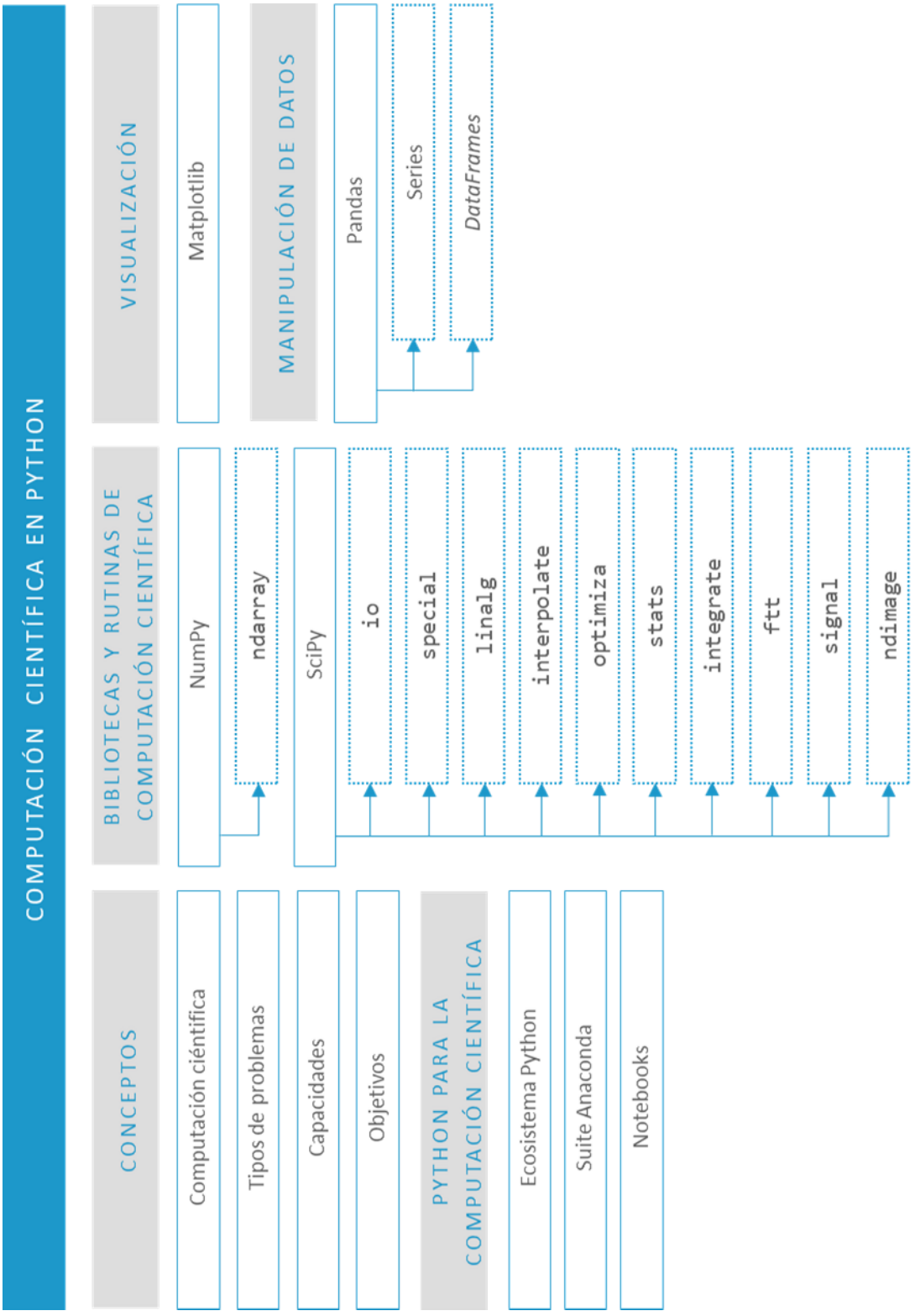
Programación Científica y HPC

Computación científica en Python

Índice

Esquema



Ideas clave

6.1. Introducción y objetivos

Antes de ver las características de Python para la computación científica es importante explicar qué es lo que se entiende por este tipo de computación.

La **computación científica,** que también se denomina informática científica o ciencia computacional, es, según la definición de Golub y Ortega (1992), la **colección de herramientas, técnicas y teorías que se requieren para poder resolver en un ordenador modelos matemáticos de problemas de ciencia e ingeniería.**

La mayoría de estos elementos se desarrollaron en matemáticas, incluso antes de la aparición de los ordenadores, y conforman lo que se conoce como **análisis numérico.** Sin embargo, la **aparición de los ordenadores y su capacidad de cómputo lleva a replantear muchos de los métodos numéricos desarrollados y aplicados manualmente,** además se tuvieron en cuenta algunos aspectos que se consideran accesorios y que, sin embargo, son relevantes para conseguir un uso eficiente y correcto de un sistema informático.

Surge, por tanto, una **nueva disciplina informática** en la que intervienen lenguajes de programación más o menos específicos, sistemas operativos y formas de manejar grandes conjuntos de datos. Sin embargo, **las matemáticas seguirán desempeñando un papel preponderante** dado que proporciona el lenguaje para especificar los modelos matemáticos y facilitan el análisis de las posibles soluciones si las hubiera.

La computación científica se basa en las matemáticas y la informática para buscar la mejor manera de usar los sistemas informáticos para resolver problemas de la ciencia y la ingeniería.

La computación científica está en rápido crecimiento y necesita usar capacidades informáticas avanzadas para resolver problemas complejos. Además, comprende un compendio de algoritmos (numéricos y no numéricos) para soportar y resolver los modelos y realizar simulaciones, *hardware* informático con el que se puedan satisfacer las exigencias de este tipo de computación.

El enfoque de la computación científica consiste en **adquirir conocimientos,** principalmente mediante el análisis de modelos matemáticos implementados en ordenadores. Dado que es una tarea multidisciplinar, los perfiles de las personas que trabajan en esto son ingenieros o profesionales expertos en el área del problema que se va a tratar, matemáticos e informáticos.

Los científicos e ingenieros **desarrollan programas informáticos,** es decir, un *software* de aplicación, que modela los sistemas que se estudian y ejecutan estos programas con varios conjuntos de parámetros de entrada. En algunos casos, **estos modelos requieren cantidades masivas de cálculos** (normalmente en coma flotante) y suelen ejecutarse en superordenadores o plataformas informáticas distribuidas.

«La esencia de la ciencia computacional es la aplicación de algoritmos numéricos y matemáticas computacionales» (Nonweiler, 1986).

La computación científica se considera hoy en día el «tercer pilar de la ciencia», junto al análisis teórico y los experimentos para el descubrimiento científico. El estudio de un sistema se aborda desde distintos puntos de vista, tal y como se ve en la Figura 1.

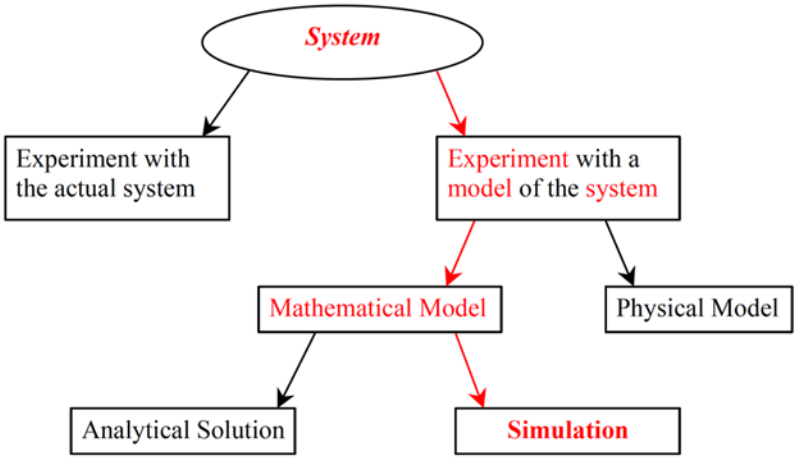


Figura 1. Formas de estudiar un sistema. Fuente: https://commons.wikimedia.org/w/index.php?curid=61926016

La computación científica pretende abordar situaciones como las que se enumeran a continuación:

Situaciones que aborda la computación científica

- Resolución de problemas que no pueden ser resueltos por medios experimentales o teóricos tradicionales, como predecir el cambio climático.
- Resolución de problemas en los que la experimentación puede resultar peligrosa. Por ejemplo, la caracterización de materiales tóxicos.
- Resolución de problemas en los que intentar resolverlos por otras vías sería muy costoso o consumiría mucho tiempo, como determinar la estructura de las proteínas.
- Realización de simulaciones por ordenador integradas con algoritmos de optimización para obtener diseños óptimos. Por ejemplo, el diseño óptimo de aviones.

Figura 2. Situaciones que aborda la computación científica. Fuente: elaboración propia.

A partir de lo expuesto y como conclusión, se pueden recopilar las **capacidades que debe tener un científico computacional, las cuales pueden ser de diversa índole,** como centradas en el modelado, en la computación o en la consultoría o asesoría.

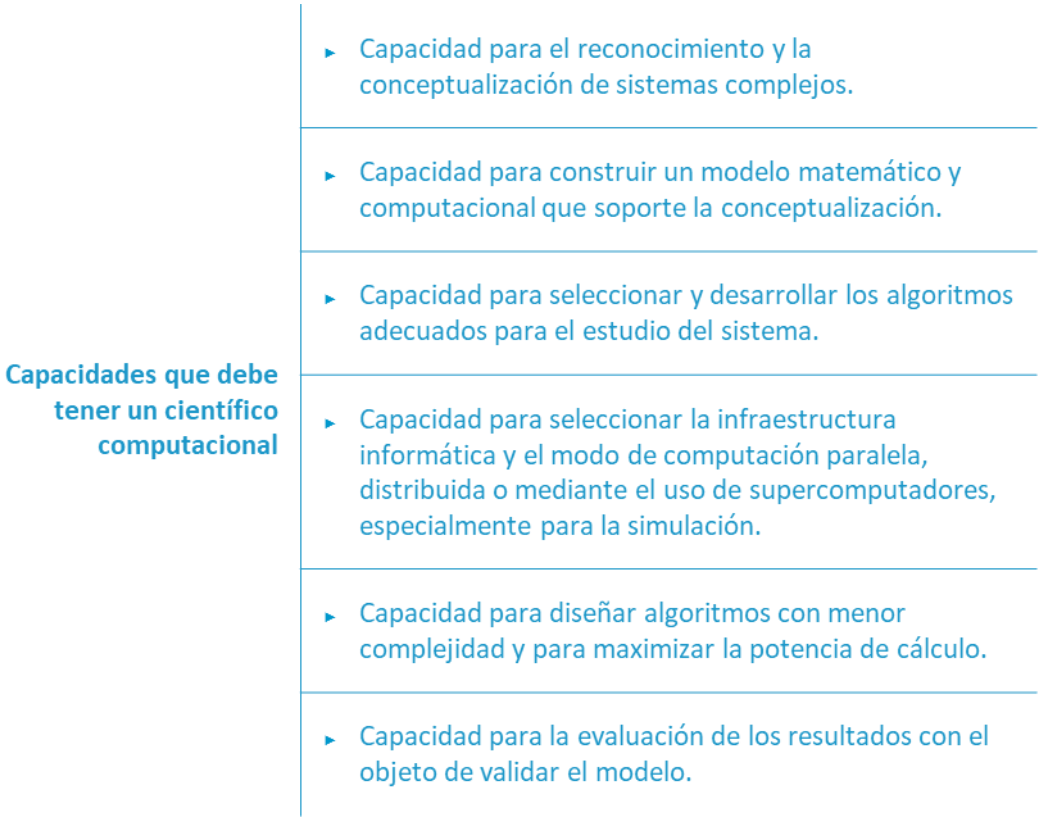


Figura 3. Capacidades que debe tener un científico computacional. Fuente: elaboración propia.

Objetivos

Tras todo lo expuesto, se puede determinar los objetivos y elementos principales de la computación científica. **El elemento base es el modelo matemático,** que debe ser especificado y, posteriormente, ser resuelto haciendo uso de los algoritmos o técnicas que se consideren más adecuadas.

En este tema se pretende ver **cómo resolver un modelo,** teniendo en cuenta que la resolución de los modelos no es exacta. Debido a esto, en la búsqueda de la solución se debe tener en cuenta la necesidad de:

- Encontrar un resultado lo más cercano posible al real con la búsqueda de algoritmos que converjan a él de la forma más rápida. Se debe intentar reducir la complejidad mediante el uso de las estructuras de representación de los datos más eficientes, implementar funciones más sencillas con una complejidad aceptable, entre otras.

- Realizar estimaciones de la exactitud y precisión de las soluciones obtenidas.

Por otro lado, se debe tener en cuenta la **simulación,** una técnica muy ligada a los modelos matemáticos para el estudio de sistemas reales, que **proporciona una forma de representación, permite estudiar su comportamiento y validar el diseño del modelo.**

La aplicación de la modelación matemática consiste en reemplazar el objeto de estudio real por su imagen o modelo matemático. Este se implementa mediante algoritmos lógico–numéricos en un ordenador, lo que permite el estudio de las características del proceso original.

Python como lenguaje para la computación científica

Una vez vistas, de manera resumida, las características y objetivos de la computación científica es importante resaltar la importancia de usar un lenguaje de programación para la implementación de los algoritmos necesarios, que pueda soportar los requisitos impuestos por ellos y mejorar la eficiencia de las implementaciones. El lenguaje debe contar con las estructuras de datos necesarias para almacenamiento eficiente de los mismos y los recursos que permitan implementar los algoritmos de forma eficiente.

Aunque existen múltiples lenguajes que podrían usarse, se ha elegido Python por las siguientes razones:

- Está orientado a objetos multiplataforma fácil de aprender y con una escritura explicita.
- Su software es libre y de acceso abierto.
- Cuenta con bibliotecas de cálculo científico bastante completas y otras que permiten complementarlas.

Preparando Python para la programación científica

Para crear un entorno de computación científica en Python es necesario ensamblar distintos módulos y herramientas. Se necesita contar con las estructuras que permitan la manipulación de los datos, obtener los resultados y visualizarlos para poder realizar análisis e informes.

Ya se sabe que **Python cuenta con estructuras de datos suficientemente potentes para soportar datos de distinta naturaleza,** además de contar con las estructuras de control que permiten la implementación de distintos tipos de algoritmos, y una serie de bibliotecas estándar que dan soporte a tareas habituales en programación. como entrada y salida, entre otras. También se cuenta con **entornos de desarrollo integrado, que nos permite escribir y ejecutar el código correspondiente.** Aunque esto no difiere mucho de los entornos de programación con los que cuentan otros lenguajes, pero Python no sorprende con más.

La cuestión entonces es cómo diseñar el mejor «ecosistema» que permita realizar programación científica de una manera más amigable y eficaz. En la Tabla 1 se muestran los elementos seleccionados.

Recursos	Descripción
Ipython notebook4	Consola avanzada que facilita el registro, la comprensión y la reproducción del análisis de datos.
NumPy	Paquete de computación científica de Python.
SciPy	Biblioteca de rutinas numéricas fáciles de usar y eficientes, como rutinas de integración numérica, interpolación, optimización, álgebra lineal y estadística.
Matplotlib	Biblioteca completa para a creación de visualizaciones estáticas, animadas e interactivas en Python.
Pandas	Biblioteca completa para a creación de visualizaciones estáticas, animadas e interactivas en Python.

Tabla 1. Recursos para un entorno de computación científica. Fuente: elaboración propia.

Gracias al soporte con el que cuenta Python se puede contar con este ecosistema completo en **Anaconda,** una distribución de código abierto y multiplataforma (Linux, macOS y Windows).

En esta *suite* se cuenta con aplicaciones**, bibliotecas y recursos diseñados para la computación científica y para la manipulación de datos,** como se muestra en la Figura 4.

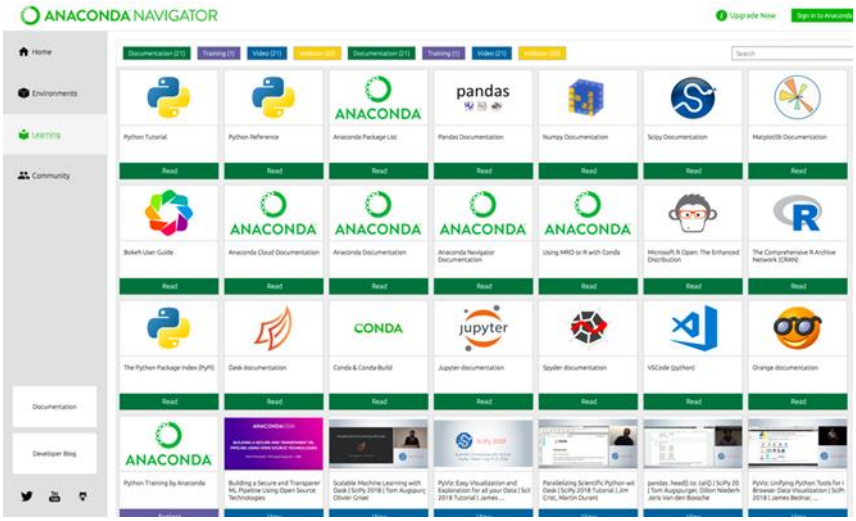


Figura 4. Galería de recursos disponibles en Anaconda. Fuente: elaboración propia.

Entre muchas características cabe destacar:

- Permite la instalación y administración de paquetes y entornos.
- Cuenta con los recursos necesarios para poder hacer computación científica y trabajar con datos.
- Elimina problemas de dependencia de paquetes y control de versiones.
- Cuenta con soporte para computación de alto rendimiento.
- Simplifica la implementación de proyectos que además son portables.

Pueden consultar la página oficial de Anaconda para su instalación.

6.2. IPyhton

Una de las características más útiles de Python es su **intérprete interactivo,** que **permite probar conjunto de instrucciones de forma rápida** sin la necesidad de crear archivos para almacenar el código y luego poderlo ejecutar. El intérprete suministrado con la distribución estándar de este lenguaje es algo limitado para un uso interactivo, eficaz y extendido, de ahí que se cuenten con otras propuestas mucho más completas como IPython.

El objetivo de IPython (Pérez y Granger, 2007) es crear un entorno completo para la computación interactiva y exploratoria. Para lograr este objetivo, cuenta con tres componentes principales:

- Una consola interactiva de Python mejorada.
- Un modelo de comunicación desacoplado de dos procesos, que permite que varios clientes se conecten a un núcleo de computación, mediante los *notebooks* basados en la web.
- Una arquitectura para la computación paralela interactiva.

Hay que tener en cuenta que todo IPython es de código abierto (publicado bajo la licencia BSD revisada).

La consola interactiva

Lo más característico de esta consola mejorada es su **propio sistema de ordenes** (comandos), que permite añadir funcionalidad cuando se trabaja de forma interactiva, y que **proporciona muchas ventajas para el analista de datos o el programador científico.**

Además, se puede usar como un intérprete incrustado al iniciarse con una sola llamada desde dentro de otro programa, proporcionando acceso al espacio de nombres actual.

En la Tabla 2 se muestran algunas ordenes sencillas.

Recursos	Descripción	Código ejemplo

Ayuda	Proporciona ayuda sobre ordenes, operaciones o paquetes disponibles.	`In [1]: sum?`
		`Signature: sum(iterable, /, start=0)`
		`Docstring:`
		`Return the sum of a 'start' value (default: 0) plus an iterable of numbers`
		
		`When the iterable is empty, return the start value.`
		`This function is intended specifically for use with numeric values and may`
		`reject non-numeric types.`
		`Type: builtin_function_or_method`
		
		`In [2]:`

Último valor	Recuerda el último valor calculado, usando _.	`In [2]: 2**3`
		`Out[2]: 8`
		
		`In [3]: _+256`
		`Out[3]: 264`

Guardar, editar y ejecutar archivos	Permite guardar la líneas de código seleccionadas en un archivo y posteriormente ejecutarlo o editarlo.	`In [15]: import numpy as np`
		
		`In [16]: array=np.linspace(0,30,6)`
		
		`In [17]: print(array)`
		`[ 0. 6. 12. 18. 24. 30.]`
		
		`In [18]: print(sum(array))`
		`90.0`
		
		`In [19]: %save /Users/mluisadiez/Desktop/pruebaArrayNumpy.py 15-18`
		
		`File `/Users/mluisadiez/Desktop/pruebaArrayNumpy.py` exists. Overwrite (y/[N])? y`
		`The following commands were written to file `/Users/mluisadiez/Desktop/pruebaArrayNumpy.py`:`
		`import numpy as np`
		`array=np.linspace(0,30,6)`
		`print(array)`
		`print(sum(array))`
		
		`In [20]: %run /Users/mluisadiez/Desktop/pruebaArrayNumpy.py`
		`[ 0. 6. 12. 18. 24. 30.]`
		`90.0`

Tabla 2. Algunas funciones sencillas. Fuente: elaboración propia.

Pero el potencial de esta consola es muchísimo mayor, lo que permite, entre otras, la **introspección dinámica de objetos,** esto posibilita acceder al código fuente y prototipos de definición de funciones y documentos, la búsqueda en módulos y espacios de nombres, navegación por el sistema de archivos y un marco de persistencia ligero.

En la página oficial de IPython pueden encontrar toda la documentación necesaria para su comprensión.

Pero la verdadera evolución de esta consola tiene lugar con la aparición del IPython *notebook*, **que permite la programación en un navegador.** De esta forma, **se puede reproducir todo el proceso de computación,** desde el desarrollo que se puede documentar debidamente con texto, la ejecución del código insertado y la comunicación de los resultados.

Los *notebooks* de IPython combinan dos componentes:

- **Una aplicación web** que se ha diseñado como una **herramienta basada en el navegador.** En esta se pueden crear, de forma interactiva, los documentos que combinan texto explicativo, instrucciones de código y resultados.
- El **documento que genera una representación de todo el contenido visible en la aplicación web,** incluidas las entradas y salidas de las ejecuciones, el texto explicativo, imágenes y representaciones multimedia de objetos.

Con esto, se pueden realizar análisis interactivos, compartir el código, reproducirlo publicarlo de forma sencilla. Además, los *notebooks* de iPython son compatibles con otros lenguajes como Julia, Haskell y Ruby, entre otros.

Para activar el servidor de *notebook*, se puede actuar de dos formas:

- A través de la línea de comandos con IPython *notebook*.
- Usando la interfaz de Anaconda, como se muestra en la Figura 5.

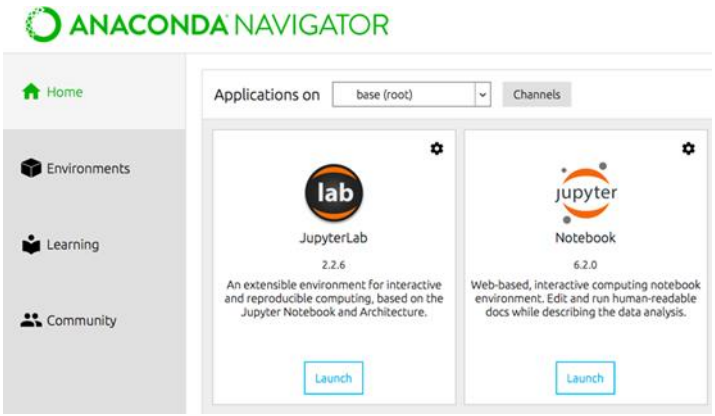


Figura 5. Home de Anaconda con el elemento Jupyter *notebook*. Fuente: elaboración propia.

Como ya se ha mencionado, esta es una aplicación web, por lo que se abre en el navegador. El elemento básico de un documento de *notebook* es la celda, por tanto, el mismo está formado por una secuencia de celdas.

El comportamiento de ejecución de una celda está determinado por el tipo de esta. Cada vez que se inserta una se puede elegir su tipo en el desplegable correspondiente a una celda que aparece en el menú.

Hay cuatro tipos de celdas:

- **Celdas de código:** para escribir código, con resaltado de sintaxis completo y completado de tabulación. Estas celdas pueden ser ejecutadas y su resultado se muestra en el propio *notebook*.

Una celda se ejecuta con Shift-Enter, haciendo clic en el botón Run de la barra de herramientas o en Cell|Run en la barra de menú.

- **Celdas de marcado:** para la inserción de texto descriptivo.
- **Celdas sin procesar:** proporcionan un lugar para escribir directamente una salida, por tanto, su formato no cambia.
- **Celdas de encabezamiento:** cada celda comienza siendo de código, pero su tipo puede cambiarse utilizando un desplegable en la barra de herramientas (que será "Código", inicialmente) o mediante atajos de teclado.

En la Figura 6, se muestra un *notebook* de ejemplo con los cuatro tipos de celda.

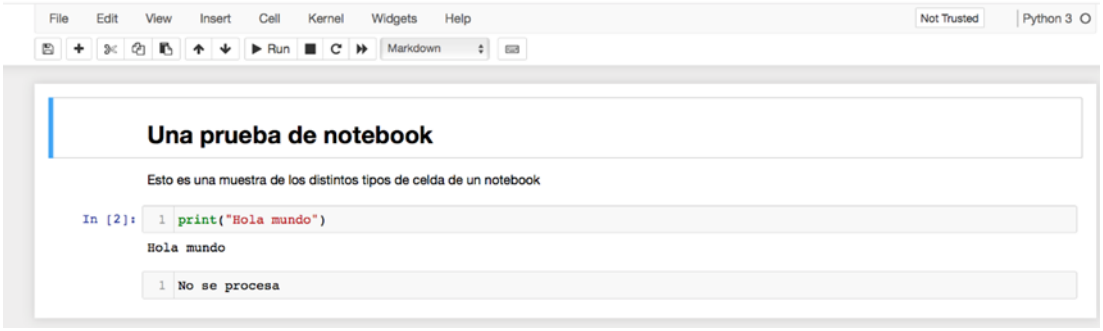


Figura 6. Los tipos de celda de un notebook. Fuente: elaboración propia.

En la página oficial de IPython pueden acceder a toda la documentación necesaria sobre los notebooks de este.

6.3. NumPy y SciPy

NumPy es un **paquete de computación científica** de Python y SciPy **proporciona biblioteca de rutinas numéricas fáciles de usar y eficientes,** como rutinas de integración numérica, interpolación, optimización, álgebra lineal y estadística.

A continuación, se describirá someramente cada una de ellas y se pondrá ejemplos de aplicación en Python.

NumPy

Es **una biblioteca básica para la computación científica y el análisis de datos.** Proporciona una estructura básica, los *arrays* multidimensionales `ndarray`. Esta estructura cubre las necesidades operacionales con colecciones de valores y permite hacerlo de forma eficiente.

Algunas características de NumPy son:

Características de NumPy
Está escrito en C (se integra en Python) y se utiliza para el cálculo matemático o numérico.
Es una biblioteca más eficiente que otras de Python con propósitos similares.
Se considera, actualmente, como la biblioteca más útil para lo que se conoce como ciencia de datos, si lo que se pretende es la realización de cálculos básicos.
No contiene nada más que el tipo de datos de *ndarray* (*arrays* multidimensionales, es decir, matrices) y las operaciones básicas sobre ellos.

Tabla 3. Características de NumPy. Fuente: elaboración propia.

Cabe recordar que, para el uso y manipulación de *arrays* en una programación tradicional, es necesario el recorrido de estos mediante el uso de bucles `for`. Esto, dependiendo del tamaño del *array* resultante, puede ser muy poco eficiente.

Para los `ndarray`, aunque tienen una estructura similar a las listas, se han definido operaciones y funciones que permiten manipulaciones eficientes de los datos. En la Tabla 4 se presentan las operaciones y funciones agrupadas por funcionalidad.

Operaciones y funciones	Descripción	Funciones y métodos disponibles

Creación	Permite crear *arrays* de distintas dimensiones con valores y especificar su tipo.	`array()`
		`arange()`
		`linspace()`
		`logspace()`

Indexación	Permite extraer el valor de un elemento de un array o de una dimensión completa en *arrrays* multidomensionales.	`operador []`

Consultas	Se puede obtener el tipo, las dimensiones y el tamaño de cada dimensión.	`dtype()`
		`ndim()`
		`shape()`

Redimensionado	Funciones que permiten cambiar las dimensiones de un array preservando los valores organizados de otra manera.	`reshape()` `flatten()`
Ordenación	Función y método que ordena un array seleccionando el criterio de ordenación por filas o columnas.	`función sort()` `método sort()`
Unión	Función que permite concatenar dos *arrays*.	`concatenate()`
Operaciones aritméticas entre *arrays* y con escalares	Los operadores aritméticos básicos están definidos para operar dos *arrays* entre si o con un escalar para el que se realiza la operación.	`+,-,*,/`

Funciones universales	Son funciones que se aplican a uno o dos *arrays* dependiendo de la definición y operan elemento por elemento.	`Matemáticas`
		`Trigonométricas`
		`Relacionales`
		`De conjuntos`
		`De algebra lineal`

Tabla 4. Operaciones y funciones sobre `ndarrays` de NumPy. Fuente: elaboración propia.

Una función destacable es la función `vectorize`, que permite aplicar funciones que están implementadas sobre escalares a todos los elementos de un *array*, mediante una vectorización de la operación. A continuación, se muestra un ejemplo:

> Aplicación de la función que comprueba si un número es par, que está definida para un número, a un array. De esta forma, la función devolverá un valor lógico o «booleano» para cada uno de los elementos del array. En la Figura 7 se muestra dicho código.

```
In [74]: def es_par(x):
    ...:     if x%2==0:
    ...:         return True
    ...:     else:
    ...:         return False
    ...:

In [75]: es_par(3)
Out[75]: False

In [76]: es_par(2)
Out[76]: True

In [77]: es_par_vec=np.vectorize(es_par)

In [78]: a=np.array([1,2,3,4,5,6,7,8,9,10])

In [79]: es_par_vec(a)
Out[79]:
array([False,  True, False,  True, False,  True, False,  True, False,
        True])
```

Figura 7. Vectorización de una función. Fuente: elaboración propia.

En la página oficial de NumPy pueden encontrar toda la documentación necesaria.

SciPy

Es una **biblioteca de código abierto en Python** que se usa para resolver problemas matemáticos, científicos y de ingeniería. La manipulación y visualización de los datos se puede realizar mediante el uso de múltiples operaciones y funciones disponibles.

Las características de SciPy son:

Características de SciPy
Contiene una gran variedad de recursos para la resolución de problemas comunes relacionados con la computación científica.
Actualmente, es la biblioteca más utilizada, solo superada por la GNU Scientific Library para C/C++ o Matlab.
Es fácil de usar y entender, además cuenta con una capacidad de cálculo eficiente.
Trabaja también con los *arrays* de NumPy, porque está construido sobre el mismo.
Cuenta con una versión completa del álgebra lineal, mientras que Numpy contiene sólo unas pocas características.
La mayoría de las características más avanzadas de la ciencia de datos están disponibles en Scipy, en lugar de Numpy.

Tabla 5. Características de SciPy. Fuente: elaboración propia.

SciPy cuenta con los paquetes y bibliotecas enumerados en la Tabla 6.

Módulo	Descripción
`scipy.io`	Módulos, clases y funciones disponibles para leer y escribir datos en una variedad de formatos de archivo.
`scipy.special`	Funciones universales para funciones de Bessel, función elíptica, función gamma y funciones para el manejo de errores
`scipy.linalg`	Funciones para operaciones de álgebra lineal
`scipy.interpolate`	Funciones para interpolación
`scipy.optimize`	Funciones para optimización de funciones objetivo, posiblemente sujetas a restricciones. Incluye solucionadores para problemas no lineales (con soporte para algoritmos de optimización local y global), programación lineal, mínimos cuadrados restringidos y no lineales, búsqueda de raíces y ajuste de curvas.
`scipy.stats`	Biblioteca de funciones estadísticas con un gran número de distribuciones de probabilidad definidas.
`scipy.integrate`	Biblioteca para integración numérica y ecuaciones diferenciales.
`scipy.ftt`	Funciones para transformadas discretas de Fourier.

Tabla 6. Paquetes y bibliotecas de SciPy. Fuente: elaboración propia.

Como se puede ver, SciPy es un **compendio de recursos que da un gran soporte para la computación científica de alto nivel.** Dado que las funciones que están implementadas se encuentran optimizadas y validadas, serán una opción para elegir por los científicos de datos, ya que les eximen de la necesidad de implementar muchas rutinas que puedan necesitar.

En la página de SciPy pueden encontrar toda la documentación necesaria para su comprensión.

6.4. Visualizaciones gráficas con Matplotlib

Es una **biblioteca para crear visualizaciones** estáticas, animadas e interactivas en Python. Esta sirve de apoyo para toda la computación científica, ya que permite mostrar sus resultados gráficamente y poder hacer análisis visuales.

Es importante conocer primero las funciones, los tipos de figuras y los patrones de uso de la biblioteca, para luego poder hacer uso de ella y aplicarlo a otras funciones científicas.

Se presenta en la siguiente Figura los elementos o la «anatomía» de una figura de Matplotlib.

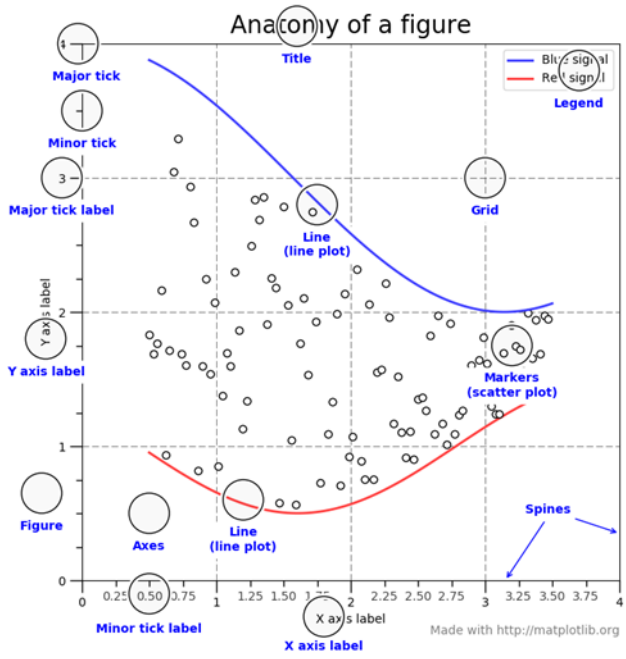


Figura 8. Anatomía de una figura. Fuente: https://matplotlib.org/stable/tutorials/introductory/usage.html

Como se puede ver en la Figura 8, y como indica en su guía, Matplotlib grafica sus datos en una figura que puede contener uno o más ejes (*axes*), es decir, áreas donde

los puntos pueden ser especificados en términos de coordenadas (x,y), theta-r en un gráfico polar o (x,y,z) en un gráfico 3D.

Es importante distinguir entre los *axes* o áreas de dibujo contenidas en una figura para mostrar las gráficas y los ejes X e Y (axis)

Básicamente, existen dos elementos de representación:

- **La figura o ventana** en la que se insertarán los gráficos y todos los elementos de configuración estética, como títulos, leyendas u otros. Es un objeto de la clase `matplotlib.figure.Figure`.

- **Los ejes,** que en realidad son las áreas de la figura que representan los gráficos propiamente dichos, representan una región de la figura con el espacio de datos. Son objetos de la clase `matplotlib.axes.Axes`.

Todas las funciones de trazado esperan `NumPy.array` como entrada. Las clases que son similares a los *arrays*, como los objetos de datos de pandas y `NumPy.matrix`, pueden o no funcionar como se pretende. Es mejor convertirlas a objetos `NumPy.array` antes de graficar.

De lo expuesto, se desprende que una forma de visualización se basa en usar el estilo orientado a objetos creando, explícitamente, figuras y ejes y llamar a métodos sobre ellos.

Pero existe otra forma relativamente sencilla, que es usar `pyplot` para crear y gestionar automáticamente las figuras y los ejes, y utilizar las funciones de este para el trazado.

Las funciones principales para crear figuras se muestran en la Tabla 7:

Función	Descripción
`figure(numeración,tamaño,resolución,color del fondo, color del perímetros,valor lógico para mostrar o no el marco)`	Crea una figura.
`subplot(numFilas,numColumnas,numGrafica)`	Crea varias gráficas en la misma ventana.
`plot(abcisas, ordenadas,colorytipo,anchoLinea,marcador)`	Varias gráficas en una única figura.

Tabla 7. Funciones para crear figuras. Fuente: elaboración propia.

En la página de Matplotlib pueden encontrar documentación pertinente para su comprensión.

Representación en *notebooks*

El uso de la función `plt.show()` da lugar a un *warning* en los *notebooks* e, incluso, puede que la figura no se muestre correctamente debido a que estas están pensadas para visualizarse en una interfaz gráfica de usuario (GUI, por sus siglas en inglés).

Para evitar esto existe una directiva `%matplotlib inline`, que es una función que renderiza la figura en un *notebook*, en lugar de mostrar un volcado del objeto figura.

6.5. Manipulación de datos con Pandas

Es la **biblioteca de software libre que proporciona Python para la manipulación de datos.** En la actualidad es la más usada en la disciplina conocida como ciencia de datos. Permite trabajar con múltiples formatos de archivos, entre ellos los archivos

CSV, que son archivos separados por comas y que se utilizan de forma habitual para el almacenamiento y organización de datos.

Con esta biblioteca se pueden manipular conjuntos grandes de datos numéricos, tablas y series de tiempo.

Pandas cuenta con estructuras de datos de distintas dimensiones: series de una dimensión, *DataFrames* de dos dimensiones, que en realidad son tablas, y paneles o cubos, que son estructuras de tres dimensiones.

Series

Están definidas como una clase en la Pandas y **representan estructuras similares a un *array* de una dimensión,** pero con algunas características:

- Todos los elementos de una serie son del mismo tipo.
- Su tamaño es fijo, no puede variar durante la ejecución.
- Su acceso es indexado por un tipo índice cuyos valores se definen cuando se crea la serie.

Estas permiten manipular listas de datos, hacer limpieza o modificar datos vacíos.

A continuación, se muestra la definición de series con índices y algunos usos de interés.

```
In [87]: #importacion de pandas
    ...: import pandas as pd
    ...:
    ...: calificaciones = pd.Series([10, 4.5, 6, 5.1])
    ...: calificaciones
Out[87]:
0    10.0
1     4.5
2     6.0
3     5.1
dtype: float64
```

Figura 9. Definición de una serie con índices por defecto. Fuente: elaboración propia.

```
In [88]: calificaciones=pd.Series([10,5.3,7.6,8.8,4.2,None],index=['pepe','juan','maria','rosa','luis','tomas'])

In [89]: calificaciones
Out[89]:
pepe     10.0
juan      5.3
maria     7.6
rosa      8.8
luis      4.2
tomas     NaN
dtype: float64

In [90]: calificaciones.values
Out[90]: array([10. ,  5.3,  7.6,  8.8,  4.2,  nan])

In [91]: calificaciones.index
Out[91]: Index(['pepe', 'juan', 'maria', 'rosa', 'luis', 'tomas'], dtype='object')
```

Figura 10. Definición de una serie con índices de nombres y un valor nulo. Fuente: elaboración propia.

La selección de datos en las series se puede realizar de la forma indexada habitual. Se muestra un código de selección de un rango de valores mediante el uso de condiciones.

```
In [93]: suspensos=calificaciones[(calificaciones>0) & (calificaciones<5)]

In [94]: suspensos
Out[94]:
luis    4.2
dtype: float64
```

Figura 11. Selección de datos con valores entre 0 y 5, sin incluir. Fuente: elaboración propia.

A continuación, se verán las **formas de aplicar una función a los elementos de una serie,** se define una función redondear que, si bien ya existe en Python (`round()`), se implementa para mostrar cómo se puede extraer la parte decimal y entera de un `float`. En esta función, se redondea el 0,5 por defecto.

```
In [95]: import math
    ...: def redondear(x):
    ...:     parte_decimal,parte_entera=math.modf(x)
    ...:     if parte_decimal>0.5:
    ...:         return x+1-parte_decimal
    ...:     else:
    ...:         return x-parte_decimal
    ...:
```

Figura 12. Función redondear. Fuente: elaboración propia.

```
In [98]: for i in range(len(calificaciones)):
    ...:     calificaciones[i]=redondear(calificaciones[i])
    ...:

In [99]: calificaciones
Out[99]:
pepe     10.0
juan      5.0
maria     8.0
rosa      9.0
luis      4.0
tomas     NaN
dtype: float64

In [100]: calificaciones=pd.Series([10,5.3,7.6,8.8,4.2,None],index=['pepe','juan','maria','rosa','luis','tomas'])

In [101]: calificaciones=calificaciones.apply(redondear)
```

Figura13. Aplicación de una función a los elementos de una serie con un `for` o con `apply()`. Fuente: elaboración propia.

Cuando el tamaño de la serie no es muy grande, no hay mucha diferencia temporal entre las dos formas. A continuación, se muestra **el tiempo que consume cada método sobre una serie muy grande.** Se ve claramente que el uso de `apply()`, es más eficiente.

```
In [113]: %%timeit
     ...: # mostrando performance
     ...:
     ...: serie = pd.Series(range(10000))
     ...: for i in range(len(serie)):
     ...:     serie[i] = redondear(serie[i])
     ...:
113 ms ± 715 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [114]: %%timeit
     ...: # mostrando performance
     ...:
     ...: lista = pd.Series(range(10000))
     ...: lista=lista.apply(redondear)
     ...:
     ...:
3.47 ms ± 37.3 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Figura 14. Tiempos medios de aplicación de una función a una serie con `for` y con `apply()`. Fuente: elaboración propia.

Se verán ahora algunas **funcionalidades interesantes en la manipulación de datos,** como puede ser la obtención de datos estadísticos o la modificación o limpieza de datos nulos que deben ser considerados en muchos modelos.

```
In [106]: calificaciones.describe(percentiles=[0.25, 0.5, 0.75])
Out[106]:
count     5.000000
mean      7.200000
std       2.588436
min       4.000000
25%       5.000000
50%       8.000000
75%       9.000000
max      10.000000
dtype: float64
```

Figura 15. Obtención valores estadísticos. Fuente: elaboración propia.

```
In [107]: calificaciones.notnull()
Out[107]:
pepe      True
juan      True
maria     True
rosa      True
luis      True
tomas    False
dtype: bool

In [108]: calificaciones.notnull().all()
Out[108]: False

In [109]: calificaciones.fillna(calificaciones.mean())
Out[109]:
pepe     10.0
juan      5.0
maria     8.0
rosa      9.0
luis      4.0
tomas     7.2
dtype: float64

In [110]: calificaciones['tomas']=None

In [111]: calificaciones
Out[111]:
pepe     10.0
juan      5.0
maria     8.0
rosa      9.0
luis      4.0
tomas     NaN
dtype: float64

In [112]: calificaciones.dropna()
Out[112]:
pepe     10.0
juan      5.0
maria     8.0
rosa      9.0
luis      4.0
dtype: float64
```

Figura 16. Consulta, modificación y limpieza de datos nulos. Fuente: elaboración propia.

Para profundizar sobre las series y sus funciones consulte la página de Pandas.

DataFrames

Uno de los **elementos más característicos** de Pandas son los marcos de datos o *DataFrames*, que son básicamente tablas, donde se almacenan los datos para su manipulación, procesamiento y análisis.

Pandas está construida sobre el paquete NumPy. Además, los datos de Pandas se usan en el análisis estadístico en SciPy, en las funciones de Matplotlib y en los algoritmos de aprendizaje automático en Scikit-learn.

El principal uso de los *DataFrames* de Pandas es la **exploración de conjuntos de datos y el uso de diversas funciones** que se enumeran a continuación:

- Limpieza de los datos y filtrado de filas o columnas de acuerdo con distintos criterios.
- Cálculo de estadísticas.
- Visualización de datos con la biblioteca Matplotlib.
- Almacenamiento de los datos transformados en distintas estructuras.

Un ejemplo sencillo y la aplicación de la función suma a los datos de las columnas (Axis=0) y a las filas (Axis=1), obteniéndose unas series, se muestra en la siguiente figura:

```
In [7]: import pandas as pd
   ...: import numpy as np
   ...:
   ...: tabla = pd.DataFrame({'A': [1, 2], 'B': [10, 20]})
   ...: print("LA TABLA")
   ...: print(tabla)
   ...: tablaSumaCol = tabla.apply(np.sum, axis=0)
   ...: print("LA SERIE SUMA DE COLUMNAS")
   ...: print(tablaSumaCol)
   ...:
   ...: tablaSumaFil = tabla.apply(np.sum, axis=1)
   ...: print("LA SERIE SUMA DE FILAS")
   ...: print(tablaSumaFil)
LA TABLA
   A   B
0  1  10
1  2  20
LA SERIE SUMA DE COLUMNAS
A     3
B    30
dtype: int64
LA SERIE SUMA DE FILAS
0    11
1    22
dtype: int64
```

Figura 17. Definición y uso de *DataFrame*. Fuente: elaboración propia.

Entrada y salida de datos

Los datos de entradas se pueden obtener de distintas fuentes:

- Archivos CVS, mediante read_csv().
- Archivos JSON, mediante read_json().

- Datos de una base de datos SQL, para lo que se necesita instalar sqlite2, que permite proporcionar las funciones necesarias para crear una conexión con la base de datos y para realizar las consultas a la base de datos.

Por supuesto, estos también serán formatos de salida mediante la conversión de los datos de una *DataFrame* en uno se esos tipos. La conversión se realizará mediante el uso de las funciones `to_csv()`, `to_json()` y `to_sql()`.

Las principales operaciones que se pueden realizar con un *DataFrame* se enumeran en la Tabla 8.

Funciones y atributos		Descripción
`head()`		Devuelve las primeras filas de un *DataFrame* (por defecto 5).
`tail()`		Devuelve las últimas filas de un *DataFrame* (por defecto 5).
`info()`		Proporciona información sobre el *DataFrame*.
`Atributo shape`		Indica el número de filas y columnas del *DataFrame*.
`append()`		Añade un *DataFrame* a otro, retorna una copia sin afectar el *DataFrame* destino.
`drop_duplicates()`		Retorna una copia del *DataFrame*. Se pueden definir varias opciones.

Operaciones y atributos sobre columnas	`columns`
	`rename()`
	`value_counts()`
		Devuelve los nombres de las columnas.
		Modifica el nombre de ciertas columnas.
		Decuelve la frecuencia de los valores de una columna.

Operaciones con valores nulos (`none` o `np.nan`)	`isnull()`	Devuelve un *DataFrame* con valores lógicos en las celdas dependiendo si están vacíos o no.
	`isnull().sum()`	Cuenta el número de los nulos en cada columna.
	`dropna()`	Elimina los valores nulos, eliminando las filas o las columnas que los contienen.

Tabla 8. Operaciones de un *DataFrame*. Fuente: elaboración propia.

Existen muchas otras características que permiten seleccionar columnas por su nombre o aplicando algún criterio y forma de aplicar funciones a las columnas de un *DataFrame*.

Como se puede ver, muchas de las cuestiones vistas para las series son aplicables a los *DataFrames* teniendo en cuenta que estos tienen una dimensión más.

Visualización gráfica de *DataFrames* y series

La integración de Pandas con MatplotLib, permite realizar la representación de los datos almacenados en los *DataFrames*.

En este vídeo se profundiza en distintas formas de **manipulación de datos con pandas y las formas de visualización de los datos con matplotlib**

Principalmente, los tipos de gráficos que se construirán son diagramas de barras, diagramas de cajas, histogramas, gráficos de dispersión y polígonos de frecuencias.

Se aconseja usar para datos categóricos los diagramas de barras y de cajas. Para variables continuas histogramas, gráficos de dispersión y polígonos de frecuencias.

6.6. Referencias bibliográficas

Golub, G. H. y Ortega, J. M. (1992). *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. Academic Press.

Matplotlib (s. f.). *Anatomy of a figure* [Imagen]. Matplotlib. Usage Guide — Matplotlib 3.4.3 documentation

Matplotlib (s. f.). *Frequencies of x and y* [Imagen]. Matplotlib. Python Plotting With Matplotlib (Guide) – Real Python

Matplotlib (s. f.). *Scatter: x versus y* [Imagen]. Matplotlib. Python Plotting With Matplotlib (Guide) – Real Python

Nonweiler, T. R. F. (1986). *Computational Mathematics: An Introduction to Numerical Approximation*. John Wiley & Sons Inc.

Perez, F., & Granger, B. E. (2007). IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering, 9*(3), pp. 21-29. https://doi.org/10.1109/MCSE.2007.53

Wikimedia Commons (13 de febrero de 2021). *Ways to study a system* [Imagen]. Wikimedia. File:Ways to study a system.png - Wikimedia Commons

6.7. Cuaderno de ejercicios

Ejercicio 1. *Arrays* con NumPy

Crear un *array* con NumPy, con valores entre 10 y 40. Mostrar el tipo de datos que almacena, su dimensión y el tamaño de las dimensiones. Crear un *array* nuevo con los datos inversos y convertir sus valores al tipo `float`.

```
import numpy as np

array=np.arange(10,40)
print(array)
print(array.dtype)
print(array.ndim)
print(array.shape)
inverse_array=array[::-1]
print(inverse_array)
array_inverso=np.asfarray(array_inverso)
print(array_inverso)
```

Código 18. Código ejercicio 1. Fuente: elaboración propia.

Consulte en la documentación del aula virtual el notebook y ejecútelo Ejercicio1_NumPy.ipynb

Ejercicio 2. Álgebra lineal con NumPy

Implementación del producto de dos matrices y cálculo de sus determinantes. Observen como el valor que da los determinantes es un `float`, pero se puede hacer la conversión a entero.

```
import numpy as np

A = [[1, 3], [2, 1]]
B = [[1, 1], [3, 1]]
AxB = np.dot(A, B)

from numpy import linalg
detA=int(np.linalg.det(A))
detB=np.linalg.det(B)



La matriz obtenida es [[10 4], [5,3]]
```

Figura 19. Código ejercicio 2. Fuente: elaboración propia.

Consulte en la documentación del aula virtual el notebook y ejecútelo

Ejercicio2_NumPyMatrices.ipynb

Ejercicio 3. Operaciones conjuntas con *arrays*

Extraer los elementos comunes de dos *arrays*, crear uno con los elementos del segundo, que no existen en el otro, obtener los elementos que tiene la misma posición en dos *arrays*.

```
import numpy as np

a = np.array([1,2,3,2,3,4,3,4,9,6])
b = np.array([7,1,10,2,7,4,9,4,9,8])
np.intersect1d(a,b)
np.setdiff1d(b,a)
np.where(a == b)

Se obtiene  c= [1 2 4 9], d= [ 7  8 10] y  d= (array([3, 5, 7, 8]),)
```

Figura 20. Código ejercicio 3. Fuente: elaboración propia.

Consulte en la documentación del aula virtual el notebook y ejecútelo

Ejercicio3_Operaciones_con_arrays.ipynb

Ejercicio 4. Álgebra lineal con `linalg` de Scipy

En este ejercicio se van a ver funciones sobre matrices de gran aplicabilidad y la resolución de un sistema de ecuaciones lineales.

- Dada una matriz, calcular su inversa.

```
from scipy import linalg
import numpy as np
#se define una matriz cuadrada
A=np.array([[1,2],[2,4]])
#se calcula su inversa
inv_A=linalg.inv(A)
print ("La inversa de A es :",inv_A)
#observese que la operacion anterior eleva una excepción
print(linalg.det(A))
#el determinante es 0
try:
    inv_A=linalg.inv(A)
except linalg.LinAlgError as e:
    print (str(e))
# se ha capturado la excepción
B=np.array([[1,1,2],[0,1,-4],[0,0,-1]])
print(linalg.inv(B))
#para B si se obtiene su inversa
```

Figura 21. Código del ejercicio 4.1. Fuente: elaboración propia.

- Calcular el determinante de una matriz cuadrada 3x3 y obtener su inversa si es distinto de 0.

```
B=np.array([[1,1,2],[0,1,-4],[0,0,-
1]])
det_B=linalg.det(B)
print(" El determinante de B es:")
print(det_B)
#en la siguiente instrucción se usa
el operador ternario de Python
inv_B=linalg.inv(B) if det_B!=0 else
[]
print("La inversa de B es ")
print(inv_B)
```

Figura 22. Código del ejercicio 4.2. Fuente: elaboración propia.

- Cálculo de los autovalores y autovectores de la matriz anterior.

```
autovalores_B,autovectores_B=linalg.eig(B)
print("Los autovalores de B son: ")
print(autovalores_B)
print("Los autovectores de B son: ")
print(autovalores_B)

#Se obtienen dos autovalores y dos autovectores
```

Figura 23. Código ejercicio 4.3. Fuente: elaboración propia.

- Dado un sistema de ecuaciones lineales, resolverlo de dos formas: usando su inversa y usando la función `solve()` de `linalg`. El sistema es:

$$3x + 2y + z = 1$$
$$5x + 3y + 4z = 2$$
$$x + y - z = 1$$

El resultado que se debe obtener es $x = -4, y = 6\ y\ z = 1$.

```
A = np.mat('[3 2 1;5 3 4;1, 1, -1]')
print(A)

b = np.mat('[1;2;1]]')
print(b)
# Usando la matriz inversa
inv_A=linalg.inv(A)

solucion=inv_A * b
print("La solución con la inversa es:")
print(solucion)

# Lo mismo, usando la funcion linalg.solve(A,b)
solucion=linalg.solve(A,b)
print("La solución con solve es:")
print (solucion)
```

Figura 24. Código ejercicio 4.4. Fuente: elaboración propia.

Consulte en la documentación del aula virtual el notebook y ejecútelo. Ejercicio 4_Algebra_lineal_en_Scipy.ipynb

Ejercicio 5. Representación de una recta, una parábola y una función cúbica

En este ejercicio se van a representar las funciones $y = x, y = x^2$ *e* $y = x^3$.

El objetivo de esta representación es mostrar la forma más simple de graficar las funciones. Se verá la visualización en un único espacio y la visualización en varios espacios (*axes*).

```
import matplotlib.pyplot as plt
import numpy as np
#rango de las x
x = np.linspace(-6,6)
#figura y gráficas (un axis)
figura,graficas = plt.subplots()
graficas.plot(x, x, label='recta')
graficas.plot(x, x**2, label='parabola')
graficas.plot(x, x**3, label='cubica')

#etiquetas y  título
graficas.set_xlabel('x')
graficas.set_title("Tres funciones")
```

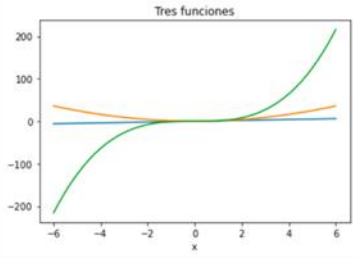


Figura 25. Código del ejercicio 5.1. Visualización en varios *axes*. Fuente: elaboración propia.

```
import matplotlib.pyplot as plt
import numpy as np

#rango de las x
x = np.linspace(-6,6)
#dibuja una figura con tres axes (1 fila y tres columnas)
fig,ax=plt.subplots(1,3)
#fija  el anco y alto de la figura
fig.set_size_inches(10,8)

#crea el gráfico de la primera columna (axe), en color rojo
ax[0].plot(x,x,label='recta', color="red" )
#crea el gráfico de la segunda columna (axe), en color por defecto
ax[1].plot(x,x**2,label='parabola')
#crea el gráfico de la tercera columna, en color verde
ax[2].plot(x,x**3,label='cubica', color="green")
#define las etiquetas de los ejes
for i in  range(3):
    ax[i].set_xlabel('x')
    ax[i].set_ylabel('y')
#define los nombre de cada gráfico
ax[0].set_title("recta")
ax[1].set_title("parabola")
ax[2].set_title("cubica")
```

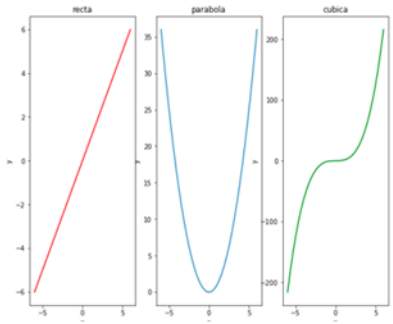


Figura 26. Código del ejercicio 5.2. Visualización en varios *axes*. Fuente: elaboración propia.

Consulte en la documentación del aula virtual el notebook y ejecútelo

Ejercicio5_Representacion_sencilla_de_funciones.ipynb

Ejercicio 6. Representación gráfica de tres funciones

En este ejercicio se van a representar sus funciones seno, coseno y producto de seno por exponencial, mostrando su amplitud.

El objetivo de esta representación no es otro que el mostrar el uso del graficador de Python y los elementos que permiten poner títulos y nombres, además de leyendas. No se proporciona ninguna interpretación de las funciones.

```
# se  definen las tres funciones
def f1(x):
    y = 8*np.sin(x)
    return y

def f2(x):
    y = 2*np.cos(x)
    return y

def f3(x):
    y = np.cos(x)*np.exp(-x/10)
    return y
# array de los valores de las abcisas con el número de valores
x = np.linspace(0, 10*np.pi, 100)
#se crean las gráficas de las tres funciones sobre los mismos valores de x
p1, p2, p3 = plt.plot(x, f1(x), x, f2(x), x, f3(x))
#Se crean las etiquetas de los ejes y el título de la figura
plt.xlabel('Tiempo en segundos')
plt.ylabel('Amplitud en cm')
plt.title('Representacion de tres funciones')
# Se añade la leyenda para cada función con tamaño de letra 10, en esquina inferior derecha
plt.legend(('Funcion seno', 'Funcion coseno', 'Funcion exponencial'),
prop = {'size': 10}, loc='lower right')

# Se crea la ventana (figura) con el tamaño (x,y) en pulgadas
figura=plt.figure(figsize=(200, 100))
#Se muestran las figuras en la ventana
plt.show()
```

Se obtiene

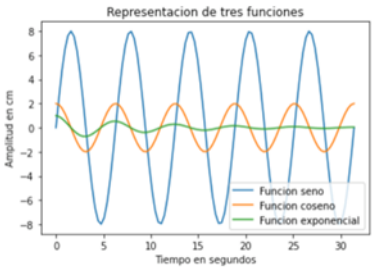


Figura 27. Código y resultado del ejercicio 6. Fuente: elaboración propia.

Consulte en la documentación del aula virtual el notebook y ejecútelo

Ejercicio6_Representacion_grafica_de_funciones.ipynb

Ejercicio 7. Representación de los valores y las frecuencias de variables discretas con distribución uniforme

En este ejercicio se van a generar los valores de dos variables discretas distribuidas uniformemente. En una de las gráficas se quiere representar la dispersión de los valores y en otra el histograma de frecuencias.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(low=1, high=11, size=50)
y = x + np.random.randint(1, 5, size=x.size)
data = np.column_stack((x, y))
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2,figsize=(8, 4))
ax1.scatter(x=x, y=y, marker='o', c='r', edgecolor='b')
ax1.set_title('Scatter: x versus y')
ax1.set_xlabel('$x$')
ax1.set_ylabel('$y$')
ax2.hist(data, bins=np.arange(data.min(), data.max()),label=('x', 'y'))
ax2.legend(loc=(0.65, 0.8))
ax2.set_title('Frequencies of $x$ and $y$')
ax2.yaxis.tick_right()
```

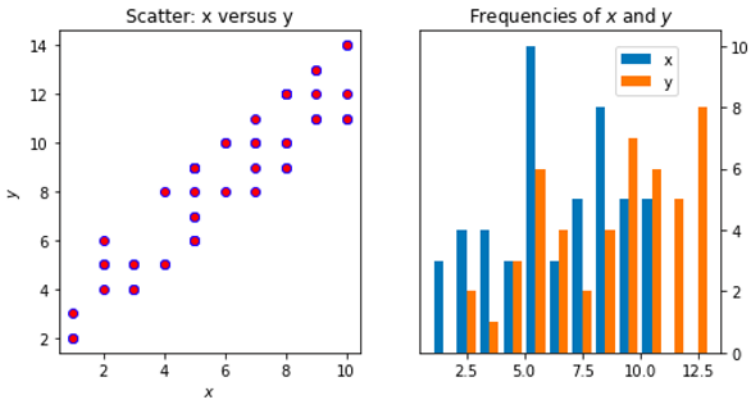


Figura 28. Código del Ejercicio 7. Fuente: https://realpython.com/python-matplotlib-guide/

Consulte en la documentación del aula virtual el notebook y ejecútelo

Ejercicio7_Representación_valores_y_frecuencias_variables_aleatorias.ipynb

Ejercicio 8. Lectura de datos de un archivo y escritura en CSV

Primero se verá la forma de leer datos de un archivo CSV indicando el delimitador de las columnas.

```
In [12]: tabla_estudiantes=pandas.read_csv('/Users/mluisadiez/Documents/Estudiantes.csv',';')

In [13]: tabla_estudiantes
Out[13]:
  Id_Estudiante Nombre_Estudiante  Curso
0           ID1   Pedro Rodríguez      1
1           ID2      Luis Sánchez      1
2           ID3      María Santos      2
3           ID4    Dolores Martos      2
4           ID5        Ana Olmedo      1
```

Figura 29. Lectura de archivo CSV delimitado por ;. Fuente: elaboración propia.

En el siguiente código se muestra como extraer solo algunas columnas o extraer las filas menos unas indicadas.

```
In [15]: tabla_nombre_curso=pandas.read_csv('/Users/mluisadiez/Documents/Estudiantes.csv',';',usecols=['Nombre_Estudiante','Curso'])

In [16]: tabla_nombre_curso
Out[16]:
  Nombre_Estudiante  Curso
0   Pedro Rodríguez      1
1      Luis Sánchez      1
2      María Santos      2
3    Dolores Martos      2
4        Ana Olmedo      1

In [17]: filas=pandas.read_csv('/Users/mluisadiez/Documents/Estudiantes.csv',';',skiprows=[2,3])

In [18]: filas
Out[18]:
  Id_Estudiante Nombre_Estudiante  Curso
0           ID1   Pedro Rodríguez      1
1           ID4    Dolores Martos      2
2           ID5        Ana Olmedo      1
```

Figura 30. Lectura de CSV con selectores. Fuente: elaboración propia.

Para crear un csv a partir de un DataFrame

```
In [23]: with open('/Users/mluisadiez/Documents/csv_data.csv','w') as csv_file:
   ...:     tabla_nombre_curso.to_csv(path_or_buf=csv_file)
   ...:
```

Figura 31. Escritura en archivo CSV. Fuente: elaboración propia.

A fondo

Preparar el entorno para la programación científica. Anaconda

Aprende Machine Learning (23 de marzo de 2018). *Instalar ambiente de Desarrollo Python Anaconda para Aprendizaje Automático*. Cómo instalar Python Anaconda | Aprende Machine Learning

En este artículo se muestra como instalar la suite Anaconda y las versiones más actualizadas de los recursos para preparar el entorno, no solo para programación científica, sino también para aprendizaje automático

Tutorial de NumPy

Interactive Chaos (s. f.). *Tutorial de NumPy. Presentación.* Presentación | Interactive Chaos

Tutorial completo para profundizar en el uso de NumPY.

Creación de paneles en Pandas

Data-Flair Training (s. f.*). 3 Ways to Create Pandas Panel – Learn to Transpose & Deprecate a Panel*. 3 Ways to Create Pandas Panel - Learn to Transpose & Deprecate a Panel - DataFlair (data-flair.training)

Recurso para profundizar sobre la creación y uso de paneles o cubos en Pandas.

Test

1. NumPy cuenta con un tipo de datos:

A. Listas multidimensionales.

B. *Arrays* multidimensionales.

C. No define ningún tipo de datos nuevos.

D. Ninguna de las anteriores es verdadera.

2. Los `ndarrays` de NumPy:

A. Se indexan mediante enteros.

B. Se indexan mediante cadenas.

C. No es una estructura indexable.

D. Ninguna de las anteriores es verdadera.

3. La función `arange` de NumPy:

A. Es una función que devuelve el rango de los índices de un `ndarray`.

B. Es una función que cambia el tamaño de un `ndarray`, pero no se pierden los valores.

C. Es una función para crear un `ndarray` con un tamaño especificado.

D. Ninguna de las anteriores es verdadera.

4. La función `vectorize` de NumPy:

A. Permite aplicar una función definida para cualquier tipo de datos a los elementos de un `ndarray`.

B. Permite aplicar una función definida para tipos escalares a los elementos de un `ndarray`.

C. Convierte una lista en un `ndarray`.

D. Ninguna de las anteriores es verdadera

5. Dados dos `ndarrray`:

A. La función `dot` de NumPy se usa para calcular potencias de matrices.

B. La función `dot` de NumPy calcula el determinante de una matriz.

C. La función `dot` de NumPy se usa para dividir matrices.

D. Ninguna de las anteriores es verdadera.

6. La biblioteca SciPy:

A. Es incompatible con los tipos de datos de NumPy.

B. Puede usar los `ndarrays` de NumPy.

C. Sus funciones no se pueden aplicar a ningún tipo de *arrays*.

D. Ninguna de las anteriores es verdadera.

7. La función `eig` definida en Scipy:

A. Es una función del módulo `linalg`.

B. Es una función del módulo `interpolate`.

C. Es una función del módulo `stats`.

D. Ninguna de las anteriores es verdadera.

8. Una figura de Matplotlib:

A. Puede contener varias gráficas, pero solo si están en *axes* o áreas independientes.

B. Puede contener varias gráficas, pero solo si están en el mismo *axe* o área.

C. Puede contener varias gráficas en *axes* o áreas independientes o en el mismo.

D. Ninguna de las anteriores es verdadera

9. Las series de Pandas:

A. Son estructuras indexadas, multidimensionales, de tamaño variable y de elementos del mismo tipo.

B. Son estructuras indexadas, multidimensionales, de tamaño fijo y de elementos del mismo tipo.

C. Son estructuras indexadas, unidimensionales, de tamaño fijo y de elementos de tipos distintos.

D. Son estructuras indexadas, unidimensionales, de tamaño fijo y de elementos del mismo tipo.

10. Los *DataFrames* de Pandas:

A. Son estructuras indexadas, de dos o más dimensiones y todos los elementos deben ser del mismo tipo.

B. Son estructuras indexadas, de dos o más dimensiones y los elementos pueden ser de distintos tipos.

C. Son estructuras indexadas de dos dimensiones y los elementos pueden ser de distintos tipos.

D. Son estructuras indexadas, de dos dimensiones y todos los elementos deben ser del mismo tipo.