Python_basics

April 7, 2020

1 Python basics

1.0.1 General comments

The first step in every Python script is to load those packages that we'll use during the analysis. A package is a set of tools that are not included in the built-in Python tools.

There are four packages that are commonly used and we will usually load: * **NumPy** is a fundamental package for scientific computing that includes N-dimensional array objects, linear algebra, Fourier transforms, random number capabilities… **NumPy** uses a vector structure called *array*; data in an *array* must be always of the same nature, i.e., integer, floating point number, string… To import **NumPy**, use the following command: > `Python import numpy as np`

- **pandas** is a pacakge that allows organizind data in a structure named *data frame*. *Data frames* resemble the usual Excel table, in the sense that columns represent variables and rows represent samples. All the elements of a column (variable) must be of the same nature (integer, string…), but different columns may differ in the type of data they contain. As Excel talbes, a *data frame* has an index and heading that identifies rows and columns, respectively, that allow us to search for specific values. To import **pandas**, use the following command: > "'Python import pandas as pd

```
* __[matplotlib](https://matplotlib.org/)__ is a package designed to plot graphs similar to tho
> ```Python
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')
```

- **SciPy** contains several numerical tools that are efficient and easty to apply, e.g., numerical integration and optimization. We will not load the complete set of tools in **SciPy**, but those we need: > "'Python from scipy.stats import genextreme from scipy.optimize import curve_fit

```
* [__os__](https://docs.python.org/3.4/library/os.html) is a package that allows us to change t
> ```Python
import os
```

[1]:
```
import numpy as np

import pandas as pd

from matplotlib import pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

from scipy.stats import genextreme
from scipy.optimize import curve_fit

import os
```

In case you need to install some of those packages, you'll need to do the following (example to install SciPy): * Launch Anaconda Prompt * Type `conda install scipy` + `Enter`

We're going to install a variable inspector to be able to check the existing objects in our analysis: * Launch Anaconda Prompt * Type: > `pip install jupyter_contrib_nbextensions` + `Enter` `jupyter contrib nbextension install --user` + `Enter` `jupyter nbextension enable varInspector/main` + `Enter`

1.0.2 Basic data structures in Python

Lists Lists are a data structure that can contain data of any type (integer, float, strings…) in a single object. Lists are mutable, meaning that we can modify the values inside a list after its declaration.

[2]:
```
# create a list
a = [1, 'hello', 1.5]
```

[3]:
```
# extract a value from the list
a[1]
```

[3]:
```
'hello'
```

[4]:
```
# modify one of the values in the list
a[1] = 'bye'
```

Tuples Tuples are a data structure similar to lists because they can also contain data of any type. Contrary to lists, tuples can no be modified after declared.

[5]:
```
# create a lista
b = (2, 'red', np.nan)
```

[6]:
```
# extract a value from the tuple
b[2]
```

[6]:
```
nan
```

[7]:
```
# modify one of the values in the tuple
b[2] = 0
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-7-98dffcf346e8> in <module>
  1 # modify one of the values in the tuple
----> 2 b[2] = 0


TypeError: 'tuple' object does not support item assignment
```

Arrays This is a specific structure of the package *NumPy* that allows us to work with vectores and matrices, and perform calculations upon them easily. All the values in an array must be of the same data type.

[8]:
```
# create an array from the list 'a'
a_ = np.array(a)
a_
```

[8]:
```
array(['1', 'bye', '1.5'], dtype='<U11')
```

[9]:
```
# create an array
c = np.array([1.5, 2.1, 4.5])
```

[10]:
```
# extract values from the array
c[1:]
```

[10]:
```
array([2.1, 4.5])
```

[11]:
```
# invert the array
c[::-1]
```

[11]:
```
array([4.5, 2.1, 1.5])
```

[12]:
```
# modify a value in the array
c[2] = 10.3
```

[13]:
```
# calculate the mean of the array
c.mean()
```

[13]:
```
4.633333333333334
```

Pandas: *series* and *data frames* *Pandas* is a package suitable for working with bidimensional (*data frames*) or unidimensional (*series*) tables. Pandas' structures use the tools in *NumPy* to perform easily several tasks with the table. In *Pandas*, all the data contained in a column of the table must be of the same type; different columns may have different types of data.

[14]:
```
# create a 'data frame' with name, age and weight
d = [['Peter', 36, 71],
     ['Laura', 40, 58],
     ['John', 25, 65]]
d = pd.DataFrame(data=d, columns=['name', 'age', 'weight'])
d
```

[14]:
```
    name  age  weight
0  Peter   36      71
1  Laura   40      58
2   John   25      65
```

[15]:
```
# a column in a data frame is a series
d2 = d.name
d2
```

[15]:
```
0    Peter
1    Laura
2     John
Name: name, dtype: object
```

[16]:
```
# calculate the mean of the dataframe
d.mean()
```

[16]:
```
age       33.666667
weight    64.666667
dtype: float64
```

Dictionaries A dictionary can store several data structures (from those above mentioned) in a single object. We need to set a *key* to access any of the data structures included in the dictionary.

[17]:
```
# crear un diccionario que contenga todos los datos anteriormente creados
# siendo la clave el tipo de estructura
# create a dictionary that contains all the data structures previously created
# in this example, the key will be the type of structure
e = {'list': a,
     'tuple': b,
     'array': c,
     'dataframe': d}
```

[18]:
```
# extract one of the structures from the dictionary
e['list']
```

[18]:
```
[1, 'bye', 1.5]
```

[ ]:
```

```