Sckit-Learn I

September 26, 2021

1 Introduction to scikit-learn

Scikit-learn is a machine learning library in Python.

Scikit-learn is the first of the several machine learning libraries we will explore in this course. It is relatively approachable, supports a wide variety of traditional machine learning models, and is ubiquitous in the world of data science.

1.1 Datasets

Scikit-learn contains methods for loading, fetching, and making (generating) data. The methods for doing this all fall under the datasets subpackage. Most of the functions in this package have `load`, `fetch`, or `make` in the name to let you know what the method is doing under the hood.

Loading functions bring static datasets into your program. The data comes pre-packaged with scikit-learn, so no network access is required.

Fetching functions also bring static datasets into your program. However, the data is pulled from the internet, so if you don't have network access, these functions might fail.

Generating functions make dynamic datasets based on some equation.

These pre-packaged dataset functions exist for many popular datasets, such as the MNIST digits dataset and the Iris flower dataset. The generation functions reference classic dataset "shape" formations such as moons and swiss rolls. These datasets are perfect for getting familiar with machine learning.

1.1.1 Loading

Let us first look at an example of loading data. We will load the iris flowers dataset using the load_iris function.

```
[ ]: from sklearn.datasets import load_iris

     iris_data = load_iris()
     print(iris_data)
```

```
{'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, 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, 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, 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        - petal length in
       cm\n        - petal width in cm\n        - class:\n                - Iris-
       Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n
       \n    :Summary Statistics:\n\n    ============== ==== ==== ======= =====
       ====================\n                    Min  Max   Mean    SD   Class
       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 Values: 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: July, 1988\n\nThe famous Iris
       database, first used by Sir R.A. Fisher. The dataset is 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 perhaps the best known
       database to be found in the\npattern recognition literature.  Fisher\'s paper is
       a classic in the field and\nis referenced frequently to this day.  (See Duda &
       Hart, for example.)  The\ndata set contains 3 classes of 50 instances 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
```

```
other.\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   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and
Scene Analysis.\n     (Q327.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 Classification Rule for Recognition in Partially
Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and
Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972)
"The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information
Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 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'}
```

That's a lot to take in. Let's examine this loaded data a little more closely. First we'll see what data type this dataset is:

[ ]:
```python
type(iris_data)
```

[ ]:
```
sklearn.utils.Bunch
```

`sklearn.utils.Bunch` is a type that you'll see quite often when working with datasets built into scikit-learn. It is a dictionary-like container for feature and target data within a dataset.

You won't find much documentation about `Bunch` objects because they are not really meant for usage beyond containing data native to scikit-learn.

Let's look at the attributes of the iris dataset:

[ ]:
```python
dir(iris_data)
```

[ ]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

`DESCR` is a description of the dataset.

[ ]:
```python
print(iris_data['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 ...
```

`filename` is the name of the source file where the data is stored.

```
[ ]: print(iris_data['filename'])
```

```
iris.csv
```

`feature_names` is the name of the feature columns.

```
[ ]: print(iris_data['feature_names'])
```

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

`target_names`, despite the name, is not the names of the target columns. There is only one column of targets.

Instead, `target_names` is the human-readable names of the classes in the target list within the bunch. In this case,`target_names` is the names of the three species of iris in this dataset.

```
[ ]: print(iris_data['target_names'])
```

```
['setosa' 'versicolor' 'virginica']
```

We can now examine `target` and see that it contains zeros, ones, and twos. These correspond to the target names 'setosa', 'versicolor', and 'virginica'.

```
[ ]: print(iris_data['target'])
```

```
[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 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 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 2 2
 2 2]
```

Last, we'll look at the `data` within the bunch. The data is an array of arrays. Each sub-array contains four values. These values match up with the `feature_names`. The first item in each sub-array is 'sepal length (cm)', the next is 'sepal width (cm)', and so on.

```
[ ]: print(iris_data['feature_names'])
     iris_data['data']
     #each value withi a subarray corresponds with the following features
```

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

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

The number of target values should always equal the number of rows in the data.

[ ]:
```
print(len(iris_data['data']))
print(len(iris_data['target']))
```

```
150
150
```

`Bunch` objects are an adequate container for data. They can be used directly to feed models. However, `Bunch` objects are *not* very good for analyzing and manipulating your data.

In this course, we will typically convert `Bunch` objects into Pandas `DataFrame` objects to make analysis, data cleaning, visualization, and train/test splitting easier.

To do this, we will take the matrix of feature data and append the target data to it to create a single matrix of data. We also take the list of feature names and append the word ‘species’ to represent the target classes in the matrix.

[ ]:
```
import pandas as pd
import numpy as np


#take matrix of the feature data and append the target data creating a single␣
 ↪matrix of data

#list of featrure names and append the word species to represent target classes␣
 ↪in the matrix
iris_df = pd.DataFrame(
  data=np.append(
    #inserting data
    iris_data['data'],
    np.array(iris_data['target']).reshape(len(iris_data['target']), 1),
    axis=1),
  columns=np.append(iris_data['feature_names'], ['species'])
)
```

```
iris_df.sample(n=10)
```

```
[ ]:     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
    22                 4.6               3.6                1.0               0.2
    82                 5.8               2.7                3.9               1.2
    4                  5.0               3.6                1.4               0.2
    34                 4.9               3.1                1.5               0.2
    77                 6.7               3.0                5.0               1.7
    3                  4.6               3.1                1.5               0.2
    54                 6.5               2.8                4.6               1.5
    8                  4.4               2.9                1.4               0.2
    63                 6.1               2.9                4.7               1.4
    44                 5.1               3.8                1.9               0.4

        species
    22      0.0
    82      1.0
    4       0.0
    34      0.0
    77      1.0
    3       0.0
    54      1.0
    8       0.0
    63      1.0
    44      0.0
```

You might notice that the integer representation of species got converted to a floating point number along the way. We can change that back.

```
[ ]: iris_df['species'] = iris_df['species'].astype('int64')

     iris_df.sample(n=10)
```

```
[ ]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
    76                  6.8               2.8                4.8               1.4
    47                  4.6               3.2                1.4               0.2
    89                  5.5               2.5                4.0               1.3
    27                  5.2               3.5                1.5               0.2
    98                  5.1               2.5                3.0               1.1
    14                  5.8               4.0                1.2               0.2
    21                  5.1               3.7                1.5               0.4
    134                 6.1               2.6                5.6               1.4
    79                  5.7               2.6                3.5               1.0
    145                 6.7               3.0                5.2               2.3

         species
    76         1
    47         0
```

```
89          1
27          0
98          1
14          0
21          0
134         2
79          1
145         2
```

1.1.2 Exercise 1

Load the Boston house price dataset into a Pandas `DataFrame`. Append the target values to the last column of the `DataFrame` called `boston_df`. Name the target column 'PRICE'.

Student Solution

```
[ ]: from sklearn.datasets import load_boston
     load_boston = load_boston()





     load_boston_df = pd.DataFrame(
       data=np.append(
         #inserting data
         load_boston['data'],
         np.array(load_boston['target']).reshape(len(load_boston['target']), 1),
         axis=1),
       columns=np.append(load_boston['feature_names'], ['species'])
     )
     new_column = load_boston['target']
     new_dataframe = pd.DataFrame(new_column)
     name = ['PRICE']
     new_dataframe.columns = name


     boston_df = load_boston_df.append(new_dataframe)
     boston_df
     #load_boston_df.columns
```

```
/Users/josemartinez/opt/anaconda3/envs/data/lib/python3.9/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is
deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this case special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows:

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and:

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

[ ]:
```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0
..       …     …     …     …      …      …     …       …   …      …
501      NaN   NaN    NaN   NaN    NaN    NaN   NaN     NaN  NaN    NaN
502      NaN   NaN    NaN   NaN    NaN    NaN   NaN     NaN  NaN    NaN
503      NaN   NaN    NaN   NaN    NaN    NaN   NaN     NaN  NaN    NaN
504      NaN   NaN    NaN   NaN    NaN    NaN   NaN     NaN  NaN    NaN
505      NaN   NaN    NaN   NaN    NaN    NaN   NaN     NaN  NaN    NaN

     PTRATIO       B  LSTAT  species  PRICE
0       15.3  396.90   4.98     24.0    NaN
1       17.8  396.90   9.14     21.6    NaN
2       17.8  392.83   4.03     34.7    NaN
```

```
3        18.7  394.63   2.94     33.4    NaN
4        18.7  396.90   5.33     36.2    NaN
..         …      …     …        …     …
501       NaN     NaN    NaN      NaN   22.4
502       NaN     NaN    NaN      NaN   20.6
503       NaN     NaN    NaN      NaN   23.9
504       NaN     NaN    NaN      NaN   22.0
505       NaN     NaN    NaN      NaN   11.9

[1012 rows x 15 columns]
```

1.1.3 Fetching

Fetching is similar to loading. Scikit-learn will first see if it can find the dataset locally, and, if so, it will simply load the data. Otherwise, it will attempt to pull the data from the internet.

We can see fetching in action with the fetch_california_housing function below.

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

     housing_data = fetch_california_housing()

     type(housing_data)
```

```
[ ]: sklearn.utils.Bunch
```

The dataset is once again a `Bunch`.

If you follow the link to the fetch_california_housing documentation, you notice that the dataset is a **regression** dataset as opposed to the iris dataset, which was a **classification** dataset.

We can see the difference in the dataset by checking out the attributes of the `Bunch`.

```
[ ]: dir(housing_data)
```

```
[ ]: ['DESCR', 'data', 'feature_names', 'frame', 'target', 'target_names']
```

We see that four of the attributes that we expect are present, but 'target_names' is missing. This is because our target is now a continuous variable (home price) and not a discrete value (iris species).

```
[ ]: print(housing_data['target'][:10])
```

```
[4.526 3.585 3.521 3.413 3.422 2.697 2.992 2.414 2.267 2.611]
```

Converting a `Bunch` of regression data to a `DataFrame` is no different than for a `Bunch` of classification data.

```
[ ]: import pandas as pd
     import numpy as np
```

```
housing_df = pd.DataFrame(
  data=np.append(
    housing_data['data'],
    np.array(housing_data['target']).reshape(len(housing_data['target']), 1),
    axis=1),
  columns=np.append(housing_data['feature_names'], ['price'])
)

housing_df.sample(n=10)
```

```
[ ]:        MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
     3849   4.5081      33.0  4.511335   0.972292       844.0  2.125945     34.18
     15846  5.0519      40.0  5.094538   1.026261      1824.0  1.915966     37.75
     12021  4.1264      15.0  5.539007   1.107270      3795.0  3.364362     33.95
     10009  5.0389      13.0  5.070093   1.060748       629.0  2.939252     39.10
     207    2.1494      37.0  4.479924   1.097514      1608.0  3.074570     37.79
     2633   2.8958      32.0  5.758123   1.115523       706.0  2.548736     40.62
     10037  1.4241      16.0  4.285285   0.957958       642.0  1.927928     39.23
     15154  7.7569       6.0  7.972727   1.029870      2693.0  3.497403     33.00
     948    5.7233      25.0  6.523100   0.995529      2038.0  3.037258     37.71
     1273   2.2216      19.0  4.975186   1.004963       972.0  2.411911     37.93

            Longitude  price
     3849     -118.46  2.514
     15846    -122.43  3.561
     12021    -117.47  1.246
     10009    -121.14  1.715
     207      -122.22  1.325
     2633     -124.17  0.864
     10037    -121.02  1.250
     15154    -117.03  3.874
     948      -121.91  2.318
     1273     -121.70  1.567
```

1.1.4 Generating

In the example datasets we've seen so far in this Colab, the data is static and loaded from a file. Sometimes it makes more sense to generate a dataset. For this, we can use one of the many generator functions.

`make_regression` is a generator that creates a dataset with an underlying regression that you can then attempt to discover using various machine learning models.

In the example below, we create a dataset with 10 data points. For the sake of visualization, we have only one feature per datapoint, but we could ask for more.

The return values are the X and y values for the regression. X is a matrix of features. y is a list of targets.

Since a generator uses randomness to generate data, we are going to set a `random_state` in this

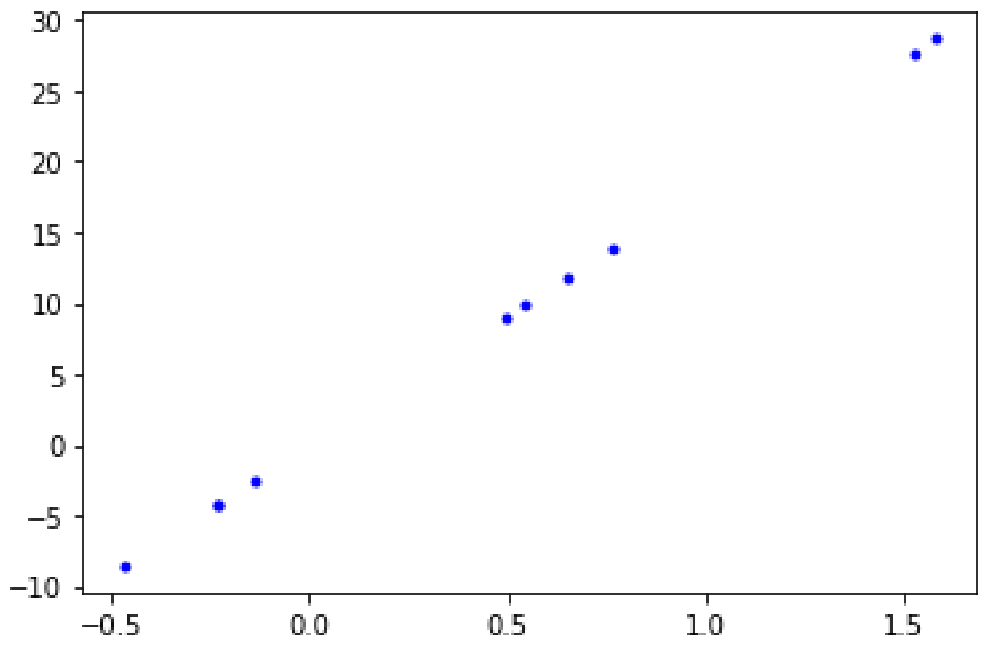
Colab for reproducibility. This ensures we get the same result every time we run the code. **You won't do this in your production code.**

```
[ ]: from sklearn.datasets import make_regression

     features, targets = make_regression(n_samples=10, n_features=1, random_state=42)

     features, targets
```

```
[ ]: (array([[ 1.57921282],
             [ 0.64768854],
             [-0.46947439],
             [ 0.76743473],
             [ 0.54256004],
             [-0.23413696],
             [-0.1382643 ],
             [ 1.52302986],
             [ 0.49671415],
             [-0.23415337]]),
      array([28.71403184, 11.77659472, -8.53621648, 13.95387945,  9.86509621,
             -4.25719445, -2.5139902 , 27.69248537,  9.03150346, -4.25749297]))
```

We can use a visualization library to plot the regression data.

```
[ ]: import matplotlib.pyplot as plt

     plt.plot(features, targets, 'b.')
     plt.show()
```

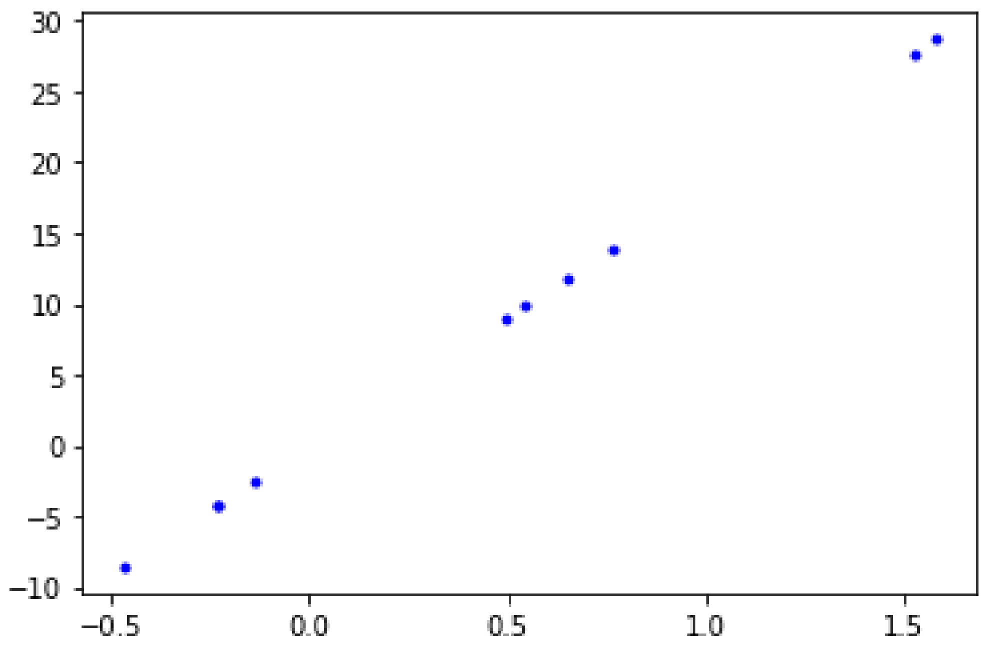


That data appears to have a very linear pattern!

If we want to make it more realistic (non-linear), we can add some noise during data generation.

Remember that random_state is for reproducibility only. Don't use this in your code unless you have a good reason to.

```
[ ]: from sklearn.datasets import make_regression


     plt.plot(features, targets, 'b.')
     plt.show()
```



There are dozens of dataset loaders and generators in the scikit-learn datasets package. When you want to play with a new machine learning algorithm, they are a great source of data for getting started.

1.1.5 Exercise 2

Search the scikit-learn datasets documentation and find a function to make a "Moons" dataset. Create a dataset with 75 samples. Use a random state of 42 and a noise of 0.08. Store the X return value in a variable called `features` and the y return value in a variable called `targets`.

Student Solution

```
[ ]:
```

```
#sklearn.datasets.make_moons(n_samples=100, *, shuffle=True, noise=None,␣
↪random_state=None)
from sklearn.datasets import make_moons
X,y = make_moons(n_samples=75,noise = 0.08, random_state = 42)

features = X
targets = y

print(y)
#sklearn.datasets.make_moons(n_samples=75,noise = 0.08, random_state = 42)
```

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

1.1.6 Exercise 3

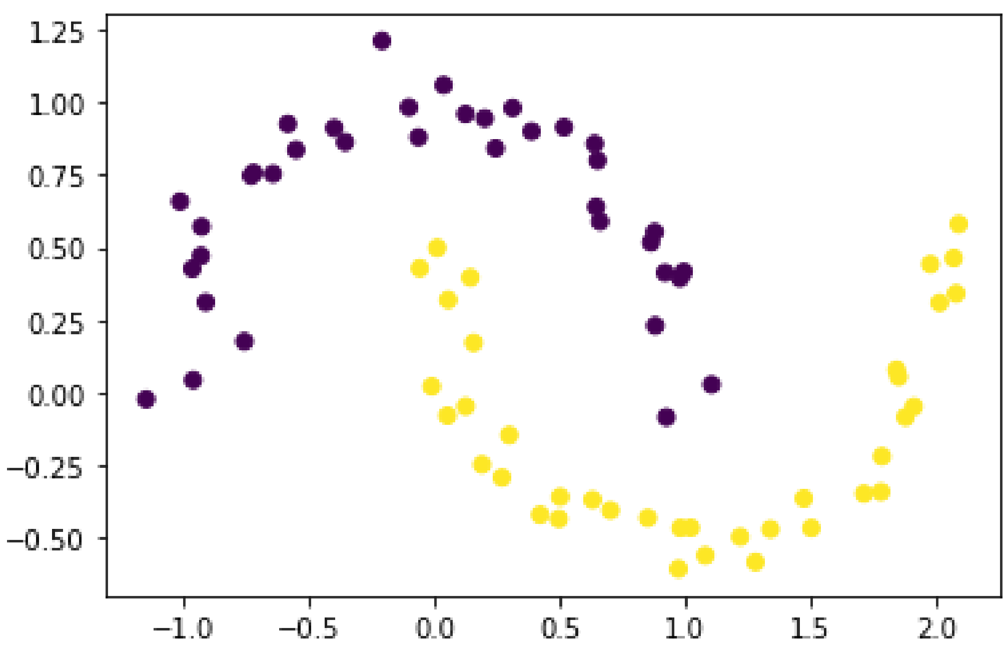
In Exercise Two, you created a "moons" dataset. In that dataset, the features are (x, y)-coordinates that can be graphed in a scatterplot. The targets are zeros and ones that represent a binary classification.

Use matplotlib's scatter method to visualize the data as a scatterplot. Use the c argument to make the dots for each class a different color.

Student Solution

```
[ ]: import matplotlib.pyplot as plt
     xcoordinates = []
     ycoordinates = []
     for i in features:
        xcoordinates.append(i[0])
        ycoordinates.append(i[1])
     plt.scatter(xcoordinates,ycoordinates, c = targets)

     #plt.scatter(features,targets)
     #plot.plot()
```

```
[ ]: <matplotlib.collections.PathCollection at 0x7f9053287ee0>
```



1.2 Models

Machine learning involves training a model to gain insight and predictive power from a dataset. Scikit-learn has support for many different types of models, ranging from classic algebraic models to more modern deep learning models.

Throughout the remainder of this course, you will learn about many of these models in much more depth. This section will walk you through some of the overarching concepts across all models.

1.2.1 Estimators

Most of the models in scikit-learn are considered estimators. An estimator is expected to implement two methods: `fit` and `predict`.

`fit` is used to train the model. At a minimum, it is passed the feature data used to train the model. In supervised models, it is also passed the target data.

`predict` is used to get predictions from the model. This method is passed features and returns target predictions.

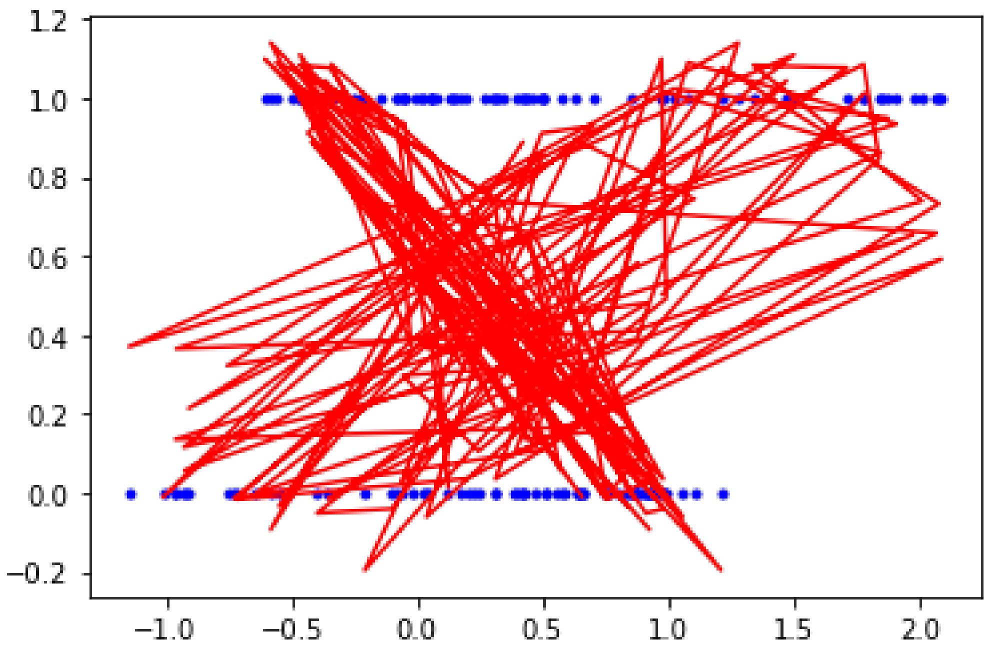
Let's see an example of this in action.

Linear regression is a simple model that you might have encountered in a statistics class in the past. The model attempts to draw a straight line through a set of data points, so the line is as close to as many points as possible.

We'll use scikit-learn's LinearRegression class to fit a line to the regression data that we generated earlier in this Colab. To do that, we simply call the `fit(features, targets)` method.

After fitting, we can ask the model for predictions. To do this, we use the `predict(features)` method.

```
[ ]: from sklearn.datasets import make_regression
     from sklearn.linear_model import LinearRegression




     regression = LinearRegression()
     regression.fit(features, targets)
     predictions = regression.predict(features)

     plt.plot(features, targets, 'b.')
     plt.plot(features, predictions, 'r-')
     plt.show()
```



At this point, don't worry too much about the details of what `LinearRegression` is doing. There is a deep-dive into regression problems coming up soon.

For now, just note the `fit/predict` pattern for training estimators, and know that you'll see it throughout our adventures with scikit-learn.

1.2.2 Transformers

In practice, it is rare that you will get perfectly clean data that is ready to feed into your model for training. Most of the time, you will need to perform some type of cleaning on the data first.

You've had some hands-on experience doing this in our Pandas Colabs. Scikit-learn can also be used to perform some data preprocessing.

Transformers are spread about within the scikit-learn library. Some are in the preprocessing module while others are in more specialized packages like compose, feature_extraction, impute, and others.

All transformers implement the `fit` and `transform` methods. The `fit` method calculates parameters necessary to perform the data transformation. The `transform` method actually applies the transformation. There is a convenience `fit_transform` method that performs both fitting and transformation in one method call.

Let's see a transformer in action.

We will use the MinMaxScaler to scale our feature data between zero and one. This scales the data with a linear transform so that the minimum value becomes 0 and the maximum value becomes 1, so all values are within 0 and 1.

Looking at our feature data pre-transformation, we can see values that are below zero and above one.

```
[ ]: features
```

```
[ ]: array([[ 0.97754808,  0.39475539],
            [ 1.78189288, -0.21831832],
            [ 0.65050093,  0.79957261],
            [ 0.92375778, -0.08382965],
            [-1.14624884, -0.02218683],
            [ 1.50110815, -0.46546351],
            [-0.72796104,  0.74708051],
            [ 0.51692696,  0.91378898],
            [ 1.83952168,  0.07913973],
            [ 1.77819003, -0.34058417],
            [-0.90880732,  0.3117695 ],
            [ 0.12540232, -0.04664424],
            [ 0.50184033, -0.35801865],
            [ 0.03742358,  1.05888979],
            [ 0.64473063,  0.63959507],
            [-0.35406285,  0.86244801],
            [ 0.91834691,  0.41354456],
            [-0.75524357,  0.17681177],
            [ 0.97987738, -0.46518352],
            [ 1.84785216,  0.05475808],
            [ 0.14421773,  0.39689783],
            [ 0.05571179,  0.31975237],
            [ 0.12482242,  0.95951448],
            [ 0.05289641, -0.07828259],
            [ 0.29897555, -0.14499539],
            [-0.92794884,  0.47121775],
            [ 0.8770787 ,  0.55311527],
            [ 1.27867297, -0.58246118],
            [ 0.26970088, -0.29120376],
            [-1.01110384,  0.65811134],
            [ 0.70161947, -0.40436449],
            [ 0.49651141, -0.43402886],
            [-0.06304992,  0.87951334],
            [ 1.47128822, -0.36324696],
            [-0.6412578 ,  0.75335289],
            [ 2.08693816,  0.58039895],
            [ 0.38727698,  0.89957458],
            [ 0.97194347, -0.60601875],
            [ 0.99296764,  0.41678474],
            [ 0.24435852,  0.8410018 ],
            [-0.05740765,  0.42868683],
            [ 0.66029429,  0.58932787],
            [ 1.97463263,  0.4431049 ],
            [ 0.86239319,  0.51735494],
            [-0.9590061 ,  0.0434791 ],
            [ 1.87526207, -0.08346191],
            [ 1.02073262, -0.46457981],
            [ 0.19030372, -0.24652299],
            [ 2.06739842,  0.46363273],
            [ 0.20119521,  0.94433112],
            [-0.71858376,  0.75646708],
            [ 0.15703781,  0.17211214],
            [-0.549251  ,  0.8359098 ],
            [ 0.42181883, -0.4194758 ],
            [-0.58221687,  0.92532465],
            [-0.00985648,  0.02182334],
            [ 1.9085485 , -0.04778112],
            [ 1.07890604, -0.55962958],
            [ 0.63886865,  0.85688863],
            [-0.96253176,  0.42724354],
            [ 2.01119727,  0.30994771],
            [ 1.3381743 , -0.46932677],
            [ 1.71051682, -0.34670448],
            [ 0.01205272,  0.4987428 ],
            [-0.92555797,  0.57155343],
            [ 1.10364816,  0.02839193],
            [ 0.63038566, -0.36777248],
            [-0.20807982,  1.21081344],
            [ 0.87932214,  0.23127916],
            [-0.39734699,  0.91105029],
            [-0.1008072 ,  0.98242972],
            [ 1.21794638, -0.49567763],
            [ 2.07688762,  0.34178425],
            [ 0.31217061,  0.97998039],
            [ 0.84985857, -0.43038397]])
```

We will now create a `MinMaxScaler` and fit it to our feature data.

Each transformer has different information that it needs in order to perform a transformation. In the case of the `MinMaxScaler`, the smallest and largest values in the data are needed.

```
[ ]: from sklearn.preprocessing import MinMaxScaler

     transformer = MinMaxScaler()
     transformer.fit(features)




     #arrays correspond to one feature
     transformer.data_min_, transformer.data_max_
```

```
[ ]: (array([-1.14624884, -0.60601875]), array([2.08693816, 1.21081344]))
```

You might notice that the values are stored in arrays. This is because transformers can operate on more than one feature. In this case, however, we have only one.

Next, we need to apply the transformation to our features. After the transformation, we can now see that all of the features fall between the range of zero to one. Moreover, you might notice that the minimum and maximum value in the untransformed `features` array correspond to the 0 and 1 in the transformed array, respectively.

```
[ ]: features = transformer.transform(features)
     features
```

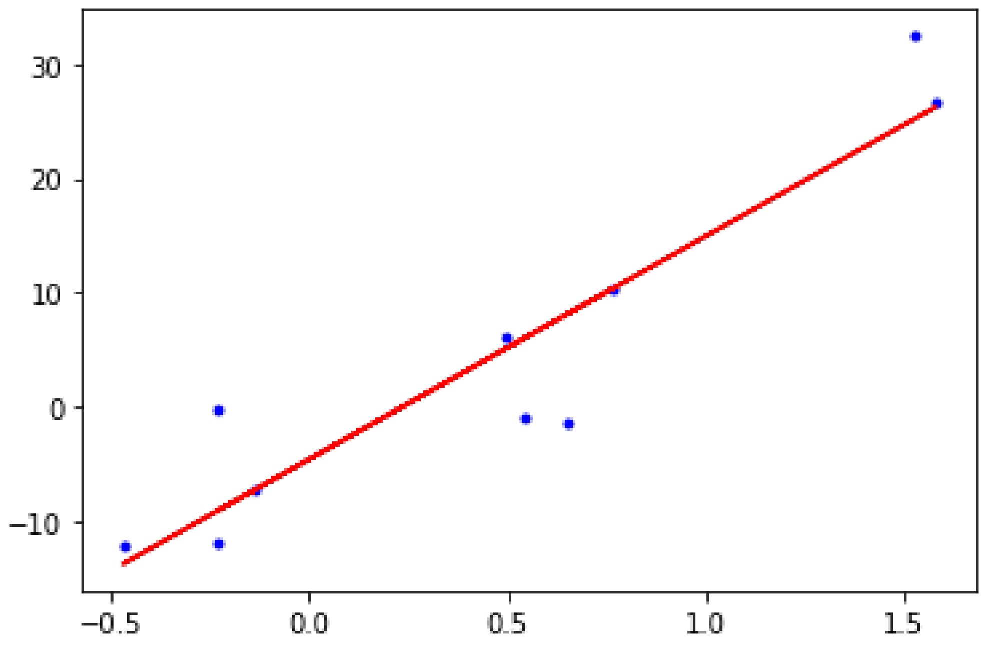
```
[ ]: array([[0.65687414, 0.55083466],
            [0.90565183, 0.21339364],
            [0.55572095, 0.77364952],
            [0.64023721, 0.28741735],
            [0.        , 0.32134609],
            [0.81880726, 0.07736281],
            [0.12937322, 0.74475742],
            [0.51440755, 0.83651519],
            [0.92347597, 0.37711709],
            [0.90450656, 0.14609746],
            [0.07343884, 0.50515851],
            [0.39331197, 0.30788452],
            [0.50974137, 0.13650137],
            [0.36610082, 0.91637992],
            [0.55393625, 0.68559652],
            [0.24501706, 0.80825668],
            [0.63856367, 0.56117638],
            [0.12093494, 0.43087662],
            [0.65759457, 0.07751691],
            [0.92605253, 0.36369723],
            [0.39913144, 0.55201387],
[0.37175723, 0.50955235],
[0.39313261, 0.8616829 ],
[0.37088645, 0.2904705 ],
[0.44699684, 0.2537512 ],
[0.06751852, 0.59292019],
[0.62579973, 0.63799729],
[0.75000976, 0.01296629],
[0.43794241, 0.17327686],
[0.04179931, 0.69578803],
[0.57153153, 0.11099223],
[0.50809317, 0.09466471],
[0.33502514, 0.81764959],
[0.80958418, 0.13362367],
[0.15618987, 0.7482098 ],
[1.        , 0.65301446],
[0.4743078 , 0.82869146],
[0.65514067, 0.        ],
[0.66164329, 0.5629598 ],
[0.43010422, 0.7964525 ],
[0.33677025, 0.56951081],
[0.55874997, 0.65792902],
[0.96526476, 0.57744664],
[0.62125761, 0.6183145 ],
[0.05791275, 0.35748918],
[0.9345302 , 0.28761976],
[0.67023078, 0.0778492 ],
[0.41338548, 0.19786954],
[0.99395651, 0.58874533],
[0.41675413, 0.85332585],
[0.13227354, 0.74992387],
[0.40309659, 0.4282899 ],
[0.18464686, 0.79364982],
[0.48499133, 0.10267484],
[0.17445077, 0.84286453],
[0.35147746, 0.34556966],
[0.94482544, 0.30725877],
[0.68822338, 0.02553299],
[0.55212318, 0.80519675],
[0.05682229, 0.56871641],
[0.97657392, 0.50415579],
[0.76841307, 0.07523643],
[0.88357576, 0.14272879],
[0.35825381, 0.60807022],
[0.068258  , 0.64814581],
[0.69587592, 0.34918507],
[0.54949946, 0.13113279],
[0.2901685 , 1.        ],
[0.6264936 , 0.46085594],
[0.23162961, 0.83500779],
[0.3233471 , 0.87429564],
[0.73122749, 0.0607327 ],
[0.99689145, 0.52167889],
[0.45107798, 0.87294751],
[0.61738075, 0.09667088]])
```

1.2.3 Pipelines

A pipeline is simply a series of transformers, often with an estimator at the end.

In the example below, we use a Pipeline class to perform min-max scaling or our feature data and then train a linear regression model using the scaled features.

```
[ ]: from sklearn.pipeline import Pipeline
     features, targets = make_regression(
         n_samples=10, n_features=1, random_state=42, noise=5.0)

     pipeline = Pipeline([
       ('scale', MinMaxScaler()),
       ('regression', LinearRegression())
     ])

     pipeline.fit(features, targets)

     predictions = pipeline.predict(features)

     plt.plot(features, targets, 'b.')
     plt.plot(features, predictions, 'r-')
     plt.show()
```



1.2.4 Metrics

So far we have seen ways that scikit-learn can help you get data, modify that data, train a model, and finally, make predictions. But how do we know how good these predictions are?

Scikit-learn also comes with many functions for measuring model performance in the metrics package. Later in this course, you will learn about different ways to measure the performance of regression and classification models, as well as tradeoffs between the different metrics.

We can use the mean_squared_error function to find the mean squared error (MSE) between the target values that we used to train our linear regression model and the predicted values.

[ ]:
```
from sklearn.metrics import mean_squared_error

mean_squared_error(targets, predictions)
```

[ ]:
```
27.955458970333645
```

In this case, the MSE value alone doesn't have much meaning. Since the data that we fit the regression to isn't related to any real-world metrics, the MSE is hard to interpret alone.

As we learn more about machine learning and begin training models on real data, you'll learn how to interpret MSE and other metrics in the context of the data being analyzed and the problem being solved.

There are also metrics that come with each estimator class. These metrics can be extracted using the `score` method.

The `regression` class we created earlier can be scored, as can the `pipeline`.

The return value of the `score` method depends on the estimator being used. In the case of `LinearRegression`, the score is the R^2 score, where scores closer to 1.0 are better. You can find the metric that `score` returns in the documentation for the given estimator you're using.

1.2.5 Exercise 4

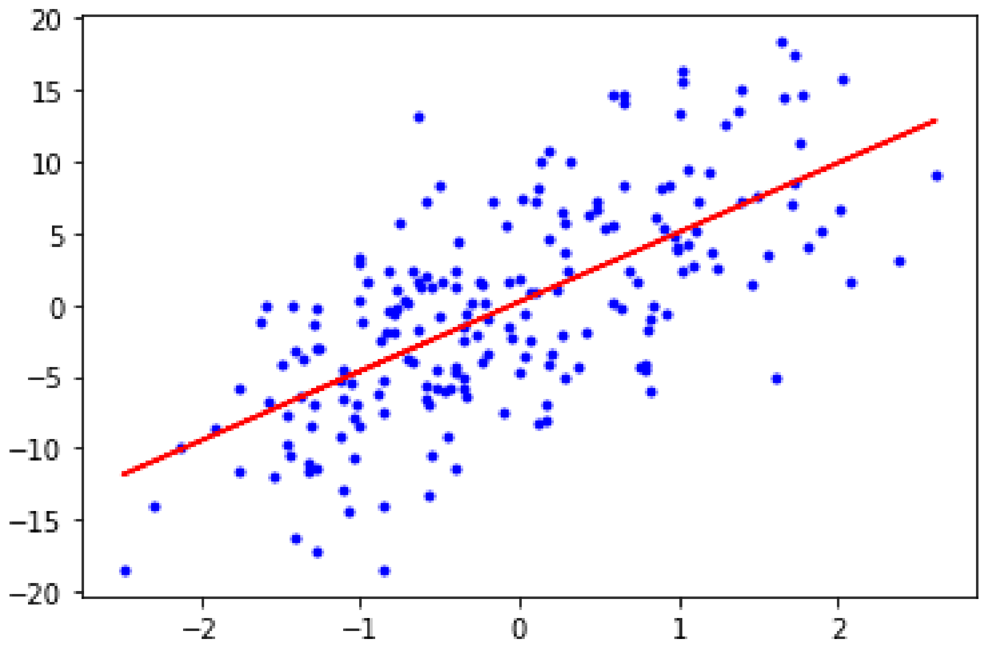
Use the Pipeline class to combine a data pre-processor and an estimator.

To accomplish this:

1. Find a preprocessor that uses the max absolute value for scaling.
2. Find a linear_model based on the Huber algorithm.
3. Combine this preprocessor and estimator into a pipeline.
4. Make a sample regression dataset with 200 samples and 1 feature. Use a random state of 85 and a noise of 5.0. Save the features in a variable called `features` and the targets in a variable called `targets`.
5. Fit the model.
6. Using the features that were created when the regression dataset was created, make predictions with the model and save them into a variable called `predictions`.
7. Plot the features and targets used to train the model on a scatterplot with blue dots.
8. Plot the features and predictions over the scatterplot as a red line.

Student Solution

```
[ ]: from sklearn.preprocessing import MaxAbsScaler
     from sklearn.pipeline import Pipeline
     from sklearn.linear_model import HuberRegressor


     pipeline = Pipeline([
       ('scale', MaxAbsScaler()),
       ('regression', HuberRegressor())
     ])


     features, targets = make_regression(n_samples=200, n_features=1,␣
      ↪random_state=85,noise =5.0)
     pipeline.fit(features, targets)
     predictions = pipeline.predict(features)
     plt.plot(features, targets, 'b.')
     plt.plot(features, predictions, 'r-')
     plt.show()
```



1.3 Conclusion

Scikit-learn is a widely used library that contains scores of resources for performing machine learning. In this Colab, we have introduced some basic concepts that you will see repeated throughout your career in data science. We will cover other parts of scikit-learn later in the course.

You are also encouraged to check out the scikit-learn documentation, where you will find a user guide, tutorials, and a full API reference.

Scikit-learn is an open source project. You can find it on Github here.

1.4 Resources

- https://scikit-learn.org/stable/documentation.html
- https://en.wikipedia.org/wiki/Scikit-learn
- https://en.wikipedia.org/wiki/Estimator
- https://en.wikipedia.org/wiki/Mean_squared_error
- https://github.com/scikit-learn/scikit-learn