Importing and Wrangling Data in Python

Aims of the Lecture

- Learn how to import numerical data to Python from different sources.
- Understand how to select certain parts of the imported data.

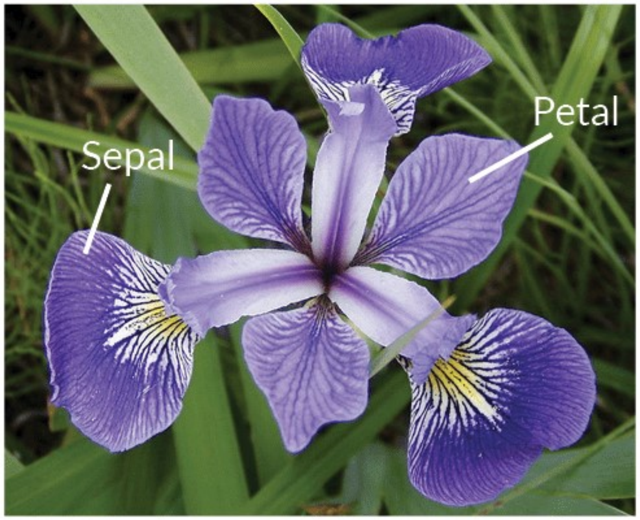
Example

Loading Data from a Module

- Python has a module called **scikit-learn** or *sklearn* which contains several datasets commonly used in data science and business analytics.

- For this excercise, we will use the **IRIS** database contained in this module.

- This dataset contains the sepal and petal lengths and widths from 150 samples of 3 different types of the iris flower.



Iris Versicolor

Iris Setosa

Iris Virginica

- Unlike last week, we will **NOT** work with the actual images, but rather with the numerical information extracted from samples.

- First, we need to install **sklearn**:

In [1]:
```
!pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\programdata\anaconda3\envs\foo\li
b\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\envs\f
oo\lib\site-packages (from sklearn) (0.19.1)

WARNING: You are using pip version 19.2.3, however version 19.3.1 is availabl
e.
You should consider upgrading via the 'python -m pip install --upgrade pip' co
mmand.
```

- Then, we can load the iris dataset:

In [2]:
```
## Load iris dataset
from sklearn import datasets
iris = datasets.load_iris()
print(type(iris))
```

```
<class 'sklearn.utils.Bunch'>
```

- The dataset is contained on a **dictionary-like** structure referred to as **sklearn.utils.Bunch**.

- If you print it, you will see a lot of things contained:

In [3]:

```python
print(iris)
```

```
{'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, 0.1],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.1, 1.5, 0.1],
```

- Therefore, we need to extract each index of this dictionary into a different variables to understand and analyse them separately.

- First, we will import the data:

In [4]:
```python
data = iris['data']
print(data, type(data), data.shape)
```

```
[[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 0.1]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
```

- The data is stored in a *numpy array* of 150 rows and 4 columns, each corresponding to the measurements of a flower.

- Then, we will import the headers of the data:

In [5]:
```
header = iris['feature_names']
print(header, type(header))
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)'] <class 'list'>
```

- **Why do you think the data and the header are stored separately?**

- Afterwards, we will import the **class/target**:

In [6]:
```
target = iris['target']
print(target, type(target), target.shape)
```

```
[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] <class 'numpy.ndarray'> (150,)
```

- The class/target is a *numpy array* which contains the **category** of each flowers.

- Each sample is labelled as 0, 1 or 2 instead of the iris type since the labels can be better used as numbers.

- A separate key called **target_names** contains the name corresponding to each numerical label.

In [7]:
```
target_names = iris['target_names']
print(target_names, type(target_names), target_names.shape)
```

```
['setosa' 'versicolor' 'virginica'] <class 'numpy.ndarray'> (3,)
```

- Finally, just in case you are interested, there is an entry containing the description of the dataset (a string):

In [8]:
```
iris['DESCR']
```

Out[8]:
```
'Iris Plants Database\n====================\n\nNotes\n-----\nData Set Characte
ristics:\n    :Number of Instances: 150 (50 in each of three classes)\n    :Nu
mber of Attributes: 4 numeric, predictive attributes and the class\n    :Attri
bute 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-Virg
inica\n    :Summary Statistics:\n\n    ============== ==== ==== ======= =====
====================\n                    Min  Max   Mean    SD   Class Correl
ation\n    ============== ==== ==== ======= ===== ====================\n    se
pal 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 At
tribute 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\nThis is a copy of UCI ML iris datasets.\nhtt
p://archive.ics.uci.edu/ml/datasets/Iris\n\nThe famous Iris database, first us
ed by Sir R.A Fisher\n\nThis is perhaps the best known database to be found in
the\npattern recognition literature.  Fisher\'s paper is a classic in the fiel
d 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; t
he\nlatter are NOT linearly separable from each other.\n\nReferences\n
----------\n   - Fisher,R.A. "The use of multiple measurements in taxonomic pr
oblems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributi
ons 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 Class
```

Wrangling Data

- Accessing an individual entry of the dataset (along with its class/target):

In [9]:
```
print(data[0], target[0])
```

```
[5.1 3.5 1.4 0.2] 0
```

- Creating a table for each iris type ("manually")

In [10]:

```
setosa = data[0:50]
print(setosa, setosa.shape)
```

```
[[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]
```

In [11]:

```
## Use this cell to create and print versicolor and virginica (with the shape)
versicolor = data[50:100]
print(versicolor, versicolor.shape)
virginica = data[100:150]
print(virginica, virginica.shape)
```

```
[[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]
```

- Creating a table for each iris type ("automatically")

In [12]:

```python
## In case that data is not in order or you don't want to count,
## we can use this alternative:
import numpy as np
setosa2 = data[np.where(target==0)]
print(setosa, setosa.shape)
```

```
[[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]
```

```
In [13]: ## Verify that we get the same
         setosa == setosa2

Out[13]: array([[ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
                [ True,  True,  True,  True],
```

- Creating a new table with "less" columns (by column number):

In [14]:
```
## creating a "reduced" table
## with ony the first two columns
data_red1 = data[:,:2]
print(data_red1,data_red1.shape)
```

```
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]
 [5.4 3.9]
 [4.6 3.4]
 [5.  3.4]
 [4.4 2.9]
 [4.9 3.1]
 [5.4 3.7]
 [4.8 3.4]
 [4.8 3. ]
 [4.3 3. ]
 [5.8 4. ]
 [5.7 4.4]
 [5.4 3.9]
 [5.1 3.5]
 [5.7 3.8]
 [5.1 3.8]
 [5.4 3.4]
 [5.1 3.7]
 [4.6 3.6]
 [5.1 3.3]
 [4.8 3.4]
 [5.  3. ]
 [5.  3.4]
 [5.2 3.5]
 [5.2 3.4]
 [4.7 3.2]
 [4.8 3.1]
 [5.4 3.4]
 [5.2 4.1]
 [5.5 4.2]
 [4.9 3.1]
```

In [15]:

```
## Use this cell to create a new dataset called data_red2
## with the last two columns
data_red2 = data[:,2:]
print(data_red2,data_red2.shape)
```

```
[[1.4 0.2]
 [1.4 0.2]
 [1.3 0.2]
 [1.5 0.2]
 [1.4 0.2]
 [1.7 0.4]
 [1.4 0.3]
 [1.5 0.2]
 [1.4 0.2]
 [1.5 0.1]
 [1.5 0.2]
 [1.6 0.2]
 [1.4 0.1]
 [1.1 0.1]
 [1.2 0.2]
 [1.5 0.4]
 [1.3 0.4]
 [1.4 0.3]
 [1.7 0.3]
 [1.5 0.3]
 [1.7 0.2]
 [1.5 0.4]
 [1.  0.2]
 [1.7 0.5]
 [1.9 0.2]
 [1.6 0.2]
 [1.6 0.4]
 [1.5 0.2]
 [1.4 0.2]
 [1.6 0.2]
 [1.6 0.2]
 [1.5 0.4]
 [1.5 0.1]
 [1.4 0.2]
 [1.5 0.1]
```

In [16]:

```
## Use this cell to create a new dataset called data_red3
## with the first and the third columns
data_red3 = data[:,[0,2]]
print(data_red3,data_red3.shape)
```

```
[[5.1 1.4]
 [4.9 1.4]
 [4.7 1.3]
 [4.6 1.5]
 [5.  1.4]
 [5.4 1.7]
 [4.6 1.4]
 [5.  1.5]
 [4.4 1.4]
 [4.9 1.5]
 [5.4 1.5]
 [4.8 1.6]
 [4.8 1.4]
 [4.3 1.1]
 [5.8 1.2]
 [5.7 1.5]
 [5.4 1.3]
 [5.1 1.4]
 [5.7 1.7]
 [5.1 1.5]
 [5.4 1.7]
 [5.1 1.5]
 [4.6 1. ]
 [5.1 1.7]
 [4.8 1.9]
 [5.  1.6]
 [5.  1.6]
 [5.2 1.5]
 [5.2 1.4]
 [4.7 1.6]
 [4.8 1.6]
 [5.4 1.5]
 [5.2 1.5]
 [5.5 1.4]
 [4.9 1.5]
```

In [17]:

```
## creating a "reduced" table with ony the first column
col_0 = data[:,0]
print(col_0,col_0.shape)
```

```
[5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.
 5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9 5.5
 6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
 6.3 6.1 6.4 6.6 6.8 6.7 6.  5.7 5.5 5.5 5.8 6.  5.4 6.  6.7 6.3 5.6 5.5
 5.5 6.1 5.8 5.  5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.  6.9 5.6 7.7 6.3 6.7 7.2
 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.  6.9 6.7 6.9 5.8 6.8
 6.7 6.7 6.3 6.5 6.2 5.9] (150,)
```

- Getting a column by it's name:

In [18]:
```
sepal_length = data[:,header.index('sepal length (cm)')]
print(sepal_length,sepal_length.shape)
```

```
[5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.
 5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9 5.5
 6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
 6.3 6.1 6.4 6.6 6.8 6.7 6.  5.7 5.5 5.5 5.8 6.  5.4 6.  6.7 6.3 5.6 5.5
 5.5 6.1 5.8 5.  5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.  6.9 5.6 7.7 6.3 6.7 7.2
 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.  6.9 6.7 6.9 5.8 6.8
 6.7 6.7 6.3 6.5 6.2 5.9] (150,)
```

Importing YOUR data

- For the coursework output 2, you will need to import the data from a **.csv** file.

- For instance, the IRIS dataset would look something like this:

Fig 1: Iris dataset in csv

- Your datasets will have a **first column** with the id of each entry (**NOT** the same as the row index).
- Your dataset will have the class/target in the **last column**.
- The **first row** contains the header.

- You need to find a pre-existing module that lets you import data from a csv file into a numpy array.
- Try to import the header in a different variable as the data.
- Since the classes/targets are numeric for all datasets, you can leave them on the same numpy array as the data.
- You don't need the target names, just work with the numbers!