# Product Recommendation - Wine Store

Module 0 | Chapter 2 | Notebook 1

---

In this notebook you'll get to know the `numpy` module a bit better, and use its basic data structure, the `ndarray`. We will cover the properties and calculation methods of arrays. Then we will explore a data set describing the characteristics of red and white wines. By the end of this exercise you will be able to calculate using arrays and you will understand the link to DataFrames from the `pandas` module. We will use arrays and the wine data set over the course of this chapter as a basis for learning useful linear algebra methods.

---

## First steps with `numpy`

**Scenario:** 1001Wines is an online retailer that sells wines through its website. 1001Wines would like to recommend its customers high-quality wines that might be to their taste. They therefore want to create a recommendation tool to compare the similarity of purchased wines with other wines from the assortment. If a wine from the range is similar to what customers have already ordered, it should be recommended to them.

In order to be able to complete this task, 1001Wines has provided us with a wine data set. This includes various measured characteristics of each kind of wine, from the alcohol volume to the sugar level. We'll look at the data more closely later.

First let's focus on the `numpy` module. It provides us with some tools that we can use to solve this task. Furthermore, you have already used `numpy` in many other modules, perhaps without realizing it. `numpy` is very common and offers many basic mathematical functions and optimized data structures, so that you can carry out calculations in `numpy` very quickly. Many other modules are based on it for this reason. For example, you will see how `numpy` and `pandas` are related.

Begin by importing `numpy` with its conventional alias `np`.

In [1]:
```
import numpy as np
```

The main data structure we need for this chapter is the `ndarray`. An array is very similar to a `list`. It contains data that appears in a certain order. However, an `Array` allows us to perform numerous arithmetical operations. Use `np.array(my_list)` to create two arrays from the lists `[5, 10, 20]` and `[1, 2, 4]`. Name the arrays `arr_1` and `arr_2`. Print the result of the four basic arithmetic operations between the two arrays.

In [2]:
```
arr_1= np.array([5,10,20])
arr_2= np.array([1,2,4])
print(arr_1 + arr_2)
print(arr_1 - arr_2)
print(arr_1 * arr_2)
print(arr_1 / arr_2)
```

```
[ 6 12 24]
[ 4  8 16]
[ 5 20 80]
[5. 5. 5.]
```

As you can see, when it comes to the basic arithmetic operations, the calculations are performed on the elements individually and their corresponding values, depend on the order. Now test the basic arithmetic for one of the two arrays with the number `3.5`.

**Important:** Some arithmetic operations produce different results when they are carried out on `lists` and `np.ndarray`. You can also try out the operations with lists to see what happens.

In [3]:
```
print(arr_1 + 3.5)
print(arr_1 - 3.5)
print(arr_1 * 3.5)
print(arr_1 / 3.5)
```

```
[ 8.5 13.5 23.5]
[ 1.5  6.5 16.5]
[17.5 35.  70. ]
[1.42857143 2.85714286 5.71428571]
```

If you calculate with arrays and single numbers, this arithmetic operation is applied to each element in the array. You already know this behavior from the `Series` and `DataFrame` data types from `pandas`. This is because `ndarrays` are used as the basis for storing the data. But unlikewith a `Series` or `DataFrame`, you can't use `my_array.loc[]` and `my_array.iloc[]` to access entries in an `array`. You access values in an `array` the same way as in a `list`. `my_array[0]` returns the first element of `my_array`. You can imagine that an `ndarray` is something between a `list` and a `DataFrame`. You can perform arithmetic operations on it, but there are no row names or column names. In `numpy` you will also find other functions and methods that you already know from lists or from `pandas`. However, these methods from the module are not usually available as methods for arrays. In the following cell you will find a short overview. Run the cell.

In [4]:
```
#similar to lists:
print('Select first element of array with brackets:', arr_1[0])
print('Append new element to array using np.append():', np.append(arr_1,22.22))
print('Delete last element of array using np.delete():', np.delete(arr_1,-1)) #You car

print() #add empty line for better reading

#similar to pandas:
print('Calculate mean of array using np.mean():', np.mean(arr_1))
print('Calculate the median using np.median()', np.median(arr_1))
print('Calculate the median using np.quantile()', np.quantile(arr_1,0.5))
```

```
Select first element of array with brackets: 5
Append new element to array using np.append(): [ 5.   10.   20.   22.22]
Delete last element of array using np.delete(): [ 5 10]

Calculate mean of array using np.mean(): 11.666666666666666
Calculate the median using np.median() 10.0
Calculate the median using np.quantile() 10.0
```

**Important:** `np.append()` and `np.delete()` do not modify the arrays directly. This means you have to assign the result of the method back to the `array` if you want to store the change. This is different for a `list`. This is modified *in place*, which means it is changed directly, so you don't have to assign the result back to the variable.

**Congratulations:** You have now got a feeling for arrays. Now let's take a look at the data set.

# The data set

The wine data set is located in the file *wine-quality.csv*. Data in a CSV file is organized in rows and columns. A comma is usually used to separate the values. That's where the file format gets its name from: CSV stands for *comma separated values*. `pandas` makes it very easy to import this kind of data. It offers us the function `pd.read_csv()`. The most important parameter of this function is `filepath_or_buffer`, which is the first one and should contain the file name. Sometimes something other than a comma is used as separator, for example a semicolon. The function has a lot more parameters that we don't need so often. If you like, you can take a look at the documentation to get an overview. Then you pass the separator as a *string* to the parameter `sep` (short for separator). `pd.read_csv()` returns a `DataFrame` which is convenient to work with.

Import `pandas` with its conventional alias `pd`. Import the file and store it as a `DataFrame` named `df`. Display the first 5 rows of `df`. Tip: Since the `filepath_or_buffer` parameter comes first, you don't need to write it out in full, `pd.read_csv(my_filename)` will do.

In [5]:
```
import pandas as pd
df = pd.read_csv('wine-quality.csv')
df.head()
```

Out[5]:

| | fixed.acidity | volatile.acidity | citric.acid | residual.sugar | chlorides | free.sulfur.dioxide | total.sulfur.dio |
|---|---|---|---|---|---|---|---|
| **0** | 7.0 | 0.27 | 0.36 | 20.7 | 0.045 | 45.0 | 1 |
| **1** | 6.3 | 0.30 | 0.34 | 1.6 | 0.049 | 14.0 | 1 |
| **2** | 8.1 | 0.28 | 0.40 | 6.9 | 0.050 | 30.0 | |
| **3** | 7.2 | 0.23 | 0.32 | 8.5 | 0.058 | 47.0 | 1 |
| **4** | 7.2 | 0.23 | 0.32 | 8.5 | 0.058 | 47.0 | 1 |

As you can see, each row contains chemical properties of a wine, such as the pH level or alcohol content. We have already learned that `pandas` uses the `ndarray` internally. The `array` is

stored in the attribute `my_df.values`. Check the data type of `df.values`.

In [6]:
```
type(df.values)
```

Out[6]:
```
numpy.ndarray
```

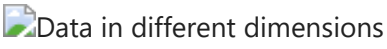
It has the data type `ndarray`. This is how `numpy` arrays are described. So the values in a `pandas` `DataFrame` are actually stored as arrays from `numpy`. This is because `numpy` arrays have already been highly optimized and why many packages are compatible with arrays.

`nd` stands for n-dimensional. This means that an `ndarray` in `numpy` can have any number of dimensions. Dimensions are the number of directions along which the data is arranged. In the `DataFrame` there are 2 dimensions. We have the vertical columns and horizontal rows. If we just take one column, it has only one dimension, since we are only looking at one vertical column. Now let's try that out. The attribute `my_array.ndim` will be helpful here. This will tell you the number of dimensions. Print `df.values.ndim` and `df.loc[:,'pH'].values.ndim`.

In [7]:
```
df.ndim
df.values.ndim
df.loc[:,'pH'].values.ndim
```

Out[7]:
```
1
```

As we mentioned above, the `array` for the `DataFrame` has two dimensions and the `array` for a column (i.e. a `Series`) has one dimension. If an `array` has one dimension, it is also called a vector. Two dimensions are called a matrix. This is visualized here once again:


Data in different dimensions

Now you might ask yourself when you need more than 2 dimensions. For example, if we recorded the measurement data of the wines every year, then we could use a separate 2-dimensional array for each year or we could use a 3-dimensional array. Then the 2-dimensional arrays are arranged one behind the other, as shown in the illustration. The values within arrays can be accessed in a the same way as you would use `my_df.iloc[]` to access values in DataFrames, but you don't use the dot and four letters `.iloc`, so you just use: `df.values[row_num, col_num]`. Each dimension needs a new number. Try it out. Which value is located at `df.values[1, 8]`?

In [8]:
```
df.values[1, 8]
```

Out[8]:
```
3.3
```

The **rows before columns** principle also applies here.

**Congratulations:** You now know what we mean by dimensions in data structures and what the words vector and matrix mean. You will come across these terms very often in the field of data science. For example, we often call the data used by a machine learning model to make predictions a feature matrix. This is the matrix that contains the feature data, the characteristics

you want to use to make the predictions. Vectors and matrices are the basic data structures for linear algebra. In the next notebooks, we will take a much closer look at vectors and use them to find a solution for our task.

**Remember:**

- `numpy` offers many features that you already know from using DataFrames.
- Access an element in an `ndarray` like a `DataFrame` but without `.iloc` - `my_array[row_num, col_num]`.
- Imagine the dimensions of an array as axes on which the data is arranged.
- A vector has one dimension. A matrix has two dimensions.

---

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

---

Found a mistake? Contact Support at support@stackfuel.com.

---

The data was published here first: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

---