ICT for Health
Laboratory # 1
Regression on Parkinson data
Part 2

Monica Visintin

Politecnico di Torino

2017/18

Table of Contents

Table of Contents

Some useful commands in Spyder [1]

- It is possible to define **portions of code** (*cells*), that can be separately run in the command window. To define a cell, use #%% at the beginning of a line in your script; if the mouse is inside a cell, you can run it by clicking shift enter (or the button with the green arrow, a white rectangle and inside it a light blu rectangle with an orange contour)
- If you run portions of your script, the workspace (that you see on the top right part of the screen, by selecting *Variable explorer*) is updated; if you want to start from scratch and you want to **reset your workspace**, write `% reset` in the command window of Spyder (in Matlab, you would write `clear all`)
- You can view the content of a DataFrame or array or series by double clicking on the variable identifier in the **Variable explorer**

Some useful commands in Spyder [2]

- You can view the **help** of an object by writing the object identifier/instantiation in the command window and clicking Ctrl (the help is shown in the top write part of the screen - **Object inspector**); otherwise if the instantiation is `x`, write `help(x)` in the command window
- If the object/instantiation `x` has a method `meth` , you can get the help by writing `help(x.meth)` in the command window
- If you want to briefly see all the methods and the attributes of an object `x`, write `x.` in the command window and click the **tab key**

Table of Contents

Notes on Python [1]

- Sometimes it might not be so clear when to use `()` (round brackets with nothing inside). In general, round brackets are used for functions/methods; sometimes the functions/methods have no argument, and therefore you have to write `()`. Assume that you have a class `myclass` with attributes `one` (for example a float number), `two` (for example a NumPy Ndarray), `three` (for example a Pandas DataFrame) and methods `methone(self, param1,param2)`, `methtwo(self)`. Assume that `x` is an instantiation of `myclass`. Then `x.one` is a float, `x.two` is an Ndarray, `x.methone(p1,p2)` gives you the result of method `methone` when the parameters are `p1` and `p2`; `x.methtwo()` is the result of method `methtwo` that needs no input parameters (or a method for which you use the default parameter values).

Notes on Python [2]

- However the above distinction might be not enough to make a decision about the use of `()`. For example, assume that `x` is a NumPy Ndarray; to get the transpose of `x`, you have to write `x.T` (without brackets) because `T` is an attribute of `x`, not a method; however, you can imagine that the method `T(self)` were defined for class Ndarray (which is more reasonable, by the way), in which case you would need to write `x.T()`. The author of each class actually decides what is a method and what is an attribute; in some cases, the only way to understand what is correct is to try. If you write `x.T()` in the Spyder command window, you get an error with the following warning `'numpy.ndarray' object is not callable` (because `x.T` is itself an Ndarray, and it is not a callable function/method). On the other side, a Numpy Ndarray has method `argmax`, you should write `x.argmax()`, but if you write `x.argmax` you get `<function ndarray.argmax>` in the command window

Table of Contents

Pandas groupby

- You can see all the methods that can be applied to DataFrames at `https://github.com/jonathanrocher/pandas_tutorial/blob/master/analyzing_and_manipulating_data_with_pandas_manual.pdf`
- One of the possible methods is `groupby`. Assume that `xx` is a Pandas DataFrame, and one of its column is ``` ``subject#'' ```, the numbers in this column are integers and the same integer appears in several rows, then
  `x = xx.groupby(["subject#"],as_index=False)`.mean()
  is a new DataFrame in which each of the possible values of `subject#` appears only once. If for example `subject#=4` appears 10 times in 10 different rows, then the new DataFrame `x` has only one row with `subject#=4` and this row is the mean of the 10 rows of `xx`.
  In the laboratory on Parkinson's disease, you have to use:
  `x = xx.groupby(["subjec#", "test_time"],as_index=False).mean()`

Split and normalize a DataFrame

- The matrix x with the data must be divided into two parts, one for the training phase (first 36 patients) and for the the testing phase (the other patients). One way to split the matrix is the following:

```
index_train=x['subject#']<=36
x_train=x[index_train]
index_test=x['subject#']>36
x_test=x[index_test]
```

  Then `x_train` and `x_test` are now two distinct DataFrames
- It is convenient to normalize the data. This can be obtained as follows:

```
m=x_train.mean()# mean (one value for each column)
s=x_train.std()# standard deviation (one value for each column)
x_train_norm=(x_train-m)/s# normalized training data
x_test_norm=(x_test-m)/s# normalized testing data
#note that m and s are those of the training matrix
```

From Pandas to NumPy

- To process the data (matrix inversion, gradient algorithm, etc) it is convenient to use NumPy. If `x` is a Pandas DataFrame, `x.values` is a NumPy Ndarray