

## Problem statement

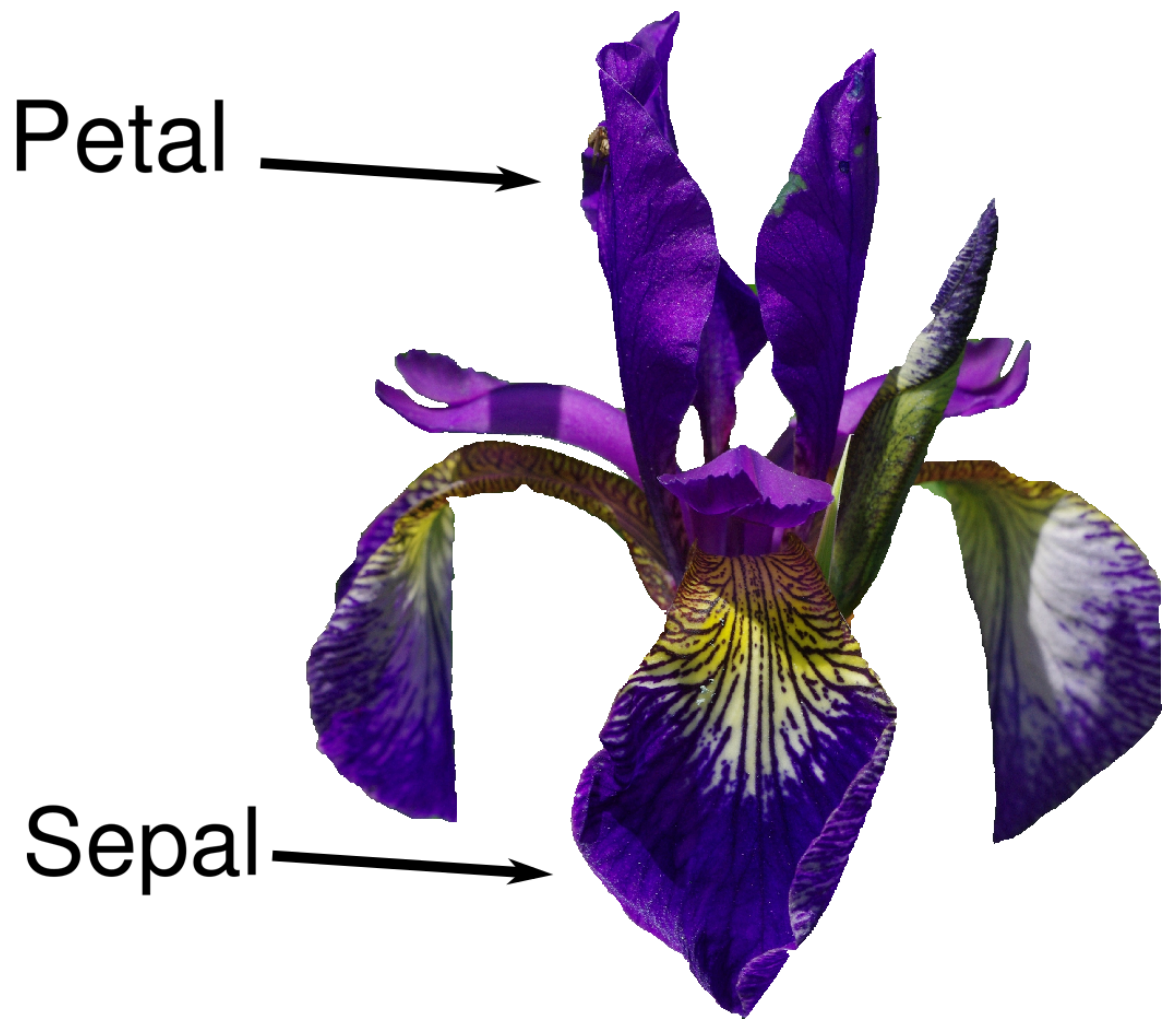
Let's assume that a Reema is interested in distinguishing the species of some iris flowers that she has found. She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimeters. She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species *setosa*, *versicolor*, or *virginica*. For these measurements, she can be certain of which species each iris belongs to. Let's assume that these are the only species Reema will encounter in the wild.

## Goal

To build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.

## Solution

Because we have measurements for which we know the correct species of iris, this is a supervised learning problem. In this problem, we want to predict one of several options (the species of iris). This is an example of a classification problem. The possible outputs (different species of irises) are called classes. Every iris in the dataset belongs to one of three classes(*setosa*, *versicolor*, or *virginica*), so this problem is a three-class classification problem. The desired output for a single data point (an iris) is the species of this flower. For a particular data point, the species it belongs to is called its label.



## 1. Discover and understand the Iris dataset

```
In [1]: # Load the dataset
from sklearn.datasets import load_iris
iris_dataset = load_iris()
iris_dataset
```

```
Out[1]: {'data': array([[5.1, 3.5, 1.4, 0.2],
                        [4.9, 3. , 1.4, 0.2],
                        [4.7, 3.2, 1.3, 0.2],
                        [4.6, 3.1, 1.5, 0.2],
                        [5. , 3.6, 1.4, 0.2],
                        [5.4, 3.9, 1.7, 0.4],
                        [4.6, 3.4, 1.4, 0.3],
                        [5. , 3.4, 1.5, 0.2],
                        [4.4, 2.9, 1.4, 0.2],
                        [4.9, 3.1, 1.5, 0.1],
                        [5.4, 3.7, 1.5, 0.2],
                        [4.8, 3.4, 1.6, 0.2],
                        [4.8, 3. , 1.4, 0.1],
                        [4.3, 3. , 1.1, 0.1],
                        [5.8, 4. , 1.2, 0.2],
                        [5.7, 4.4, 1.5, 0.4],
                        [5.4, 3.9, 1.3, 0.4],
                        [5.1, 3.5, 1.4, 0.3],
                        [5.7, 3.8, 1.7, 0.3],
                        [5.1, 3.8, 1.5, 0.3],
                        [5.4, 3.4, 1.7, 0.2],
                        [5.1, 3.7, 1.5, 0.4],
                        [4.6, 3.6, 1. , 0.2],
                        [5.1, 3.3, 1.7, 0.5],
                        [4.8, 3.4, 1.9, 0.2],
                        [5. , 3. , 1.6, 0.2],
                        [5. , 3.4, 1.6, 0.4],
                        [5.2, 3.5, 1.5, 0.2],
                        [5.2, 3.4, 1.4, 0.2],
                        [4.7, 3.2, 1.6, 0.2],
                        [4.8, 3.1, 1.6, 0.2],
                        [5.4, 3.4, 1.5, 0.4],
                        [5.2, 4.1, 1.5, 0.1],
                        [5.5, 4.2, 1.4, 0.2],
                        [4.9, 3.1, 1.5, 0.2],
                        [5. , 3.2, 1.2, 0.2],
                        [5.5, 3.5, 1.3, 0.2],
                        [4.9, 3.6, 1.4, 0.1],
                        [4.4, 3. , 1.3, 0.2],
                        [5.1, 3.4, 1.5, 0.2],
                        [5. , 3.5, 1.3, 0.3],
                        [4.5, 2.3, 1.3, 0.3],
                        [4.4, 3.2, 1.3, 0.2],
                        [5. , 3.5, 1.6, 0.6],
                        [5.1, 3.8, 1.9, 0.4],
                        [4.8, 3. , 1.4, 0.3],
                        [5.1, 3.8, 1.6, 0.2],
                        [4.6, 3.2, 1.4, 0.2],
                        [5.3, 3.7, 1.5, 0.2],
                        [5. , 3.3, 1.4, 0.2],
                        [7. , 3.2, 4.7, 1.4],
                        [6.4, 3.2, 4.5, 1.5],
                        [6.9, 3.1, 4.9, 1.5],
                        [5.5, 2.3, 4. , 1.3],
                        [6.5, 2.8, 4.6, 1.5],
                        [5.7, 2.8, 4.5, 1.3],
```

[6.3, 3.3, 4.7, 1.6],  
[4.9, 2.4, 3.3, 1. ],  
[6.6, 2.9, 4.6, 1.3],  
[5.2, 2.7, 3.9, 1.4],  
[5. , 2. , 3.5, 1. ],  
[5.9, 3. , 4.2, 1.5],  
[6. , 2.2, 4. , 1. ],  
[6.1, 2.9, 4.7, 1.4],  
[5.6, 2.9, 3.6, 1.3],  
[6.7, 3.1, 4.4, 1.4],  
[5.6, 3. , 4.5, 1.5],  
[5.8, 2.7, 4.1, 1. ],  
[6.2, 2.2, 4.5, 1.5],  
[5.6, 2.5, 3.9, 1.1],  
[5.9, 3.2, 4.8, 1.8],  
[6.1, 2.8, 4. , 1.3],  
[6.3, 2.5, 4.9, 1.5],  
[6.1, 2.8, 4.7, 1.2],  
[6.4, 2.9, 4.3, 1.3],  
[6.6, 3. , 4.4, 1.4],  
[6.8, 2.8, 4.8, 1.4],  
[6.7, 3. , 5. , 1.7],  
[6. , 2.9, 4.5, 1.5],  
[5.7, 2.6, 3.5, 1. ],  
[5.5, 2.4, 3.8, 1.1],  
[5.5, 2.4, 3.7, 1. ],  
[5.8, 2.7, 3.9, 1.2],  
[6. , 2.7, 5.1, 1.6],  
[5.4, 3. , 4.5, 1.5],  
[6. , 3.4, 4.5, 1.6],  
[6.7, 3.1, 4.7, 1.5],  
[6.3, 2.3, 4.4, 1.3],  
[5.6, 3. , 4.1, 1.3],  
[5.5, 2.5, 4. , 1.3],  
[5.5, 2.6, 4.4, 1.2],  
[6.1, 3. , 4.6, 1.4],  
[5.8, 2.6, 4. , 1.2],  
[5. , 2.3, 3.3, 1. ],  
[5.6, 2.7, 4.2, 1.3],  
[5.7, 3. , 4.2, 1.2],  
[5.7, 2.9, 4.2, 1.3],  
[6.2, 2.9, 4.3, 1.3],  
[5.1, 2.5, 3. , 1.1],  
[5.7, 2.8, 4.1, 1.3],  
[6.3, 3.3, 6. , 2.5],  
[5.8, 2.7, 5.1, 1.9],  
[7.1, 3. , 5.9, 2.1],  
[6.3, 2.9, 5.6, 1.8],  
[6.5, 3. , 5.8, 2.2],  
[7.6, 3. , 6.6, 2.1],  
[4.9, 2.5, 4.5, 1.7],  
[7.3, 2.9, 6.3, 1.8],  
[6.7, 2.5, 5.8, 1.8],  
[7.2, 3.6, 6.1, 2.5],  
[6.5, 3.2, 5.1, 2. ],  
[6.4, 2.7, 5.3, 1.9],

```
[6.8, 3. , 5.5, 2.1],  
[5.7, 2.5, 5. , 2. ],  
[5.8, 2.8, 5.1, 2.4],  
[6.4, 3.2, 5.3, 2.3],  
[6.5, 3. , 5.5, 1.8],  
[7.7, 3.8, 6.7, 2.2],  
[7.7, 2.6, 6.9, 2.3],  
[6. , 2.2, 5. , 1.5],  
[6.9, 3.2, 5.7, 2.3],  
[5.6, 2.8, 4.9, 2. ],  
[7.7, 2.8, 6.7, 2. ],  
[6.3, 2.7, 4.9, 1.8],  
[6.7, 3.3, 5.7, 2.1],  
[7.2, 3.2, 6. , 1.8],  
[6.2, 2.8, 4.8, 1.8],  
[6.1, 3. , 4.9, 1.8],  
[6.4, 2.8, 5.6, 2.1],  
[7.2, 3. , 5.8, 1.6],  
[7.4, 2.8, 6.1, 1.9],  
[7.9, 3.8, 6.4, 2. ],  
[6.4, 2.8, 5.6, 2.2],  
[6.3, 2.8, 5.1, 1.5],  
[6.1, 2.6, 5.6, 1.4],  
[7.7, 3. , 6.1, 2.3],  
[6.3, 3.4, 5.6, 2.4],  
[6.4, 3.1, 5.5, 1.8],  
[6. , 3. , 4.8, 1.8],  
[6.9, 3.1, 5.4, 2.1],  
[6.7, 3.1, 5.6, 2.4],  
[6.9, 3.1, 5.1, 2.3],  
[5.8, 2.7, 5.1, 1.9],  
[6.8, 3.2, 5.9, 2.3],  
[6.7, 3.3, 5.7, 2.5],  
[6.7, 3. , 5.2, 2.3],  
[6.3, 2.5, 5. , 1.9],  
[6.5, 3. , 5.2, 2. ],  
[6.2, 3.4, 5.4, 2.3],  
[5.9, 3. , 5.1, 1.8]]),  
  
'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0,  
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,  
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),  
  
'frame': None,  
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),  
'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n-----\n\n**Data Set Characteristics:**\n\n :Number of Instances: 150 (50 in each of three classes)\n :Number of Attributes: 4 numeric, predictive attributes and the class\n :Attribute Information:\n - sepal length in cm\n - sepal width in cm\n \n - petal length in cm\n - petal width in cm\n - class:\n- Iris-Setosa\n      - Iris-Versicolour\n      - Iris-Virginic  
a\n \n :Summary Statistics:\n\n ======
```

```

Correlation\n      =====\n
sepal length:  4.3  7.9   5.84  0.83   0.7826\n    sepal width:   2.0  4.4
3.05  0.43  -0.4194\n    petal length:  1.0  6.9   3.76  1.76   0.9490 (high!)\n
petal width:   0.1  2.5   1.20  0.76   0.9565 (high!)\n    =====
===== \n\n      :Missing Attribute Value
s: None\n      :Class Distribution: 33.3% for each of 3 classes.\n      :Creator: R.A.
Fisher\n      :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n      :Date: Ju
ly, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset i
s taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the
UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perh
aps the best known database to be found in the\npattern recognition literature. F
isher\'s paper is a classic in the field and\nis referenced frequently to this da
y. (See Duda & Hart, for example.) The\ndata set contains 3 classes of 50 instan
ces each, where each class refers to a\ntype of iris plant. One class is linearly
separable from the other 2; the\nlatter are NOT linearly separable from each othe
r.\n\n.. topic:: References\n\n    - Fisher, R.A. "The use of multiple measurements
in taxonomic problems"\n        Annual Eugenics, 7, Part II, 179-188 (1936); also in
"Contributions to\n        Mathematical Statistics" (John Wiley, NY, 1950).\n    - Dud
a, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n        (Q32
7.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n    - Dasarathy, B.
V. (1980) "Nosing Around the Neighborhood: A New System\n        Structure and Classi
fication Rule for Recognition in Partially Exposed\n        Environments". IEEE Tran
sactions on Pattern Analysis and Machine\n        Intelligence, Vol. PAMI-2, No. 1, 6
7-71.\n    - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transact
ions\n        on Information Theory, May 1972, 431-433.\n    - See also: 1988 MLC Proc
eedings, 54-64. Cheeseman et al\'s AUTOCLASS II\n        conceptual clustering system
finds 3 classes in the data.\n    - Many, many more ...',
'feature_names': ['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)'],
'filename': 'iris.csv',
'data_module': 'sklearn.datasets.data'}

```

```

In [2]: # Keys of iris dataset
print("Keys of iris_dataset: \n{}".format(iris_dataset.keys()))

```

```

Keys of iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'fil
ename', 'data_module'])

```

```

In [5]: # The value of the key DESCR is a short description of the dataset
print(iris_dataset['DESCR'])

```

```
.. _iris_dataset:
```

```
Iris plants dataset
```

```
-----
```

```
**Data Set Characteristics:**
```

```
:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm
  - class:
    - Iris-Setosa
    - Iris-Versicolour
    - Iris-Virginica
```

```
:Summary Statistics:
```

```
=====  ====  ====  =====  =====  =====
                        Min  Max    Mean    SD    Class Correlation
=====  ====  ====  =====  =====  =====
sepal length:   4.3  7.9    5.84    0.83    0.7826
sepal width:    2.0  4.4    3.05    0.43   -0.4194
petal length:   1.0  6.9    3.76    1.76    0.9490 (high!)
petal width:    0.1  2.5    1.20    0.76    0.9565 (high!)
=====  ====  ====  =====  =====  =====
```

```
:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988
```

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

```
.. topic:: References
```

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed

Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.

- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

```
In [6]: # The value of the key target_names is an array of strings, containing the species
print("Target names: {}".format(iris_dataset['target_names']))
```

Target names: ['setosa' 'versicolor' 'virginica']

```
In [7]: # The value of feature_names is a list of strings, giving the description of each f
print("Feature names: \n{}".format(iris_dataset['feature_names']))
```

Feature names:

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

```
In [10]: # The data itself is contained in the target and data fields. data contains the num
print("Type of data: {}".format(type(iris_dataset['data'])))
print("data: {}".format(iris_dataset['data']))
```



Type of data: <class 'numpy.ndarray'>

data: [[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.6 1.4 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]  
[6.3 3.3 4.7 1.6]  
[4.9 2.4 3.3 1. ]  
[6.6 2.9 4.6 1.3]  
[5.2 2.7 3.9 1.4]  
[5. 2. 3.5 1. ]  
[5.9 3. 4.2 1.5]  
[6. 2.2 4. 1. ]  
[6.1 2.9 4.7 1.4]  
[5.6 2.9 3.6 1.3]  
[6.7 3.1 4.4 1.4]  
[5.6 3. 4.5 1.5]  
[5.8 2.7 4.1 1. ]  
[6.2 2.2 4.5 1.5]  
[5.6 2.5 3.9 1.1]  
[5.9 3.2 4.8 1.8]  
[6.1 2.8 4. 1.3]  
[6.3 2.5 4.9 1.5]  
[6.1 2.8 4.7 1.2]  
[6.4 2.9 4.3 1.3]  
[6.6 3. 4.4 1.4]  
[6.8 2.8 4.8 1.4]  
[6.7 3. 5. 1.7]  
[6. 2.9 4.5 1.5]  
[5.7 2.6 3.5 1. ]  
[5.5 2.4 3.8 1.1]  
[5.5 2.4 3.7 1. ]  
[5.8 2.7 3.9 1.2]  
[6. 2.7 5.1 1.6]  
[5.4 3. 4.5 1.5]  
[6. 3.4 4.5 1.6]  
[6.7 3.1 4.7 1.5]  
[6.3 2.3 4.4 1.3]  
[5.6 3. 4.1 1.3]  
[5.5 2.5 4. 1.3]  
[5.5 2.6 4.4 1.2]  
[6.1 3. 4.6 1.4]  
[5.8 2.6 4. 1.2]  
[5. 2.3 3.3 1. ]  
[5.6 2.7 4.2 1.3]  
[5.7 3. 4.2 1.2]  
[5.7 2.9 4.2 1.3]  
[6.2 2.9 4.3 1.3]  
[5.1 2.5 3. 1.1]  
[5.7 2.8 4.1 1.3]  
[6.3 3.3 6. 2.5]  
[5.8 2.7 5.1 1.9]  
[7.1 3. 5.9 2.1]  
[6.3 2.9 5.6 1.8]  
[6.5 3. 5.8 2.2]  
[7.6 3. 6.6 2.1]  
[4.9 2.5 4.5 1.7]  
[7.3 2.9 6.3 1.8]  
[6.7 2.5 5.8 1.8]  
[7.2 3.6 6.1 2.5]  
[6.5 3.2 5.1 2. ]

```
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
[6.5 3.  5.2 2. ]
[6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]
```

```
In [9]: # The rows in the data array correspond to flowers, while the columns represent the
print("Shape of data: {}".format(iris_dataset['data'].shape))
```

Shape of data: (150, 4)

We see that the array contains measurements for 150 different flowers. Remember that the individual items are called samples in machine learning, and their properties are called features. The shape of the data array is the number of samples multiplied by the number of features. This is a convention in scikit-learn, and your data will always be assumed to be in this shape. Here are the feature values for the first five samples:

```
In [12]: # feature values for the first five samples
print("First five columns of data:\n{}".format(iris_dataset['data'][:5]))
# From this data, we can see that all of the first five flowers have a petal width
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

Shape of target: (150,)

[illegible]

To assess the model's performance, we show it new data (data that it hasn't seen before) for which we have labels. This is usually done by splitting the labeled data we have collected (here, our 150 flower measurements) into two parts:

1. First part of the data is used to build our machine learning model, and is called the training data or training set.
2. The rest of the data will be used to assess how well the model works; this is called the test data, test set, or hold-out set.

This function extracts 75% of the rows in the data as the training set, together with the corresponding labels for this data.

The remaining 25% of the data, together with the remaining labels, is declared as the test set.

In scikit-learn, data is usually denoted with a capital  $X$ , while labels are denoted by a lowercase  $y$ .

This is inspired by the standard formulation  $f(x)=y$  in mathematics, where  $x$  is the input to a function and  $y$  is the output.

Following more conventions from mathematics, we use a capital  $X$  because the data is a two-dimensional array (a matrix) and a lowercase  $y$  because the target is a one-dimensional array (a vector).

## 2.1 Split the data into training and testing set

```
In [21]: #####
# random_state: To make sure that we will get the same output if we run the same f
# we provide the pseudorandom number generator with a fixed seed using the random_s
# This will make the outcome deterministic, so this line will always have the same
#####
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                    random_state=0)
```

```
In [22]: # The output of the train_test_split function is X_train, X_test, y_train, and y_te
# X_train contains 75% of the rows of the dataset, and X_test contains the remainin
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
```

```
X_train shape: (112, 4)
y_train shape: (112,)
```

```
In [23]: print("X_train shape: {}".format(X_test.shape))
print("y_train shape: {}".format(y_test.shape))
```

```
X_train shape: (38, 4)
y_train shape: (38,)
```

## 2.2 Visualize the data

Before building a machine learning model it is often a good idea to inspect the data, to see if the task is easily solvable without machine learning, or if the desired information might not be contained in the data.

Inspecting your data is a good way to find abnormalities and peculiarities. Maybe some of your irises were measured using inches and not centimeters, for example. In the real world, inconsistencies in the data and unexpected measurements are very common.

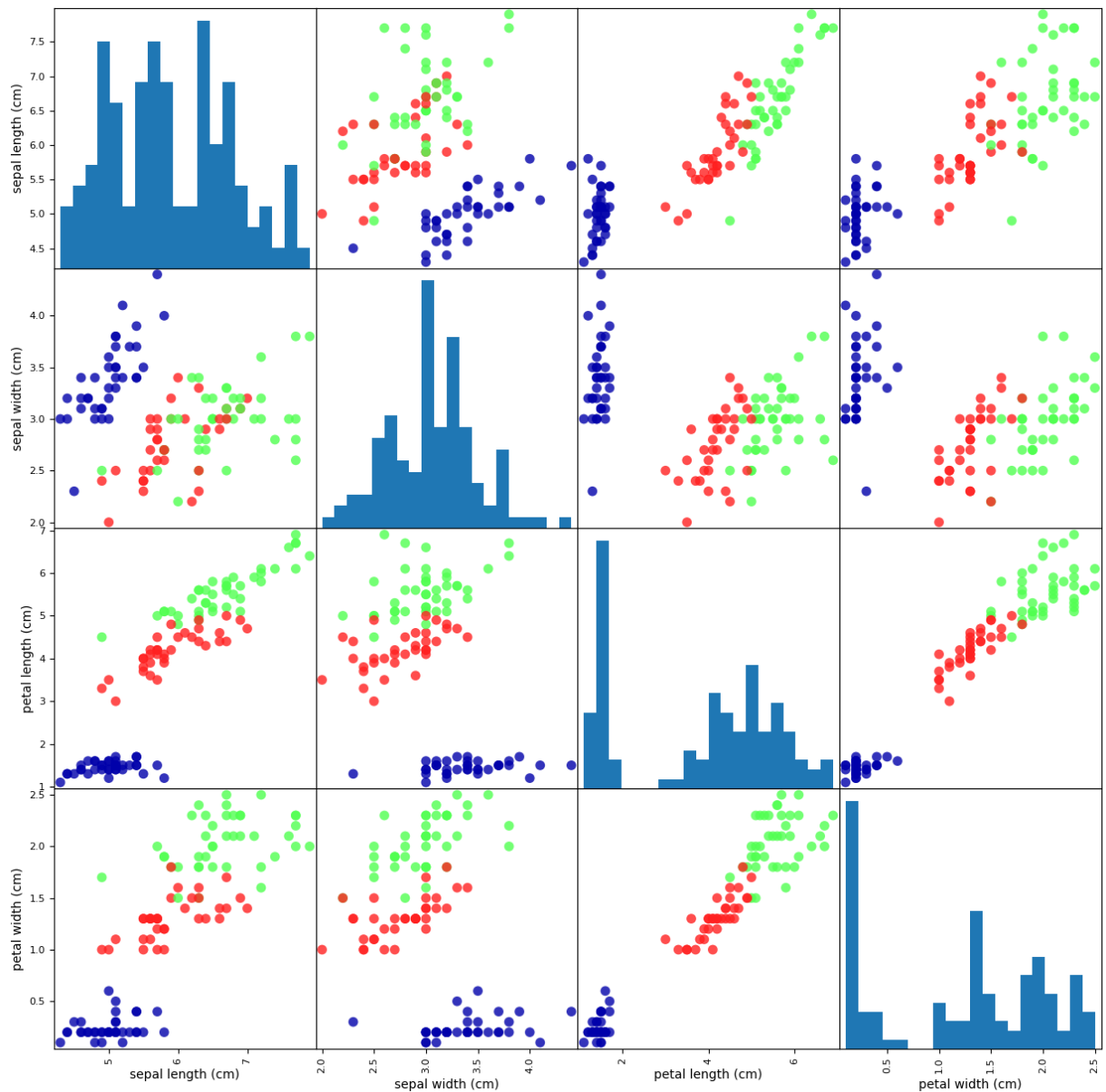
One of the best ways to inspect data is to visualize it. One way to do this is by using a scatter plot. A scatter plot of the data puts one feature along the x-axis and another along the y-axis, and draws a dot for each data point. Unfortunately, computer screens have only two dimensions, which allows us to plot only two (or maybe three) features at a time. It is difficult to plot datasets with more than three features this way. One way around this problem is to do a pair plot, which looks at all possible pairs of features. If you have a small number of

features, such as the four we have here, this is quite reasonable. You should keep in mind, however, that a pair plot does not show the interaction of all of features at once, so some interesting aspects of the data may not be revealed when visualizing it this way.

Below code generate pair plot of the features in the training set. The data points are colored according to the species the iris belongs to. To create the plot, we first convert the NumPy array into a pandas DataFrame. pandas has a function to create pair plots called `scatter_matrix`. The diagonal of this matrix is filled with histograms of each feature:

```
In [ ]: pip install mglearn
```

```
In [37]: from pandas.plotting import scatter_matrix
import mglearn
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
grr = scatter_matrix(iris_dataframe,
                      c=y_train, figsize=(15, 15),
                      marker='o',
                      hist_kws={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
# https://pandas.pydata.org/docs/reference/api/pandas.plotting.scatter\_matrix.html
```



From the plots, we can see that the three classes seem to be relatively well separated using the sepal and petal measurements. This means that a machine learning model will likely be able to learn to separate them.

To know more about scatter\_matrix: <https://www.youtube.com/watch?v=SNurMMcFVy8&t=26s>

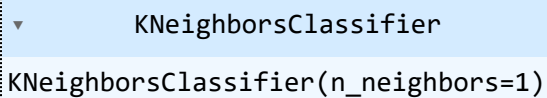
## 2.3 Building Your First Model: k-Nearest Neighbors

we will use a k-nearest neighbors classifier, which is easy to understand. Building this model only consists of storing the training set. To make a prediction for a new data point, the algorithm finds the point in the training set that is closest to the new point. Then it assigns the label of this training point to the new data point.

The  $k$  in  $k$ -nearest neighbors signifies that instead of using only the closest neighbor to the new data point, we can consider any fixed number  $k$  of neighbors in the training (for example, the closest three or five neighbors). Then, we can make a prediction using the majority class among these neighbors.

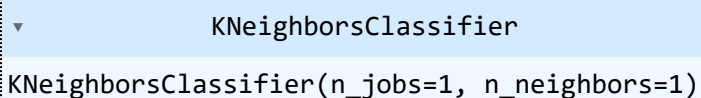
All machine learning models in scikit-learn are implemented in their own classes, which are called Estimator classes. The  $k$ -nearest neighbors classification algorithm is implemented in the `KNeighborsClassifier` class in the `neighbors` module. Before we can use the model, we need to instantiate the class into an object. This is when we will set any parameters of the model. The most important parameter of `KNeighborsClassifier` is the number of neighbors, which we will set to 1.

```
In [42]: from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=1)
```

```
Out[42]:  KNeighborsClassifier  
KNeighborsClassifier(n_neighbors=1)
```

The `knn` object encapsulates the algorithm that will be used to build the model from the training data, as well the algorithm to make predictions on new data points. It will also hold the information that the algorithm has extracted from the training data. In the case of `KNeighborsClassifier`, it will just store the training set. To build the model on the training set, we call the `fit` method of the `knn` object, which takes as arguments the NumPy array `X_train` containing the training data and the NumPy array `y_train` of the corresponding training labels:

```
In [45]: knn.fit(X_train, y_train)  
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
weights='uniform')
```

```
Out[45]:  KNeighborsClassifier  
KNeighborsClassifier(n_jobs=1, n_neighbors=1)
```

The `fit` method returns the `knn` object itself (and modifies it in place), so we get a string representation of our classifier. The representation shows us which parameters were used in creating the model. Nearly all of them are the default values, but you can also find `n_neighbors=1`, which is the parameter that we passed. Most models in scikit-learn have many parameters, but the majority of them are either speed optimizations or for very special use cases

## 2.4 Making Predictions



We can now make predictions using this model on new data for which we might not know the correct labels.

Imagine we found an iris in the wild with a sepal length of 5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm.

What species of iris would this be? We can put this data into a NumPy array, again by calculating the shape—that is, the number of samples (1) multiplied by the number of features (4)

```
In [47]: import numpy as np
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

X\_new.shape: (1, 4)

**To make a prediction, we call the predict method of the knn object:**

```
In [49]: prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))
```

Prediction: [0]

Predicted target name: ['setosa']

Our model predicts that this new iris belongs to the class 0, meaning its species is setosa.

## 2.5 Evaluating the Model prediction

We will use test set which we created earlier to evaluate result.

Therefore, we can make a prediction for each iris in the test data and compare it against its label (the known species). We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted:

```
In [50]: y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
```

Test set predictions:

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

```
In [52]: print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

Test set score: 0.97

**For this model, the test set accuracy is about 0.97, which means we made the right prediction for 97%**

of the irises in the test set.