**What every asset management company should know about Artificial Intelligence 1: Setting up a Python Machine Learning platform. Basic dummy english versión.**

What every asset management Company should know about Artificial Intelligence 1

Setting up a Python Machine Learning platform.

Basic dummy version in english.

info@robexia.com

### **Note: robexia.com is an artificial intellingence laboratory that develops consulting and projects, focused on the financial sector.**

## **The series *what every asset management company should know about artificial intellingence* consists on several chapters with an informative versión with limited Access and a basic dummy dummy versión with full access.**

Machine Learning can be understood as a modern statistic that can be extraordinarily powerful, amazing and with spectacular results that we had never imagined possible, but they are still algorithms. With sufficient calculation capacity and adequate knowledge, there are countless code libraries available, in fact, all that we may need.

But, to use them, we will need adequate software. No, it is not valid with excel. We have to install either a free software, such as Python or R or a proprietary one, such as MatLab. Each has its own pros and cons.

Suppose we decided to use Python as the console and calculation engine for AI development in the asset management company. You have to start by installing and configuring it. It is easy? Yes, very very simple, although like everything, when they tell you how to do it. So let's go there.

Let's do a comparison to try and understand Python. If we do not buy a Mac or use Linux, when we buy a computer we must install Windows as an operating system and the programs we need, such as Excel, Word, a pdf file reader, etc. Also, suppose we have several disk drives in addition to C, for example D, E and F and on each disk drive several folders (program files, documents, etc.). Well, if we understand this we can perfectly understand how to install Python. Python would be like installing Windows, the programs the libraries and the drive / folder would be the "environment".

Be careful, it is only a comparison to try to understand it better, it is not real nor does it mean that Python is an operating system, we are simply in the basic dummy version and we want to depart from the "Python is a multiplatform language of dynamic interpreted typing of multiparadigm programming that is object oriented ".

So, do we need Python and libraries? Yes. Python alone is of little use to us. If we continue with the comparison, from our Windows desktop we can do some basic things, such as copying files, but if we want to write a letter or do calculations we access Excel or Word. We do the same with Python. From our Python desktop (console) we can do some basic things, certain calculations, but as soon as we want to do something else we need our programs. In the case of Windows they are Excel, Word, Paint or Acrobat Reader and in Python the libraries, such as NumPy, Pandas or Sckit-Learn. Programs are opened, libraries are imported. Just change the term.

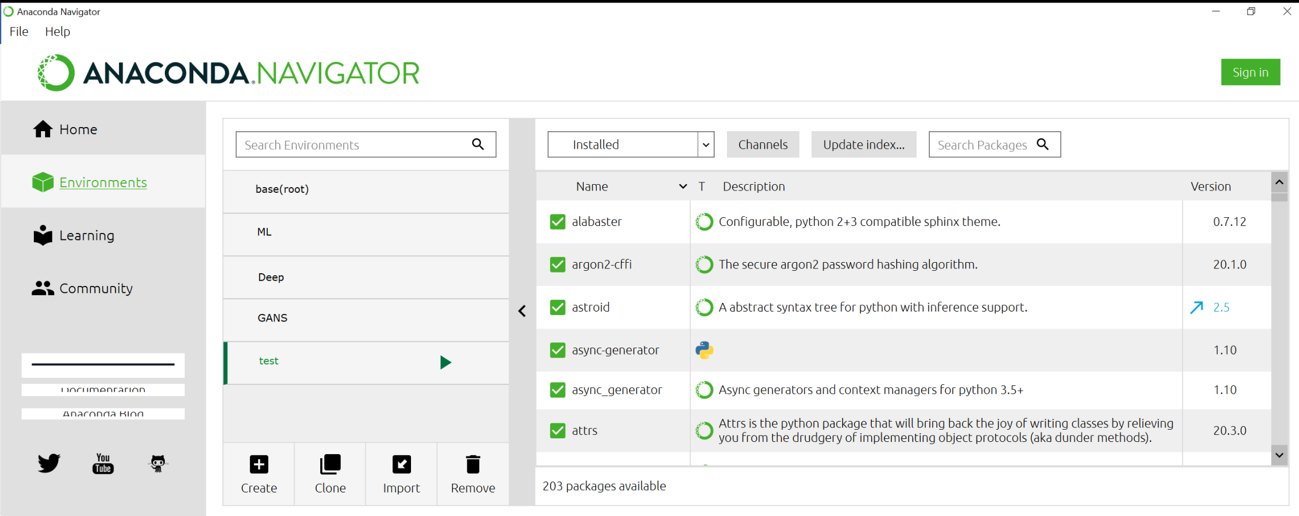
I install Windows I install Python (with Anaconda)

I use it from the desktop I use it from the console (Spyder)

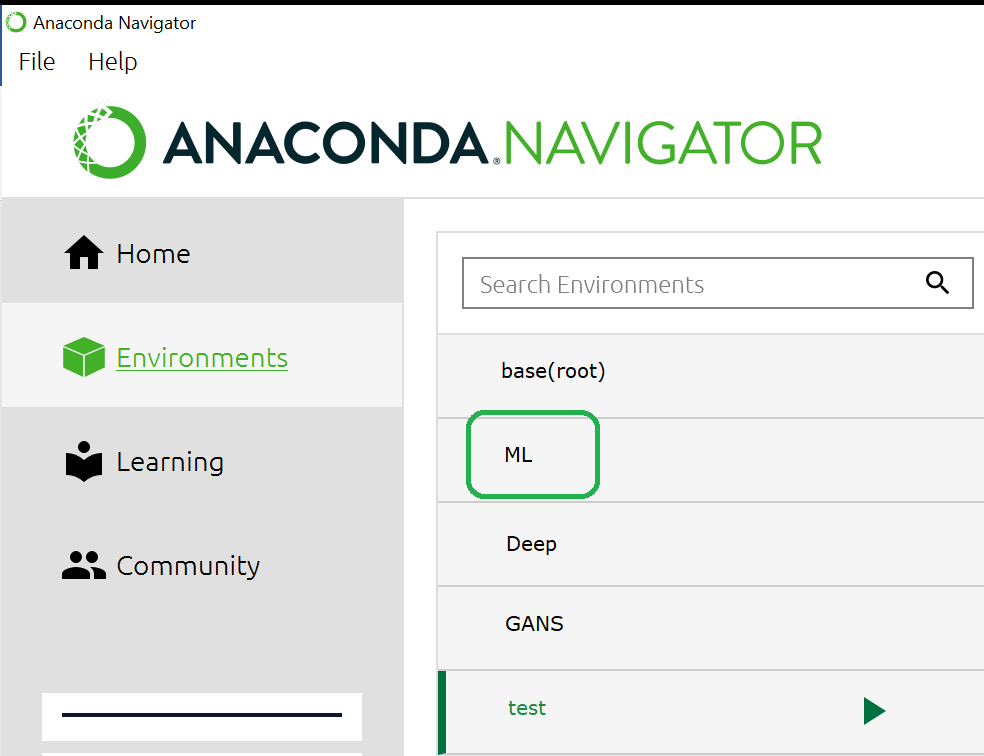
I install the programs I install the libraries

I open programs I import libraries (from the code)

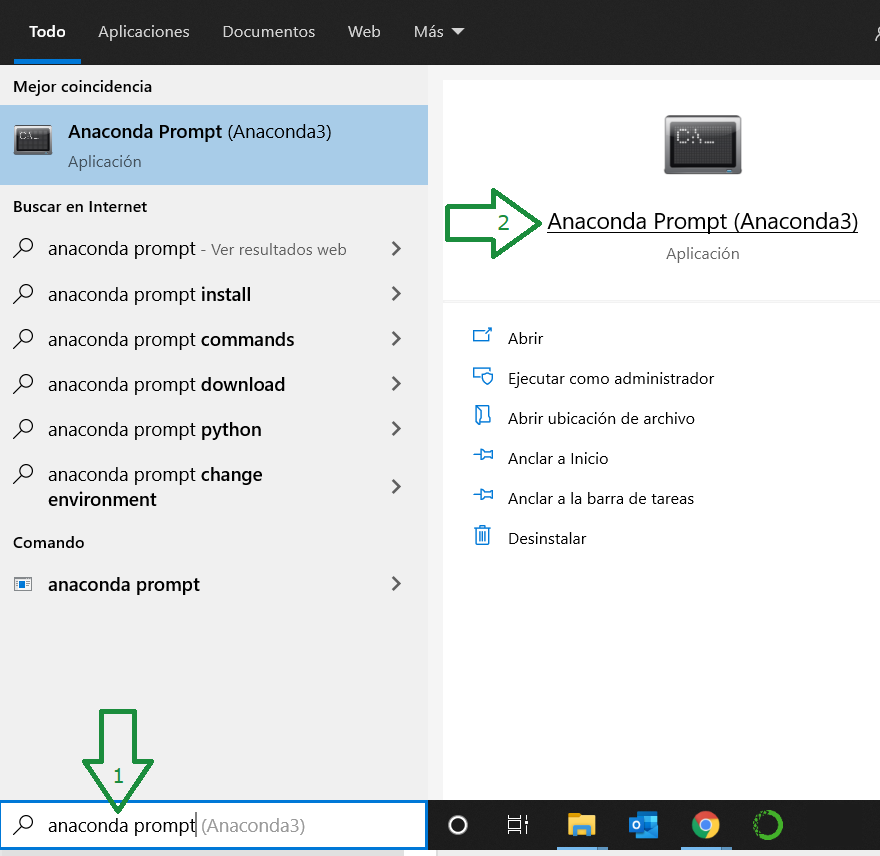
1. **Install Python**: We will use the individual edition of Anaconda, which is free and will help us maintain the system. Once installed, we can use it with Anaconda-Navigator or directly with the prompt (a black window of command lines). Some libraries are already installed by default, let's say it would be as if Windows brought Paint by default, we don't have to install it separately.



2. **Enviroment:** Just as in the computer we have disk drives like C, D or E, where we have the programs, in Python we will use an enviroment with the name we want. Base would be like unit C. But we can create the enviroments that we want, each one with the libraries (as if they were the programs) that we need. We will always work from an environment and we will have the libraries installed there. If we have an enviroment called ML, and we have installed Pandas in it, we will have to open (import) Pandas being located in that enviroment.



3. **Anaconda prompt:** If instead of the browser we prefer to use command lines to manage libraries, we need to open a little black window where to write (just go to Windows search (bottom left) and write Anaconda Prompt. then the orders (commands) of "conda".

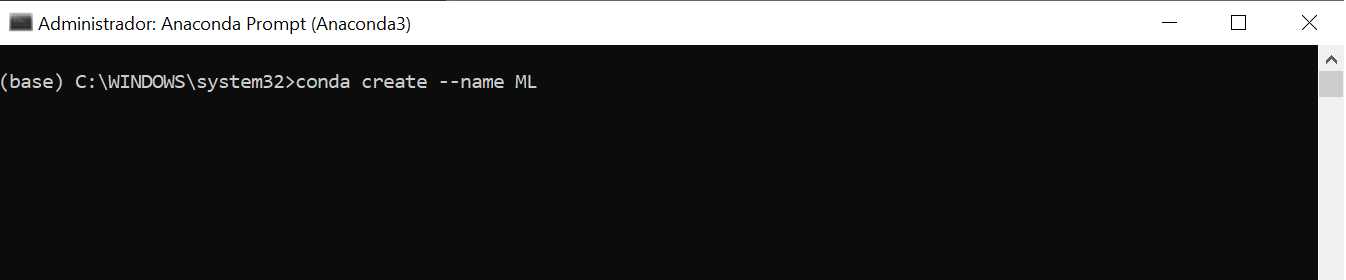


4. Download the **Conda Cheat Sheet** (searching for it in google and using the official conda.io site) and you will have the main orders.

5. **We will prepare what we need**, having our Conda Cheat Sheet at hand. From the prompt (black window) we type:

conda create --name ML

and logically we say yes when you ask whether or not to proceed. We have created our environment ML.



Typing

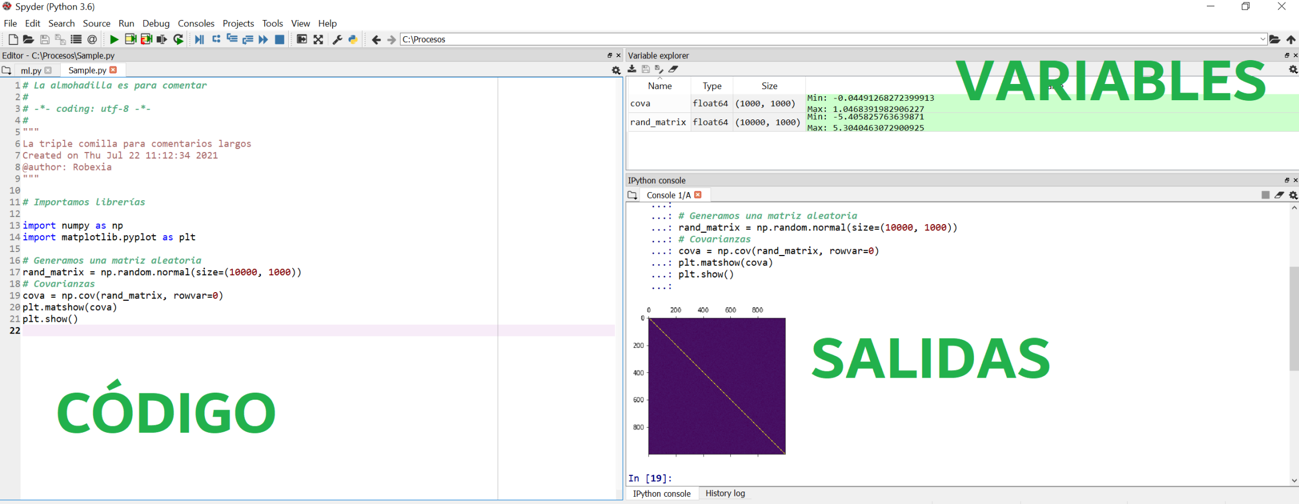
conda activate ML

We enter the ML environment and see that the name in parentheses changes from 'base' to 'ML'.



6. **Code:** When we use Python we will do it based on commands, that is, a series of lines of code that will execute what we tell it, from opening a spreadsheet to calculating the parameters of a neural network. This may be a bit scary at first, but you soon learn the basics. Programming well is difficult, but there is a lot of shared code that we can use if we understand it a little.

We will write the orders somewhere. You can use Jupyter Notebooks, which is simple and shows the result just below what we type, as an "all in one", or a somewhat more advanced console, such as Spyder, where we put the commands in a window and see the execution and the data (like an excel sheet) in others. It's more like Matlab or R and it's comfortable. There are also more complex and professional softwares like PyCharm.



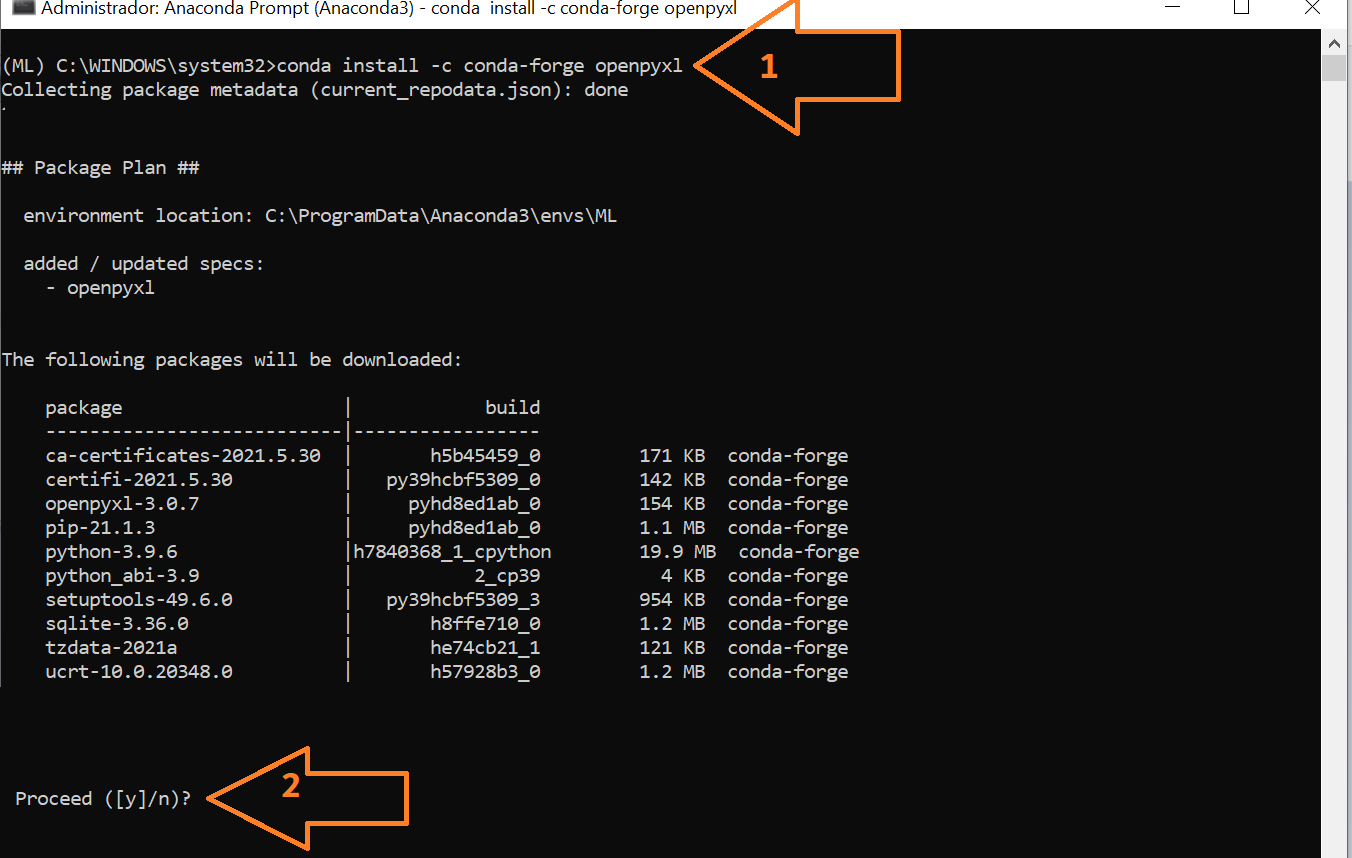
7. **Install Libraries**: Let's say that they are, following the comparison, our programs to make calculations, texts, graphics, etc. Let's suppose that working with a code we see that we need a library that allows our Spyder -from which we use Python- to read and save excel data. We search the internet and see that the OpenPyXl library allows us to do so. To install it we look in the conda-forge community in

anaconda.org/conda-forge/OpenPyXl

and we see that the installation is done with

conda install -c conda-forge openpyxl

Sometimes you have to use other ways to install, such as pip, instead of conda, or the library is in another internet channel or community that is not conda-forge, but it is not usually complicated. And in case of problems the internet is full of Python help forums and you can also ask us in private or public.



8. **Spyder**: Now you just have to start using it. We configure Spyder, with the colors we want (Tools / preferences) and with the appearance of the windows we want (View / Window Layout), for example, leaving the code in one window, the output in another (graphics, etc.) and in another the variables (the values ​​or data). Once configured from the menu we begin to type or copy code.

9. **Use of libraries**: As we have said before, some things are typical of Python, just as copying and pasting files is typical of Windows, but other things are done by libraries (just like programs in Windows).

The operation would be something like this. Let's imagine that in Windows we want to copy something and we use Ctrl + c, but let's suppose that we want to use the sum of excel and for that we had to use excel.suma. Or we want to change the color of the text, and for this we use word.textcolor. Windows doesn't work like this, but Python does.

If we want something typical of Python, like saying that name = ‘Ana’ we type it and it understands it. But if we want the square root function, that is what NumPy does and therefore we will tell it:

**import numpy** (import is only told once in the code, then it is 'open')

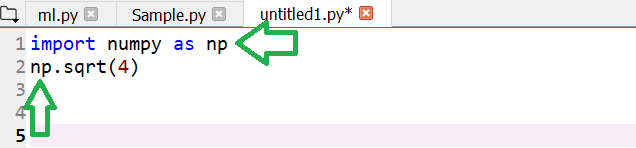
**numpy.sqrt (4)**

and it will return 2 to us.

Note: You can also use aliases for libraries, to shorten the names (Numpy is not very long, but matplotlib a little longer ...).

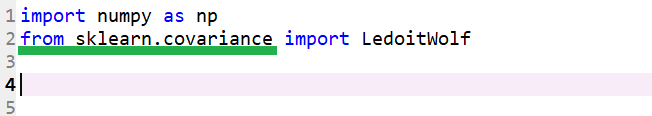
**import numpy as np**

**np.sqrt (4)**



By the way, with that output we can do whatever we want, save it, put it in an excel, send it by email, graph it and upload it to a server ...

10. On the other hand, we can also **import** (open) **only a part of the library**, so as not to load unnecessary resources. For example, if I am going to shrink a covariance matrix using the Ledoit and Wolf method, I don't have to import the entire machine learning library scikit learn (sklearn) but I would use “from sklearn.covariance import LedoitWolf”.



11. **Everything is ready**. It only remains to run the codes. If you know how to program Python, you will program it, if you don't, someone else will do it, but what you do need to know is to run it. If you know how to build a car and it has a problem, you can solve it, if not, you take it to the workshop, and if it doesn't even start, you call the tow truck. To drive it you don't need to know how the engine works.