

Python para finanzas

1. ¿Porqué Python para finanzas?

Jorge de la Vega Góngora

Gerencia de Análisis de Riesgos del Sistema Financiero
DGEF

Sesión 1

python: Introducción, instalación, configuración

¿Qué es python?

- `python` es un lenguaje de programación de alto nivel creado a principios de los 90's por el holandés Guido van Rossum. El nombre del programa viene de la serie de TV de la BBC *Monty Python's Flying Circus*.

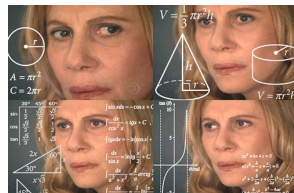


- Cuenta con las siguientes características:
 - Es un intérprete, con algunas partes compiladas: CPython compila el código fuente de `python` a bytecode y en ese momento interpreta ese bytecode, ejecutándose sobre la marcha.
 - Soporta diferentes paradigmas de programación: orientada a objetos y funcional.
 - `python` se puede usar en combinación con otros lenguajes e interactuar para ampliar las capacidades computacionales.
- Hubo dos versiones de `python` que se estuvieron desarrollando durante un tiempo la versión 2.6 se lanzó en paralelo con la versión 3.0 y eran incompatibles. Sin embargo, las versiones 2 fueron “descontinuadas” a partir del 20 de abril de 2020.
- La versión actual de `python` es la versión 3.8.8 (Abril 2021). La familia 3 se desarrolló para corregir errores de diseño fundamentales en el lenguaje y que no podían tener compatibilidad inversa con la familia 2.

- Hay varias maneras de instalar `python`. De hecho no hay un sólo `python`, hay muchas variedades y sabores de `python`'s: CPython, Jython, IronPython, PyPy (son máquinas virtuales).

Implementación	Máquina virtual	Lenguaje compatible
CPython	CPython VM	C
Jython	JVM	Java
IronPython	CLR	C#
Brython	Motor Javascript	JavaScript
RubyPython	Ruby VM	Ruby

Fuente: [¿porqué hay tantos pythons?](#)



- Al igual que R, `python` tiene un intérprete muy básico y su funcionalidad se aumenta a través de *bibliotecas* que necesitan instalarse de manera adicional y separada.
- Aunque `python` es multiplataforma, algunos paquetes no estándar tienen dependencias específicas del sistema operativo (aunque lo mismo pasa en R).

- Puede ser muy enredoso y complicado mantener todos los paquetes y versiones y puede consumir mucho tiempo (eg: recompilar dependencias). En un ratito se puede hacer un batidillo el sistema de paquetes y sus actualizaciones. Es mejor dejar que algo o alguien lo haga por nosotros. Algunas de las herramientas que nos pueden ayudar son las siguientes:
 - **Gestores de paquetes:** `pip` y `conda` ayudan con la instalación, actualización y eliminación de los paquetes, y llevan la gestión de las versiones.
 - **Gestores de ambientes:** `virtualenv` y `conda` permiten manejar diferentes instalaciones en paralelo sin que los paquetes de una versión se crucen entre si y haya conflictos.

Las opciones de instalación consideradas son las siguientes:

- Solo y directo: no se recomienda, es un lío administrar los paquetes en cualquier sistema. Aunque en las versiones usuales de Linux ya viene instalado.

Nivel de expertise: **alto**.

- Versión **Anaconda**: fácil y sin problema, la versión preferida si:
 - Nuevo a conda o python.
 - Gusta la conveniencia de tener python y sobre 1,500 paquetes científicos automáticamente instalados de una sola vez.
 - Se tiene espacio de disco: 3G
 - No se quiere instalar individualmente cada paquete que se quiere usar.

Nivel de expertise: **bajo**.

- Versión **Miniconda**: versión minimalista de python con conda. Disponible para Windows 😊, Mac, Linux. Opción preferida si:
 - Se prefiere instalar cada paquete de manera individual
 - No se tiene el tiempo (10min) o el espacio para instalar cerca de 1,500 paquetes de golpe.
 - Se quiere un acceso rápido a python y a los comandos de conda y se desea arreglar los otros programas después.
 - No se requiere tener una versión muy actualizada.

Nivel de expertise: **bajo**.

Adicionalmente, hay varios IDEs para trabajar de manera interactiva con python: Ipython + editor de texto, Jupyter, Spyder, PyDev, Atom, PyCharm, etc.

Yo preferí instalar Anaconda, y prefiero Spyder y Jupyter como IDEs.

- En Windows 😞:

- ➊ Ir a <https://www.anaconda.com/products/individual>
- ➋ Obtener el archivo del sistema operativo deseado. Pide registrarse para poder obtener el archivo de instalación correspondiente. Se obtiene un archivo ejecutable y hay que seguir las instrucciones.
- ➌ Para verificar si quedó bien instalado, abrir la línea de comando de Anaconda (buscar con `cmd`) y teclear
`python`
- ➍ Para salir del shell, teclear `Ctrl-Z` o `exit()`. Si se usa la línea de comando del sistema, no funcionará el atajo.
- ➎ También en la línea de comando se puede teclear `Ipython` y se abrirá la versión interactiva.

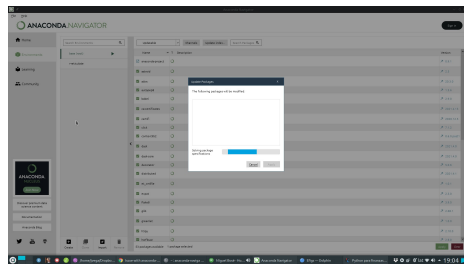
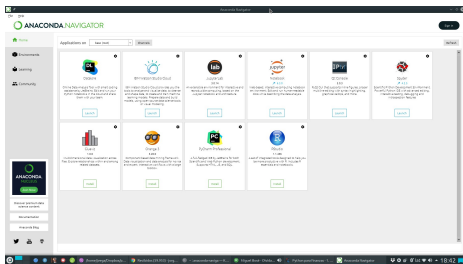
- En Mac:

- ➊ Obtener de <https://www.anaconda.com/products/individual> el OS X installer.
- ➋ Hacer doble click en el `.pkg` para ejecutar el installer
- ➌ Cuando el installer se ejecuta, pega la ruta del ejecutable al archivo `.bash_profile` que se encuentra en `/Users/$USER/.bash_profile`.
- ➍ Ejecutar `ipython`. Para salir del shell, presionar `Ctrl-D` o `exit()`.

- En Linux:

- ➊ Igual que en Mac, solo que en lugar de obtener `.pk` se obtiene el `.sh` y se ejecuta
`bash Anaconda3-2021.05-Linux-x86_64.sh`

- En todos los casos, una vez instalado Anaconda, se puede llamar a Anaconda-Navigator para actualizar, agregar o borrar paquetes instalados, crear nuevos ambientes e incluso instalar distribuciones de otros programas,



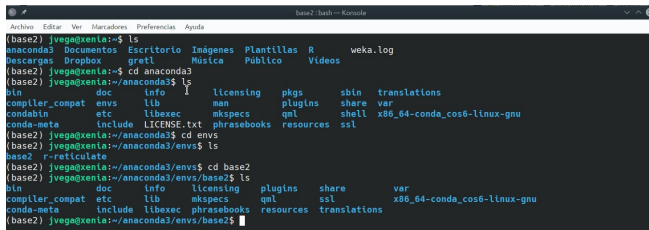
- 1 En Windows 😞: usar el [installer](#) correspondiente a la versión deseada.
- 2 Usar el correspondiente para [Mac](#)

Hilpisch muestra como instalar Miniconda en un contenedor de docker. En lo personal prefiero Anaconda.

- Un *ambiente virtual* es una copia de trabajo de `python` con un nombre, aislada y que mantiene sus propios archivos, directorios y rutas, para trabajar con versiones específicas de los paquetes sin afectar otros proyectos de `python`.
- Por ejemplo, consideren los siguientes ejemplos:
 - en mi laptop, cuando instalé Anaconda, se creó un ambiente llamado `base`.
 - Para hacer esta presentación, usé el paquete `reticulate` de R. El paquete creó un ambiente separado llamado `r-reticulate`
 - Adicionalmente, tuve que crear un ambiente `base2`, por problemas de instalación con el ambiente `base` que se creó en la instalación original.
- Para crear un ambiente virtual de `python` con nombre `tuambiente`:

```
conda create -n tuambiente python=x.x anaconda
```

El comando anterior instalará la versión de `python` indicada y todos los paquetes anaconda asociados en el directorio en `tu_anaconda3_dir/env/tuambiente`. Por ejemplo en mi caso:



```
(base2) jvega@xenia:~$ ls
anaconda3  Documentos  Escritorio  Imágenes  Plantillas  R          weka.log
Descargas  Dropbox     gretl      Música    Público     Videos
(base2) jvega@xenia:~$ cd anaconda3
(base2) jvega@xenia:~/anaconda3$ ls
bin          doc          info         licensing   pkgs        sbin        translations
compiler_compat  envs        lib         man         plugins     share       var
conda-meta    etc         libexec     mkspecs     qml         shell       x86_64-conda_cos6-linux-gnu
(base2) jvega@xenia:~/anaconda3$ cd envs
(base2) jvega@xenia:~/anaconda3/envs$ ls
base2  r-reticulate
(base2) jvega@xenia:~/anaconda3/envs$ cd base2
(base2) jvega@xenia:~/anaconda3/envs/base2$ ls
bin          doc          info         licensing   plugins     share       var
compiler_compat  etc         lib         mkspecs     qml         ssl         x86_64-conda_cos6-linux-gnu
conda-meta    include     libexec     phrasebooks resources    translations
(base2) jvega@xenia:~/anaconda3/envs/base2$
```

- Lista de los ambientes que están disponibles: `conda info -e`
- Para activar o cambiarse a un ambiente virtual usar:

`conda activate tuambiente`

Al activar un ambiente, se modifica el `PATH` y las variables del shell hacia la estructura creada. Esto cambiará el prompt para mostrar el ambiente en el que se está en un momento determinado.

```
(base) jvega@xenia:~/anaconda3/envs/base2$ conda activate r-reticulate
(r-reticulate) jvega@xenia:~/anaconda3/envs/base2$ conda info -e
# conda environments:
#
base                        /home/jvega/anaconda3
base2                       /home/jvega/anaconda3/envs/base2
r-reticulate                *  /home/jvega/anaconda3/envs/r-reticulate

(r-reticulate) jvega@xenia:~/anaconda3/envs/base2$
```

- Para terminar una sesión en el ambiente actual: `conda deactivate`. Esto regresará al ambiente base, que también se puede desactivar con `conda config --set auto_activate_base false`

Operaciones básicas con conda I

- Instala python X.X: `conda install python=x.x`
- Instala un paquete: `conda install $PAQUETE`
- Actualiza un paquete: `conda update $PAQUETE`
- Elimina un paquete: `conda remove $PAQUETE`
- Buscar paquetes: `conda search $TERMINO_A_BUSCAR`

```
(base2) jvega@xenla:~$ conda install matplotlib pandas pytables
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /home/jvega/anaconda3/envs/base2

added / updated specs:
- matplotlib
- pandas
- pytables

The following packages will be downloaded:

package | build | size
-----|-----|-----
hdf5-1.10.6 | hb1b8bf9_0 | 3.7 MB
numexpr-2.7.3 | py39h22e1b3c_1 | 185 KB
pandas-1.2.4 | py39h2531618_0 | 8.9 MB
pytables-3.6.1 | py39h77479fe_1 | 1.3 MB
-----|-----|-----
Total: 14.0 MB

The following NEW packages will be INSTALLED:

blosc | pkgs/main/linux-64::blosc-1.21.0-h8c45485_0
bzip2 | pkgs/main/linux-64::bzip2-1.0.8-h7b6447c_0
hdf5 | pkgs/main/linux-64::hdf5-1.10.6-hb1b8bf9_0
lzo | pkgs/main/linux-64::lzo-2.10-h7b6447c_2
mock | pkgs/main/neararch::mock-4.0.3-pyhd3eb1b0_0
numexpr | pkgs/main/linux-64::numexpr-2.7.3-py39h22e1b3c_1
pandas | pkgs/main/linux-64::pandas-1.2.4-py39h2531618_0
pytables | pkgs/main/linux-64::pytables-3.6.1-py39h77479fe_1
pytz | pkgs/main/neararch::pytz-2021.1-pyhd3eb1b0_0

Proceed ([y]/n)?
```

- El stack científico es uno de los principales atractivos de python actualmente:
 - **Numpy**: permite manejar arreglos de datos multidimensionales tanto homogéneos como heterogéneos y métodos/funciones optimizadas para este tipo de arreglos. (Similar a dataframes y tibbles en R).
 - **Scipy**: colección de subpaquetes y funciones que se usan comunmente en ciencias y en finanzas.
 - **matplotlib**: funciones para gráficas y visualización en 2D y 3D (similar a ggplot2 en R).
 - **pandas**: Amplía las capacidades de gestión de Numpy para el análisis de series de tiempo y datos tabulares.
 - **scikit-learn**: Paquetes para ML: estimación, clasificación, conglomerados.
 - **PyTables**: Almacenamiento de datos con entradas/salidas optimizadas.
 - **statsmodels**: Paquete de métodos y modelos estadísticos. Incluye modelos lineales, modelos de series de tiempo, modelos no paramétricos, visualización de los resultados de modelos estadísticos.
- **Ipython**: Una versión mejorada del intérprete que permite tener una versión interactiva. Esta versión se usa en **spyder** que es un IDE bastante amigable, similar a RStudio

```
jvega@xenia:~$ conda activate base2
(base2) jvega@xenia:~$ ipython
Python 3.9.5 (default, May 18 2021, 19:34:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.22.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print "Hola Mundo"
File "<ipython-input-1-d9ec7d547a2c>", line 1
      print "Hola Mundo"
      ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("Hola Mundo")?

In [2]: print("Hola Mundo")
Hola Mundo

In [3]: 3+4
Out[3]: 7

In [4]:
```

Ejemplos de aplicación

Ejemplo 1

El siguiente ejemplo calcula el valor de un call europeo vía Montecarlo.

```
import math
import numpy as np

S0 = 100      # Valor inicial del stock
K = 105      # precio strike
T = 1.0      # plazo
r = 0.05     # tasa libre de riesgo
sigma = 0.2  # Volatilidad

I = 100000   # número de simulaciones
np.random.seed(1000) # semilla aleatoria
z = np.random.standard_normal(I) # variables aleatorias normales estándar
ST = S0 * np.exp((r - sigma**2/2) * T + sigma * math.sqrt(T)*z) # Browniano geométrico
hT = np.maximum(ST-K, 0) # función a evaluar
C0 = math.exp(-r*T)*np.mean(hT) # Valuación de la opción
print('Valor de la opción call europea: {:.5f}'.format(C0)) # imprime resultado

Valor de la opción call europea: 8.0191.
```

El código anterior se puede escribir a un archivo de texto y ejecutarse como un script.

```
jvega@xenia:~/Dropbox/python_finanzas/scripts$ python3 callEur.py
Valor de la opción call europea: 8.0191.
```

Ejemplo 2 I

Los datos del libro de Hilpisch pueden obtenerse de su [sitio de Github](#).

```
import numpy as np          # Importa Numpy y pandas
import pandas as pd
from pylab import plt, mpl

# lee datos ejemplo provistos por Hilpisch
data = pd.read_csv('./data/tr_eikon_eod_data.csv', index_col=0, parse_dates=True)
data = pd.DataFrame(data['.SPX'])
data.dropna(inplace=True)

data.info()                # características de los datos

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2138 entries, 2010-01-04 to 2018-06-29
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   .SPX    2138 non-null       float64
dtypes: float64(1)
memory usage: 33.4 KB

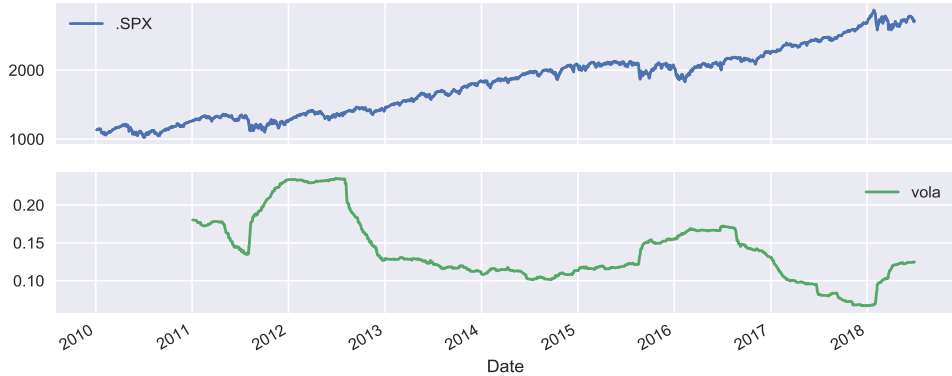
data['rets'] = np.log(data/data.shift(1))          # calcula los log-rendimientos
data['vola'] = data['rets'].rolling(252).std()*np.sqrt(252)  # calcula la volatilidad anualizada móvil

# Grafica las series de tiempo.
plt.style.use('seaborn')      # estilo de la gráfica
data[['.SPX', 'vola']].plot(subplots = True, figsize = (10,4))

array([<AxesSubplot:xlabel='Date'>, <AxesSubplot:xlabel='Date'>],
      dtype=object)

plt.show()                    # para que se muestren los resultados en la presentación.
```

Ejemplo 2 II



Ejemplo 3: Desempeño computacional de `python`. I

Ejemplo 3: Desempeño computacional de python. II

The screenshot shows a Jupyter Notebook titled 'PerformanceAI' running on a local host. The notebook contains 17 input cells. The first three cells set up the environment: importing math, defining a range of 3000000 loops, and defining a function f(x) that returns 3*log(x) + cos(x)**2. The fourth cell uses %timeit to measure the execution time of a standard Python loop for x in a, resulting in 1.06 s ± 4.47 ms per loop. The eighth cell imports numpy as np. The tenth cell defines a = np.arange(1, loops). The eleventh cell uses %timeit to measure the execution time of a standard NumPy loop, resulting in 55.6 ms ± 696 µs per loop. The twelfth cell imports numexpr as ne. The thirteenth cell sets ne.set_num_threads(1). The fourteenth cell defines f = '3*log(a) + cos(a)**2'. The fifteenth cell uses %timeit to measure the execution time of ne.evaluate(f), resulting in 28.4 ms ± 1.18 ms per loop. The sixteenth cell sets ne.set_num_threads(4). The seventeenth cell uses %timeit to measure the execution time of ne.evaluate(f), resulting in 8.9 ms ± 93.9 µs per loop. The output of the fourteenth cell is 8, and the output of the sixteenth cell is 1.

```
In [1]: import math
loops = 3000000

In [2]: a = range(1, loops)

In [3]: def f(x):
        return 3*math.log(x) + math.cos(x)**2

In [4]: %timeit r = [f(x) for x in a]
1.06 s ± 4.47 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [8]: import numpy as np

In [10]: a = np.arange(1, loops)

In [11]: %timeit r = 3*np.log(a) + np.cos(a)**2
55.6 ms ± 696 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [12]: import numexpr as ne

In [13]: ne.set_num_threads(1)
Out[13]: 8

In [14]: f = '3*log(a) + cos(a)**2'

In [15]: %timeit r = ne.evaluate(f)
28.4 ms ± 1.18 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [16]: ne.set_num_threads(4)
Out[16]: 1

In [17]: %timeit r = ne.evaluate(f)
8.9 ms ± 93.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Ejemplo 4: Aplicacion de ML y AI I

```
import numpy as np
import pandas as pd

data = pd.read_csv('../data/tr_eikon_eod_data.csv', index_col=0, parse_dates=True)
data = pd.DataFrame(data['AAPL.O']) # Selecciona Stocks de Apple
data['Rendimientos'] = np.log(data/data.shift()) # Calcula log-rendimientos de toda la serie
data.dropna(inplace = True)

lags = 6

cols = []
for lag in range(1, lags+1):
    col = 'lag_{}'.format(lag)
    # Genera columnas de un dataframe con rezagos direccionales
    data[col] = np.sign(data['Rendimientos'].shift(lag))
    cols.append(col)
data.dropna(inplace=True)
```

Ejemplo 4: Aplicacion de ML y AI II

```
from sklearn.svm import SVC

model = SVC(gamma = 'auto')
model.fit(data[cols], np.sign(data['Rendimientos']))

SVC(gamma='auto')

SVC(C = 1.0, cache_size = 200, class_weight=None, coef0 = 0.0,
    decision_function_shape = 'ovr', degree = 3, gamma = 'auto', kernel = 'rbf',
    max_iter = -1, probability = False, random_state = None, shrinking = True,
    tol = 0.001, verbose = False)

SVC(gamma='auto')

data['Prediction'] = model.predict(data[cols])
data['Estrategia'] = data['Prediction']*data['Rendimientos']
data[['Rendimientos', 'Estrategia']].cumsum().apply(np.exp).plot(figsize=(10,5))

<AxesSubplot:xlabel='Date'>

plt.show()
```

Ejemplo 4: Aplicacion de ML y AI III

