SERIES DE TIEMPO PARA PRONÓSTICOS EN ECONOMÍA Y FINANZAS

CLASE 0: Una muy breve introducción a Python

Elaborado por Jorge Guerra
Basado principalmente en el código de Jose S. Ñungo

> **¿Por qué incluir Python en un curso de Series de Tiempo?**

Python es un lenguaje de programación orientado a objetos, es decir, que el foco del mismo está en torno a los datos u objetos (diccionarios, listas, tuplas, dataframes, etc.) y no en las funciones y la lógica. Un objeto se puede definir como un campo de datos que tiene atributos y comportamiento únicos. En ese sentido, es fácil manipular dichos objetos y atributos de los mismos (ej: extraer el r2 o número de parámetros de una lista de 10000 posibles regresiones). Adicionalmente:

- Es el lenguaje de programación más popular del mundo en el 2021.
- Es el primero cuando se trata de temas de análisis de datos. Muchos economistas importantes usan Python.
- Es un lenguaje de propósito general, es decir que se utiliza para muchas cosas como Machine Learning, Internet of things, creación de aplicaciones web, de escritorio y de móviles. Por tanto es muy fácil integrarlo con diversas aplicaciones. Se puede usar para procesamiento de textos, recorte de videos, creación de bots para compras online, webscraping, simulaciones númericas, modelos dinámicos de equilibrio general, entre otros.
- Open-source (no necesita de licencia, como Matlab o Stata).
- Comunidad grande y colaborativa. Generalmente es muy fácil encontrar videos en Youtube, Blogs o foros de Stackoverflow con cualquier duda o problema que pueda tener el código.

A diferencia del Matlab, el lenguaje de Python no está cimentado sobre un unico objeto como lo son los vectores y matrices. A través de las librerias es posible darle el enfoque deseado. Por ejemplo, la libreria *Numpy* está orientada al procesamiento matricial en Python y uso en algunas ocasiones es necesario.

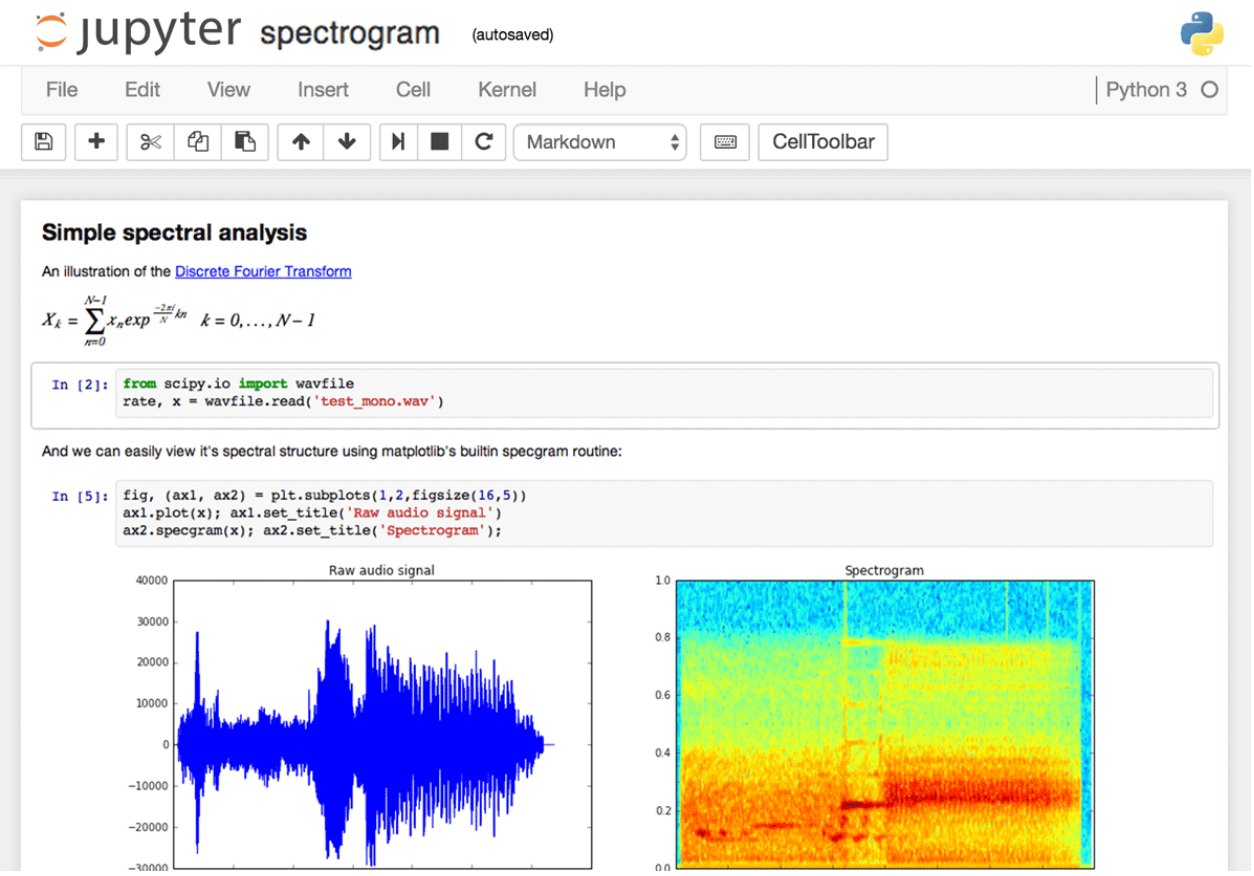
> **Recuerde**: Encuentra la instalación de Python, VSCode y Anaconda aquí.

> **¿Qué son los Jupyter Notebooks?**

Los archivos de extensión *.ipynb* presentan una interfaz muy cómoda para el aprendizaje, y la colaboración. Contiene:

- Celdas de código
- Celdas de markdown (como esta)
- Celdas de texto crudo
- Kernel

Lo que hacen los cuadernos de jupyter es transformar la información del código plano de python (*.py*) a una interfaz visual más amigable (como lo hace RStudio con R). Spyder es otra interfaz para Python muy similar a la de Matlab o RStudio.



La documentación oficial de Jupyter está aquí y puede encontrar algunos Notebooks didácticos aquí.

Estos pueden ser abiertos desde VSCode o Anaconda. Mientras el primero es un editor de Microsoft que soporta distintos lenguajes, la segunda una distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos, y aprendizaje automático. Puede encontrar más información de VSCode en este link y sobre la integración de los notebooks aquí. Para el caso de Anaconda puede explorar su sitio oficial.

Objetos en Python

> **Variables** : estas son creadas cuando se les asigna un valor y se les da un nombre.

In [ ]:
```
x = 5
y = 'Hola Mundo !!'
z = 5.4
```

Si queremos saber el valor de una variable utilizaremos la función nativa de Python *print()*

In [ ]:
```
print(x)
print(x,y)
print(f"el valor de la variable x es {x}")
```

```
5
5 Hola Mundo !!
```

```
el valor de la variable x es 5
```

Tipos de variables básicas

- Integers `x = 3`
- Floats `x = 3.0`
- Strings `x = '3'`
- Boolean `x = True` o `x = False`

Tipo de variable	Dentro de python
Texto	`str`
Numéricas	`int` , `float` , `complex`
Secuenciales	`list` , `tuple` , `range`
Mapeos	`dict`
Conjuntos	`set` , `frozenset`
Booleano	`bool`
Binario	`byte` , `bytearray` , `memoryview`
Vacios	`None`

In [ ]:
```
## EJEMPLOS

str1 = "Julian"
int1 = 4; float1 = 4.5; compl1=5j
list1 = [1,2,3,4,"Jorge", 4]; tuple1 = (1,2,3,4,"Jorge"); range1 = range(1,30,3) #rango del 1
dict1 = {"Dirección 1": "Carrera 5 numero 40-35", "numero de manzanas": 45, "rango": range1}
set1 = set(list1) ## es igual a la lista 1 solo que sin repetidos, es el conjunto
bool1 = True; bool2 = False
binari1 = 1
None1 = None


##NOTA: agregar ";" es equivalente a un salto de línea.
```

Casting de variables

Hay casos en los que queremos especificar el tipo de nuestra variable. Esto se puede hacer con el casting de variables:

> - `int()` - Construye un entero utilizando el argumento, puede construir un entero de un numero real haciendo utilizando la funcion *piso*.
> - `float()` - Construye un decimal utilizando el argumento.
> - `str()` - Construye una cadena de caracteres utilizando el argumento.

In [ ]:
```
x = int(2.8)
y = float(2.8)
z = str(2.8)

print(f"{x}, {y}, {z}")
print(f"{type(x)}, {type(y)}, {type(z)}")
```

```
2, 2.8, 2.8
<class 'int'>, <class 'float'>, <class 'str'>
```

Hay otros casos donde no es posible convertir las variables. Ej (strings a enteros):

In [ ]:
```
nombre1 = "Carol"
nombre2 = int(nombre1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-ce10ff38bf06> in <module>
      1 nombre1 = "Carol"
----> 2 nombre2 = int(nombre1)

ValueError: invalid literal for int() with base 10: 'Carol'
```

Que es distinto a esto:

In [ ]:
```
nombre1 = "3864"
nombre2 = int(nombre1)
nombre2
```

Out[ ]:
```
3864
```

Operacion de variables numéricas

Operación	Resultado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Modulo
//	División entera
**	Potencia

In [ ]:
```
a, b, c = 5, 2, 87 #otra forma de declarar variables

print(a,b,c)
```

```
5 2 87
```

Imprima los siguientes calculos:

1. Divida "a" en "b"
2. Divida usando "//" interactue "a" en "b" ¿qué hace "//"?
3. Multiplique "a" y "b".
4. Eleve "c" a la 3
5. Calcule el residuo entre "a" y "c"

In [ ]:
```
print(6%4)
```

```
2
```

Operaciones Booleanos (comparar valores)

Operación	Resultado
==	Igual
!=	No igual
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

In [ ]:
```
a == 27
```

Out[ ]:
```
False
```

In [ ]:
```
print(b>=87)
```

```
False
```

Imprima los siguientes calculos:

1. Pregunte si "a" es mayor que "b"
2. Pregunte si "a" es mayor o igual que "a"
3. Pregunte si "a" es igual a cualquier número

Listas y tuplas

Las listas son usadas para guardar muchos elementos en una sola variable.

In [ ]:
```
lista1 = ["manzana", "banano", "fresa"]
print(lista1)
```

```
['manzana', 'banano', 'fresa']
```

Para consultar cada elemento de una lista

In [ ]:
```
print(lista1[0])
print(lista1[1])
print(lista1[2])
```

```
manzana
banano
fresa
```

Para saber cuál es el tamaño de una lista siempre utilizaremos la función nativa de Python, `len()`. Por ejemplo:

In [ ]:
```
s = [0, 1, 2]
print(len(s))
```

```
3
```

1. Imprima el tamaño de la lista1

In [ ]:
```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
```

```
list3 = [True, False, False]
list4 = ["bc", 34, True, 40, "Hombre"]
```

No necesariamente las listas deben contener los mismos objetos y sus elementos pueden ser modificados.

In [ ]:
```
print(list4)
list4[4] = 3
list4[4] = 'Hola'
print(list4)
```

```
['bc', 34, True, 40, 'Hombre']
['bc', 34, True, 40, 'Hola']
```

Ahora bien **la principal diferencia** entre las listas y las tuplas es que las tuplas se definen con paréntesis y que no se pueden cambiar sus valores.

In [ ]:
```
tuple1 = ("bc", 34, True, 40, "Hombre")
list1 = ["bc", 34, True, 40, "Hombre"]
```

1. Verifique el tipo de tuple1 y list1
2. Intente cambiar un elemento de tuple1

Algunos métodos de las listas

1. `.sort()`
2. `.append()`
3. `.remove()`
4. `.reverse()`
5. `.index()`

In [ ]:
```
list1.append(7)
print(list1)
```

```
['bc', 34, True, 40, 'Hombre', 7]
```

In [ ]:
```
list1.reverse()
print(list1)
```

```
[7, 'Hombre', 40, True, 34, 'bc']
```

In [ ]:
```
list1.index(40)
```

Out[ ]:
```
2
```

Para mayor información sobre las listas puede consultar aquí. Dado que esta es una muy breve introducción, a lo largo de cada clase se irán explicando los conceptos de código necesarios para llevar a cabo las aplicaciones empíricas.

Diccionarios

Los diccionarios son usados para hacer variables que mapean e indexan elementos no organizados. En un formato de llaves y valores.

```
diccionario = {
     key1: value1,
     key2: value2,
        .
```

```
        .
        .
    }
```

In [ ]:
```
midict = {
  "marca": "Ford",
  "modelo": "Mustang",
  "anio": 1964
}
```

Para saber cuál es el tamaño de un diccionario siempre utilizaremos la función nativa de Python, `len()`.
Por ejemplo:

```
s = {'llave1': 123, 'llave2': 345}
print(len(s))
```

In [ ]:
```
s = {'llave1': 123, 'llave2': 345}
print(len(s))
```

```
2
```

Algunos métodos de los diccionarios

1. `.get()`
2. `.items()`
3. `.keys()`
4. `.values()`
5. `.update()`

Para mayor información sobre los diccionarios puede ir al siguiente sitio

> **En la introducción faltaron objetos importantes como los dataframes o arreglos de numpy, pero serán introducidos según lo requiera el caso**

In [ ]:
```

```