Basics of Scikit-Learn

Data Representation

- Machine learning is about creating models from data: for that reason, it is important to understant how data can be represented.
- In Scikit-Learnm, the best way to think about data is in terms of tables of data.

Data as `table`

- A basic table is a two-dimensional grid of data, in which the rows represent individual elements of the dataset, and the columns represent quantities related to each of these elements.

In [8]:
```
# For example, consider the Iris dataset
# in the form of a Pandas DataFrame using the seaborn library

import seaborn as sns

iris = sns.load_dataset("iris")
iris.head()
```

Out[8]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- In a dataset, each row of the data refers to a single observed flower, and the number of rows is the total number of flowers in the dataset.
- In general, the rows of the matrix are referred to as `samples`, and the number of rows as `n_samples`.
- Likewise, each column of the data refers to a particular quantitative piece of information that describes each sample.
- In general, the columns of the matrix are referred to as `features`, and the number of columns as `n_features`.

Features matrix

- This table layout makes clear that the information can be thought of as a two-dimensional numerical array or matrix, which we will call the **features matrix**.
- By convention, this *features matrix* is often stored in a variable named `X`.
- The features matrix is assumed to be two-dimensional, with shape `[n_samples, n_features]`, and is most often contained in a NumPy array or a Pandas `DataFrame`, though some Scikit-Learn models also accept SciPy sparse matrices.
- The samples (i.e., rows) always refer to the individual objects described by the dataset. For example, the sample might be a flower, a person, a document, an image, a sound file, a video, an astronomical object, or anything else you can describe with a set of quantitative measurements.
- The features (i.e., columns) always refer to the distinct observations that describe each sample in a quantitative manner.

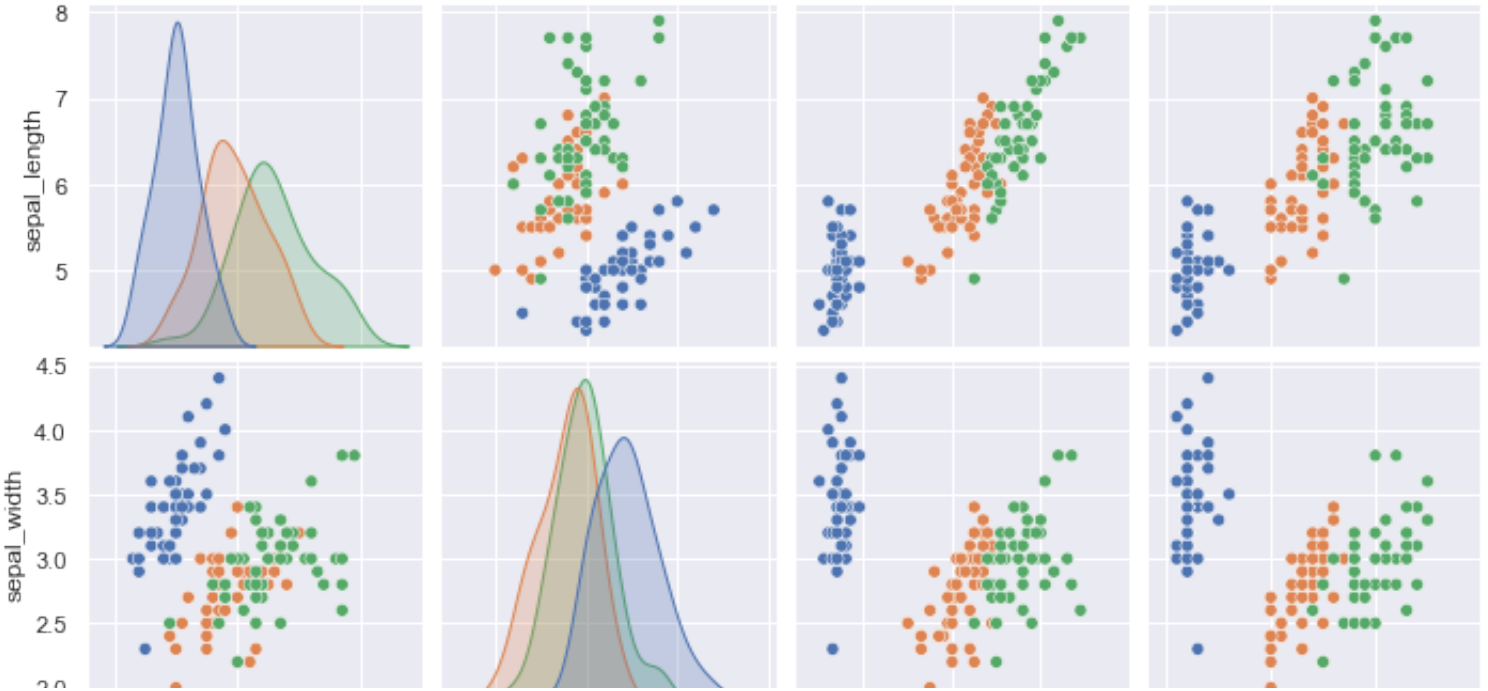
Target array

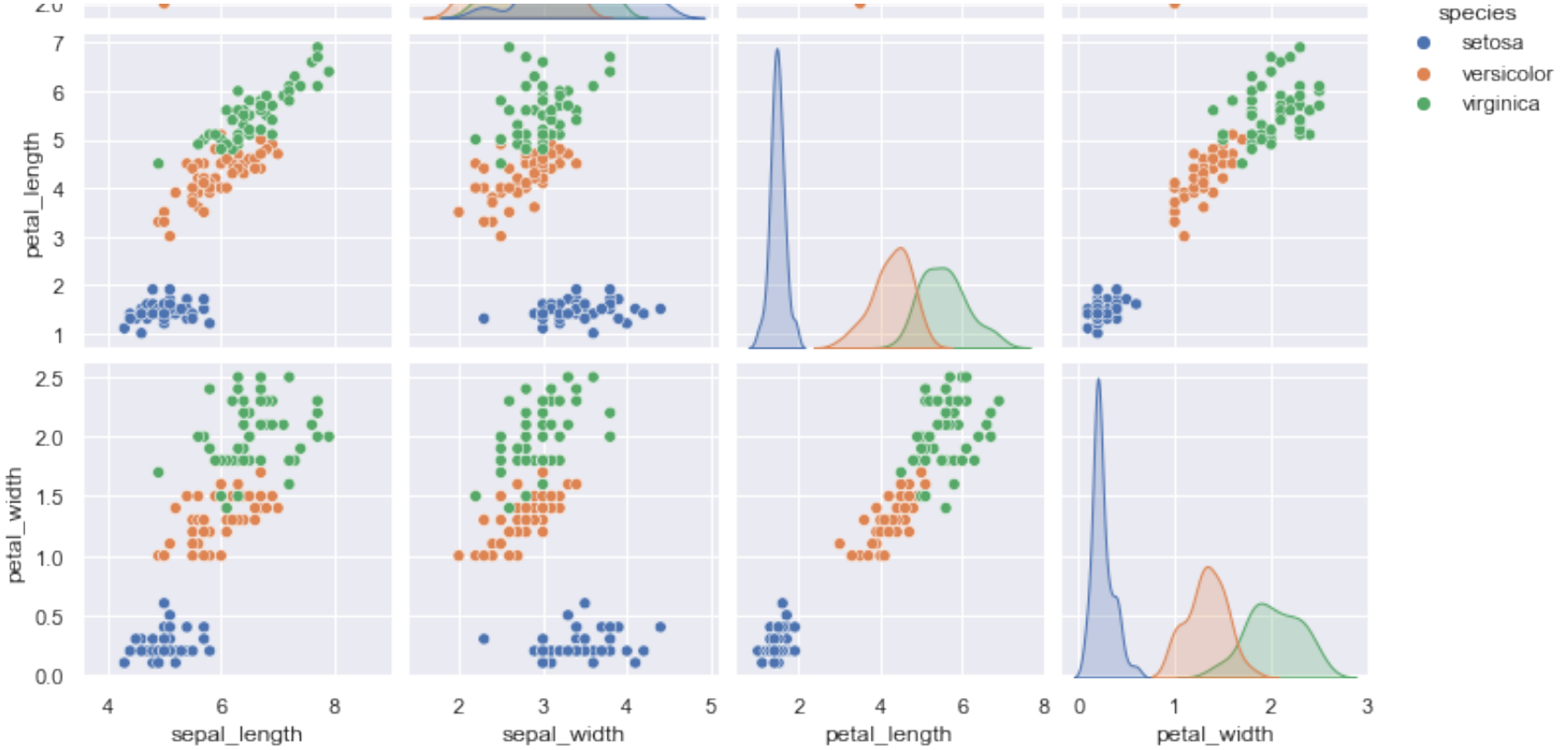
- In addition to the feature matrix `X`, there is a label or target array, which by convention is usually called `y`.
- The target array is usually one dimensional, with length `n_samples`, and is generally contained in a NumPy array or Pandas `Series`.
- The target array may have continuous numerical values, or discrete classes/labels.
- The target array is usually the quantity we want to predict from the data: in statistical terms, it is the *dependent variable*. For example, in the `iris` data we may wish to construct a model that can predict the species of flower based on the other measurements; in this case, the `species` column would be considered the target array.

In [9]:

```
# let's visualize the iris data using seaborn
# with the target array 'species' in mind

%matplotlib inline
sns.pairplot(iris, hue='species');
```





- For use in Scikit-Learn, we need to extract the features matrix and target array from the `DataFrame`.

In [16]:

```
# here, axis=1 means to drop from columns
# (1 or 'columns') and (0 or 'rows')
X_iris = iris.drop('species', axis='columns')
X_iris # features matix
```

Out[16]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [14]:

```
y_iris = iris['species']
y_iris # target array/vector
```

Out[14]:

```
0          setosa
1          setosa
2          setosa
3          setosa
4          setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

Estimator API

- Estimator API (application programming interface) is one of the main APIs implemented by Scikit-learn.
- Every machine learning algorithm in Scikit-Learn is implemented via the Estimator API, which provides a consistent interface for a wide range of machine learning applications.

Guiding Principles of the API

- The Scikit-Learn API is designed with the following guiding principles in mind:
  - *Consistency*: All objects share a common interface drawn from a limited set of methods, with consistent documentation.
  - *Limited object hierarchy*: Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.
  - *Composition*: Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.
  - *Sensible defaults*: When models require user-specified parameters, the library defines an appropriate default value.
  - *Inspection*: All specified parameter values are exposed as public attributes.

Basics of the API

- Most commonly, the steps in using the Scikit-Learn estimator API are as follows:

  1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.
  2. Choose model hyperparameters by instantiating this class with desired values.
  3. Arrange data into a features matrix and target vector.
  4. Fit the model to your data by calling the `fit()` method of the model instance.
  5. Apply the Model to new data:
     - For supervised learning, often we predict labels for unknown data using the `predict()` method.
     - For unsupervised learning, we often transform or infer properties of the data using the `transform()` or `predict()` method.
