd.DataFrame({"baseline":baseline_metrics,
                                "clf_2":metrics_2,
                                 "random search":rs_metrics,
                                 "grid search":gs_metrics})

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

[187]:
```
<Axes: >
```



[188]:
```
# it depends what should be the focus of the model accuracy or precision or␣
↪recall or f1
# look at: https://colab.research.google.com/drive/
↪1ISey96a5Ag6z2CvVZKVqTKNWRwZbZl0m
# for right train test split for all models
```

[189]:
```
whats_were_covering
```

[189]:
```
['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']
```

3.5 6. Save and load a trained model

Two ways to save and load machine learning models: 1. With Python's `pickle` module 2. With the `joblib`module

Pickle Python Object Serialization

[190]:
```
# Our Python Object is our model
import pickle

# Save an existing model to file
with open("gs_random_forest_model_1.pkl", "wb") as f: #
    pickle.dump(gs_clf, f)
```

[191]:
```
# Load a saved model
with open("gs_random_forest_model_1.pkl", "rb") as f:
    loaded_pickle_model = pickle.load(f)
```

[192]:
```
# Make some predictions
pickle_y_preds = loaded_pickle_model.predict(X_test)
evaluate_preds(y_test,pickle_y_preds)
```

```
Accuracy: 78.69%
Precision: 74.19
Recall: 82.14
F1 score: 77.97
```

[192]:
```
{'accuracy': 0.79, 'precision': 0.74, 'recall': 0.82, 'f1': 0.78}
```

3.5.1 Joblib Module

[193]:
```
from joblib import dump, load # joblib is more efficient!!!

# save model to file

dump(gs_clf, filename="gs_random_forest_model_1.joblib")
```

[193]:
```
['gs_random_forest_model_1.joblib']
```

[194]:
```
# import a saved joblib model

loaded_job_model = load(filename="gs_random_forest_model_1.joblib")
```

[195]:
```
joblib_y_preds = loaded_job_model.predict(X_test)
evaluate_preds(y_test,joblib_y_preds)
```

```
Accuracy: 78.69%
Precision: 74.19
Recall: 82.14
F1 score: 77.97
```

[195]:
```
{'accuracy': 0.79, 'precision': 0.74, 'recall': 0.82, 'f1': 0.78}
```

[196]:
```
whats_were_covering
```

[196]:
```
['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']
```

3.6 7. Putting it all together!

[197]:
```
data = pd.read_csv("data/car-sales-extended-missing-data.csv")
data
```

[197]:

	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 x 5 columns]

[198]:
```
data.dtypes
```

[198]:
```
Make              object
Colour            object
Odometer (KM)    float64
Doors            float64
Price            float64
dtype: object
```

[199]:
```
data.isna().sum()
```

[199]:
```
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 model on the data

[209]:
```
# Getting the data ready
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
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 diffrent 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 = door_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy ="mean" )),
    ])

# Setup the 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 the 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)
```

[209]: 0.21733566969263773

[210]:
```
# Warum Pipeline/Imputer wichtig ist:
# Alles was .fit() macht (Imputer, OneHotEncoder, Scaling) "lernt" Regeln aus␣
 ↪den Daten
# z.B. Mittelwert/ häufigste Kategorie/ alle vorhandenen Kategorien.
# Wenn du diese Schritte auf dem ganzen X fit-test, fließen Infos aus X_test ins␣
 ↪Training → Data Leakage.
# Das Modell sieht die Testdaten dann indirekt, weil X_train durch␣
 ↪Test-Statistiken transformiert wird.
# Pipeline verhindert das automatisch: fit nur auf X_train, transform auf X_test.
```

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

[212]:
```
# use GSCV 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": ["sqrt"],
    "model__min_samples_split":[2,4]
}

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

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.2s
```

```
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.4s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   4.1s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.7s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.9s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   4.4s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   3.8s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   3.7s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   3.7s
```

```
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   4.6s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   3.9s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.4s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.5s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.9s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.6s
```

```
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   3.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   3.3s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   3.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   4.2s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.9s
[CV] END model__max_depth=None, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.6s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.2s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.2s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
```

```
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   2.0s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.0s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   1.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=2, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   1.8s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=mean; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
```

```
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=100,
preprocessor__num__imputer__strategy=median; total time=   0.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   2.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   2.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   2.1s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   2.5s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=mean; total time=   2.8s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.9s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.5s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.5s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.6s
[CV] END model__max_depth=5, model__max_features=sqrt,
model__min_samples_split=4, model__n_estimators=1000,
preprocessor__num__imputer__strategy=median; total time=   2.8s
```

[212]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('cat',
Pipeline(steps=[('imputer',
```

```
                 SimpleImputer(fill_value='missing',
                               strategy='constant')),
                ('onehot',
                 OneHotEncoder(handle_unknown='ignore'))]),
       ['Make',
       'Colour']),
                                                                      ('door',
       Pipeline(steps=[('imputer',
                 SimpleImputer())]),
       ['Doors']),
                                                                      ('num',
       Pipeline(steps=[('imputer',
                 SimpleImputer())]),
       ['Odometer '
       '(KM)'])])),
                                           ('model', RandomForestRegressor())]),
                    param_grid={'model__max_depth': [None, 5],
                                'model__max_features': ['sqrt'],
                                'model__min_samples_split': [2, 4],
                                'model__n_estimators': [100, 1000],
                                'preprocessor__num__imputer__strategy': ['mean',
                                                                         'median']},
                    verbose=2)
```

[213]:
```
gs_model.score(X_test,y_test)
```

[213]:
```
0.28630677589213915
```

[214]:
```
whats_were_covering
```

[214]:
```
['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']
```

[ ]:
```

```