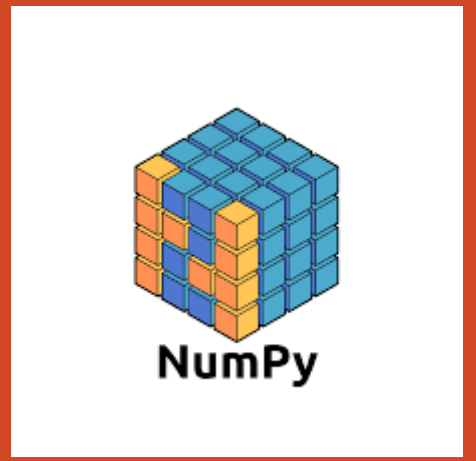
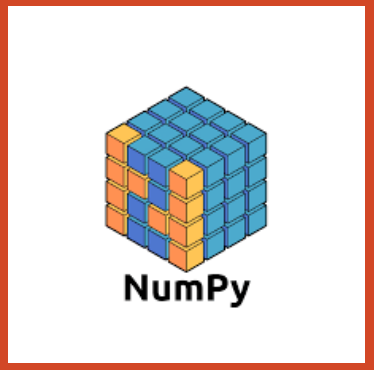
Módulos para para data science



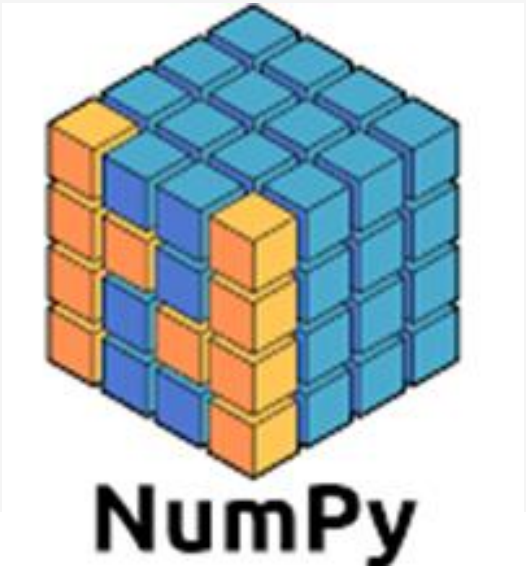
NumPy

Tutorial de Python Numpy Array



Intro

NumPy es un paquete de Python que significa “Numerical Python”, es la librería principal para la informática científica, proporciona potentes estructuras de datos, implementando matrices y matrices multidimensionales. Estas estructuras de datos garantizan cálculos eficientes con matrices.

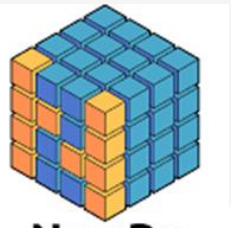


Intro

NumPy es, al igual que SciPy, Scikit-Learn, Pandas, etc.

- Es uno de los paquetes que no te puedes perder cuando estás aprendiendo ciencia de datos.
- Principalmente porque esta biblioteca te proporciona una estructura de datos de matriz que tiene algunos beneficios sobre Listas de Python, tales como:

1. Más compacto.
2. Acceso más rápido en la lectura y escritura de elementos.
3. Más conveniente y más eficiente.



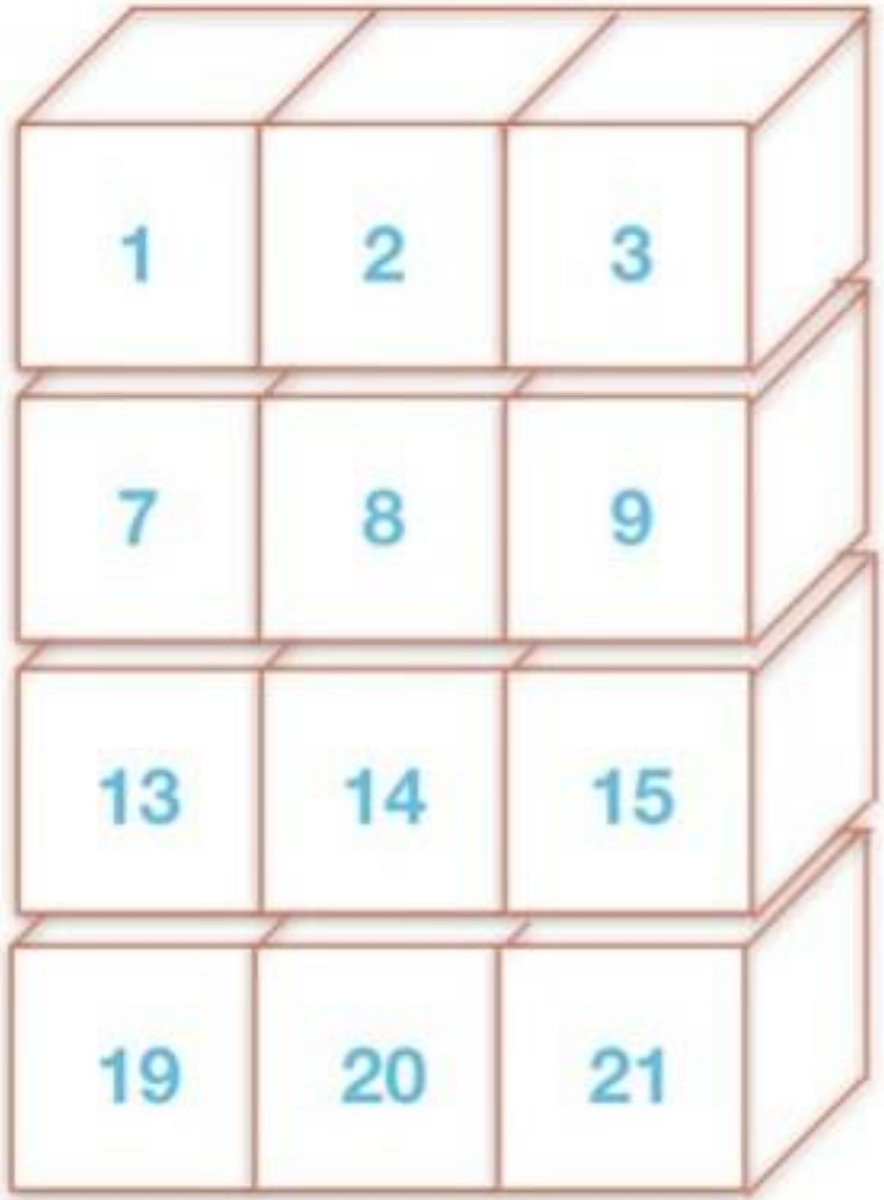
NumPy array o el arreglo de matrices de NumPy.

NumPy array es un potente objeto de matriz N-dimensional que tiene forma de filas y columnas, en la que tenemos varios elementos que están almacenados en sus respectivas ubicaciones de memoria.

En la siguiente imagen, tenemos una matriz bidimensional porque tiene filas y columnas, como puedes ver tiene cuatro filas y tres columnas, por lo que se convierte en una matriz bidimensional.

En el caso de que solo tuviera una hilera entonces habría sido una matriz unidimensional.

A =



A[0,0], A[0,1], A[0,2]

A[1,0], A[1,1], A[1,2]

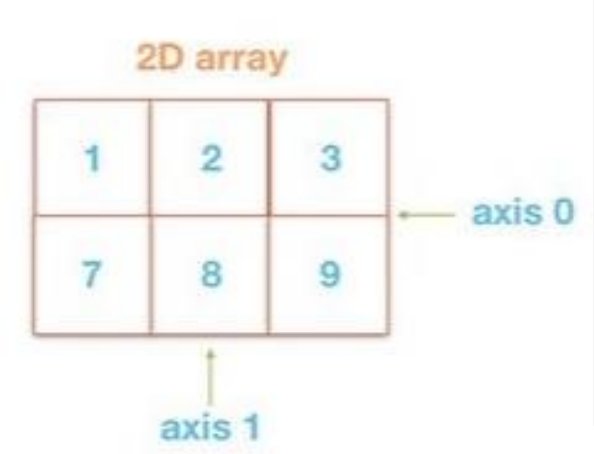
A[2,0], A[2,1], A[2,2]

A[3,0], A[3,1], A[3,2]

Matrices

Según lo explicado anteriormente en la primera figura tenemos una matriz unidimensional o 1D.

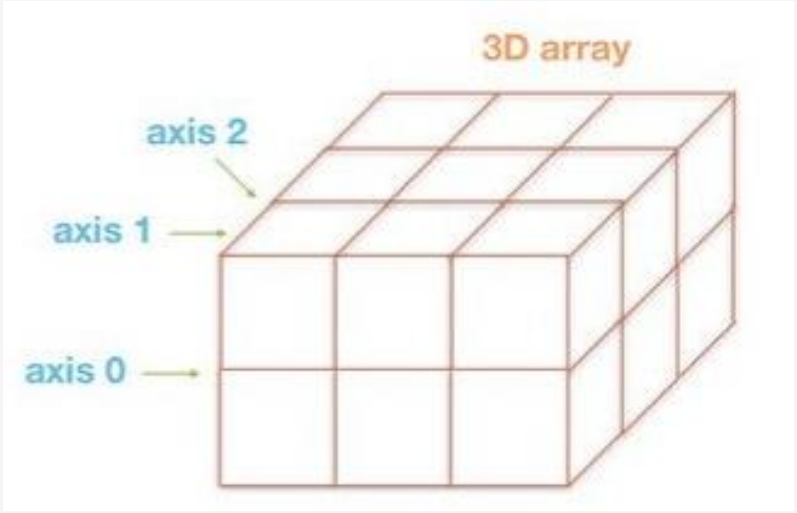
En la segunda figura, tenemos una matriz bidimensional o 2D, en donde las filas se indican como el eje 0, mientras que las columnas son el eje 1.



Matrices

Toma en cuenta que el número de ejes aumenta de acuerdo con el número de dimensiones, en matrices 3D, se tendrá un eje 2 adicional.

Ten en cuenta que estos ejes solo son válidos para matrices que tienen al menos dos dimensiones, ya que no tiene sentido esto para matrices unidimensional.



Numpy

Sabiendo todo esto, creemos ahora dos matrices una unidimensional y otra bidimensional.

```
import numpy as np


a=np.array([1,2,3])
print(a)


b=np.array([(1,2,3),(4,5,6)])
print(b)
```

Numpy

Otro Ejemplo, ahora con rapidez.

```
import time

SIZE = 1000000

L1= range(SIZE)
L2= range(SIZE)
A1= np.arange(SIZE)
A2=np.arange(SIZE)

start= time.time()
result=[(x,y) for x,y in zip(L1,L2)]
print((time.time()-start)*1000)

start=time.time()
result= A1+A2
print((time.time()-start)*1000)
```

Quien es mas rápido??

En este código se han definido dos listas en Python y dos matrices esta vez utilizando NumPy, posteriormente hemos comparado el tiempo tomado para encontrar la suma de los elementos.

Si observas los resultados, hay una diferencia significativa entre los dos, en el primero obtuvimos 207 ms mientras que la operación realizada con NumPy solamente tomo 51 ms en ser ejecutada.

Por lo tanto, las matriz creadas con NumPy son más rápida que la listas creadas directamente por Python.

Como trabaja?

Para hacer una matriz NumPy, puedes usar la función **np.array().**

Todo lo que necesitas hacer es colocarle una lista y, opcionalmente, también puedes especificar el tipo de datos a utilizar.

```
import numpy as np

array = np.array([[1,2,3,4], [5,6,7,8]], dtype=np.int64)
print(array)
```

Matrices Vacías

En ocasiones se requiere crear matrices vacías, esto se refiere a que se requieren marcadores de posición iniciales, que luego pueden ser rellenados. Se puede inicializar matrices con unos o ceros, pero también puedes hacer matrices que se llenan con valores espaciados uniformemente, valores constantes o aleatorios.

Matrices Vacías

Algunas de las instrucciones para crear este tipo de matrices son las siguientes:

```
# Crear una matriz de unos - 3 filas 4 columnas
unos = np.ones((3,4))
print(unos)
```

Para crear una matriz en donde todos los valores sean igual a 0, se utiliza la instrucción np.zeros y colocamos la cantidad de filas y columnas que contendrá la matriz.

```
# Crear una matriz de ceros - 3 filas 4 columnas
ceros = np.zeros((3,4))
print(ceros)
```

Otras opciones:

En caso de que se requiera crear una matriz con números aleatorios solamente debemos utilizar la instrucción np.random.random y definir el número de filas y columnas.

```
# Crear una matriz con valores aleatorios
aleatorios = np.random.random((2,2))
print(aleatorios)
```

Para crear una matriz vacía se utiliza la instrucción np.empty definiendo el número de filas y columnas.

```
# Crear una matriz vacía
vacia = np.empty((3,2))
print(vacia)
```

Otras opciones:

Para crear una matriz que contenga un solo valor en todas las posiciones utilizamos la instrucción np.full, definimos el número de filas y columnas, y a su vez definimos el valor que queremos en todas las posiciones, para nuestro ejemplo ese valor será igual a 8.

```
# Crear una matriz con un solo valor
full = np.full((2,2),8)
print(full)
```

Otras opciones:

Para crear matrices con valores espaciados uniformemente Podemos utilizer np.arange() y np.linspace().

La diferencia entre estas dos funciones es que para la primera se especifica que se desea crear una matriz que comience en 0 y por pasos de 5 genere los valores para la matriz que está creando.

```
# Crear una matriz con valores espaciados uniformemente
espacio1 = np.arange(0,30,5)
print(espacio1)
```

Otras opciones:

Por su parte para el segundo caso, se requiere una matriz con 5 valores que se encuentren entre 0 y 2.

```
espacio2 = np.linspace(0,2,5)
print(espacio2)
```

Otras opciones:

NumPy también permite crear matrices identidad utilizando las funciones np.eye() y np.identity().

Recuerda que una matriz de identidad es una matriz cuadrada de la cual todos los elementos en la diagonal principal son igual a 1 y todos los demás elementos son igual a 0.

```
# Crear una matriz identidad
identidad1 = np.eye(4,4)
print(identidad1)

identidad2 = np.identity(4)
print(identidad2)
```

Inspección de matrices

NumPy contiene varias instrucciones para obtener más información sobre las matrices que se han creado utilizando esta librería algunas de ellas son las siguientes:

Puedes encontrar la dimensión de la matriz, ya sea una matriz bidimensional o una matriz dimensional única, utilizando la función “ndim”.

```
# Conocer las dimensiones de una matriz
a = np.array([(1,2,3),(4,5,6)])
print(a.ndim)
```

Inspección de matrices

También puedes encontrar el tipo de datos de los elementos que están almacenados en una matriz, para ello utilizar la función “dtype”, el cual arrojará el tipo de datos junto con el tamaño.

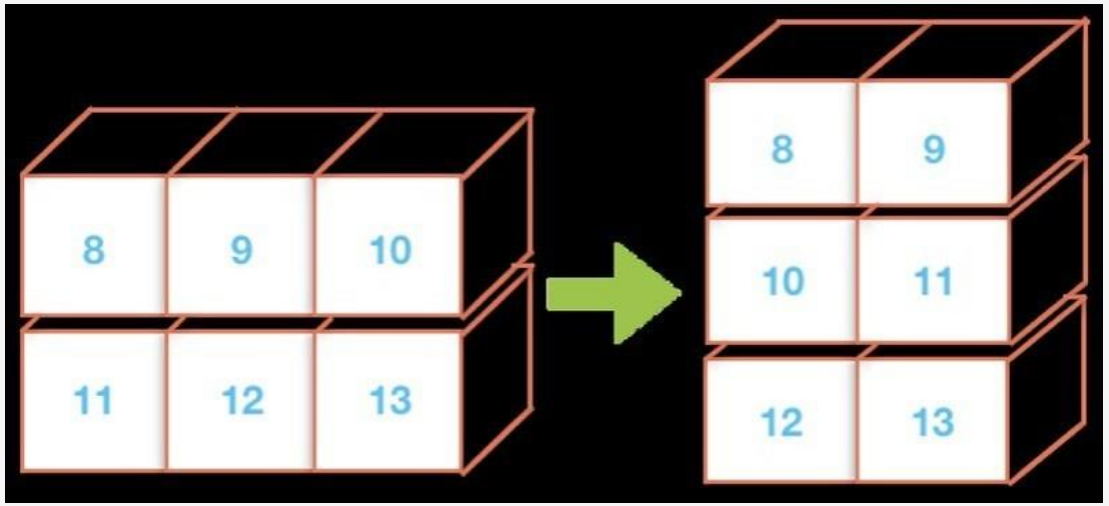
```
# Conocer el tipo de los datos
a = np.array([(1,2,3)])
print(a.dtype)
```

Se puedes encontrar el tamaño y la forma de la matriz con la función “size” y “shape”, respectivamente.

```
# Conocer el tamaño y forma de la matriz
a = np.array([(1,2,3,4,5,6)])
print(a.size)
print(a.shape)
```

Cambio de tamaño y forma de las matrices

Otras de las operaciones que podemos realizar con NumPy es el cambio de tamaño y forma de las matrices.



Cambio de tamaño y forma de las matrices

El cambio de la forma de las matrices es cuando se cambia el número de filas y columnas que le da una nueva vista a un objeto.

```
# Cambio de forma de una matriz
import numpy as np
a = np.array([(8,9,10),(11,12,13)])
print(a)
print("\n"*2)
a=a.reshape(3,2)
print(a)
```

Como puedes ver en la imagen, tenemos 3 columnas y 2 filas que se han convertido en 2 columnas y 3 filas utilizando la instrucción “reshape”.

Otras opciones:

podemos utilizar con NumPy es seleccionar un solo elemento de la matriz, para ello solamente tenemos que especificar el número de columna y fila en donde está ubicado y nos devolverá el valor almacenado en dicha ubicación.

```
import numpy as np
a=np.array([(1,2,3,4),
            (3,4,5,6)])
print (a)
print ("\n"*1)
print(a[1,2])
```

Otras opciones:

podemos utilizar con NumPy es seleccionar un solo elemento de la matriz, para ello solamente tenemos que especificar el número de columna y fila en donde está ubicado v nos devolverá el valor almacenado en dicha ubicación.

```
import numpy as np
a=np.array([(1,2,3,4),
            (3,4,5,6)])
print (a)
print ("\n"*1)
print(a[1,2])
```

Utilizando este mismo método podemos obtener todos los valores ubicados en la columna 2 de todas las filas. Para ello simplemente le decimos que desde la fila 0 en adelante, los dos puntos representa todas las filas, va a tomar el valor de la columna 2. Recuerda que la numeración de filas y columnas siempre comienza en 0.

```
import numpy as np
a=np.array([(1,2,3,4),
            (3,4,5,6)])
print(a[0:,1])
```

Operaciones matemáticas entre matrices

Con la librería de NumPy puedes realizar de manera muy fácil operaciones aritméticas, veamos algunas de ellas. Podemos encontrar el valor mínimo, máximo y la suma de la matriz con NumPy, para ello utilizamos “min”, “max” y “sum”, respectivamente.

```
import numpy as np
a= np.array([2,4,8])
print(a.min())
print(a.max())
print(a.sum())
```

También puedes obtener la raíz cuadrada y la desviación estándar de la matriz.

```
import numpy as np
a=np.array([(1,2,3),(3,4,5,)])
print("\n"*2)
print(np.sqrt(a))
print("\n"*2)
print(np.std(a))
```

Operaciones matemáticas entre matrices

Puedes realizar más operaciones con las matrices de NumPy, como son la suma, resta, multiplicación y división de las dos matrices.

```
import numpy as np
x= np.array([(1,2,3),
             (3,4,5)])
y= np.array([(1,2,3),
             (3,4,5)])
print(x+y)
print("\n")
print(x-y)
print("\n")
print(x*y)
print("\n")
print(x/y)
```

Usar documentación complementaria de Numpy:

https://numpy.org/doc/

Pandas

Tutorial de Python Pandas

Pandas

Pandas es un popular paquete de Python para la ciencia de datos y Machine Learning.

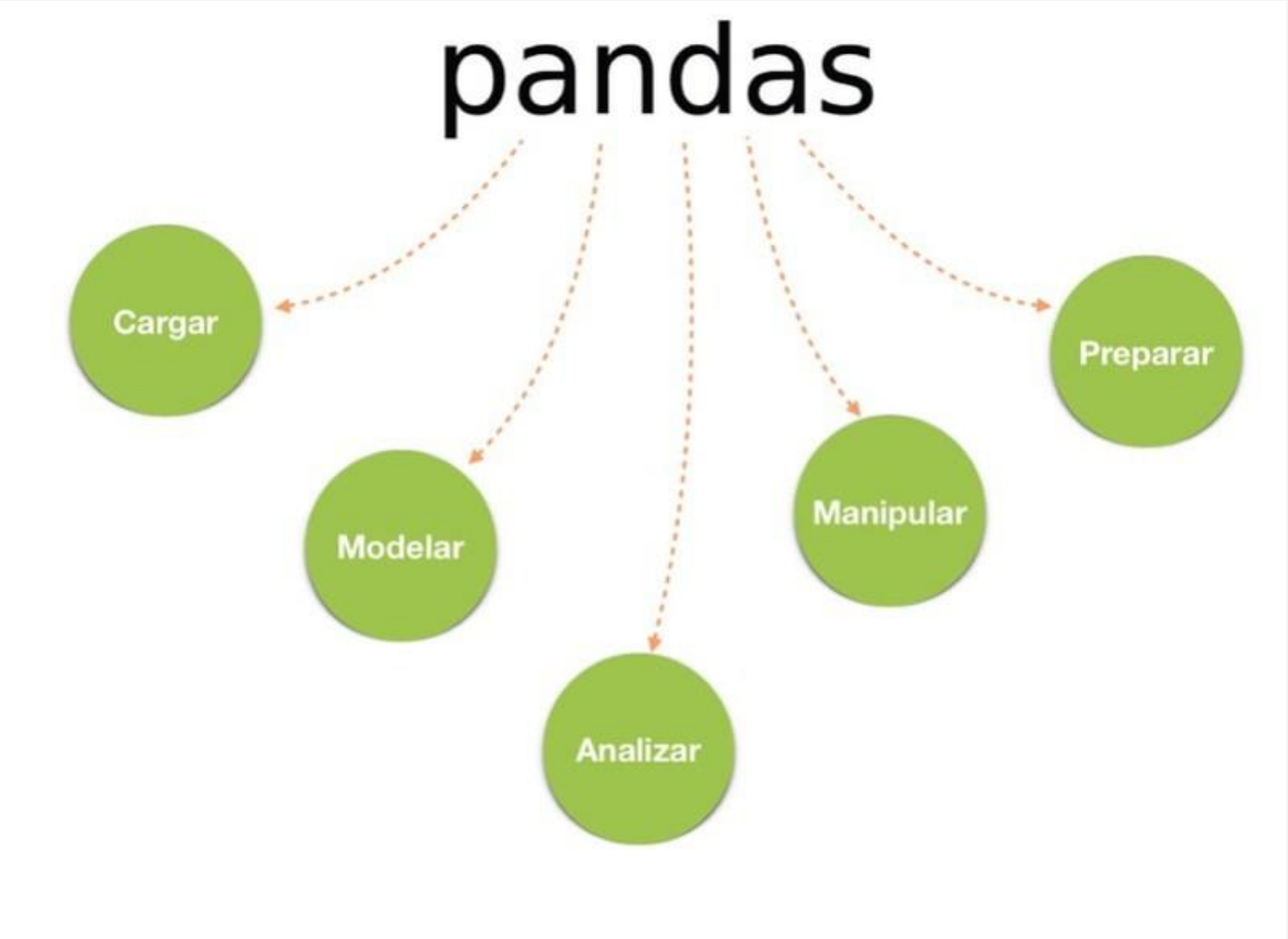
Las razones son muchas y es que ofrece estructuras de datos poderosas, expresivas y flexibles que facilitan la manipulación y análisis de datos.

Entre las estructuras más utilizadas se encuentra el DataFrame.

Pandas es una librería de código abierto de Python que proporciona herramientas de análisis y manipulación de datos de alto rendimiento utilizando sus potentes estructuras de datos.

El nombre de Pandas se deriva del término “Panel Data” y es la librería de análisis de datos de Python.

Pandas



Pandas

Esta librería se desarrollo debido a la necesidad de tener una herramienta flexible de alto rendimiento para el análisis de datos.

Anteriormente Python se utilizaba para la manipulación y preparación de datos por lo que no era utilizado para Machine Learning, Pandas resolvió este problema.

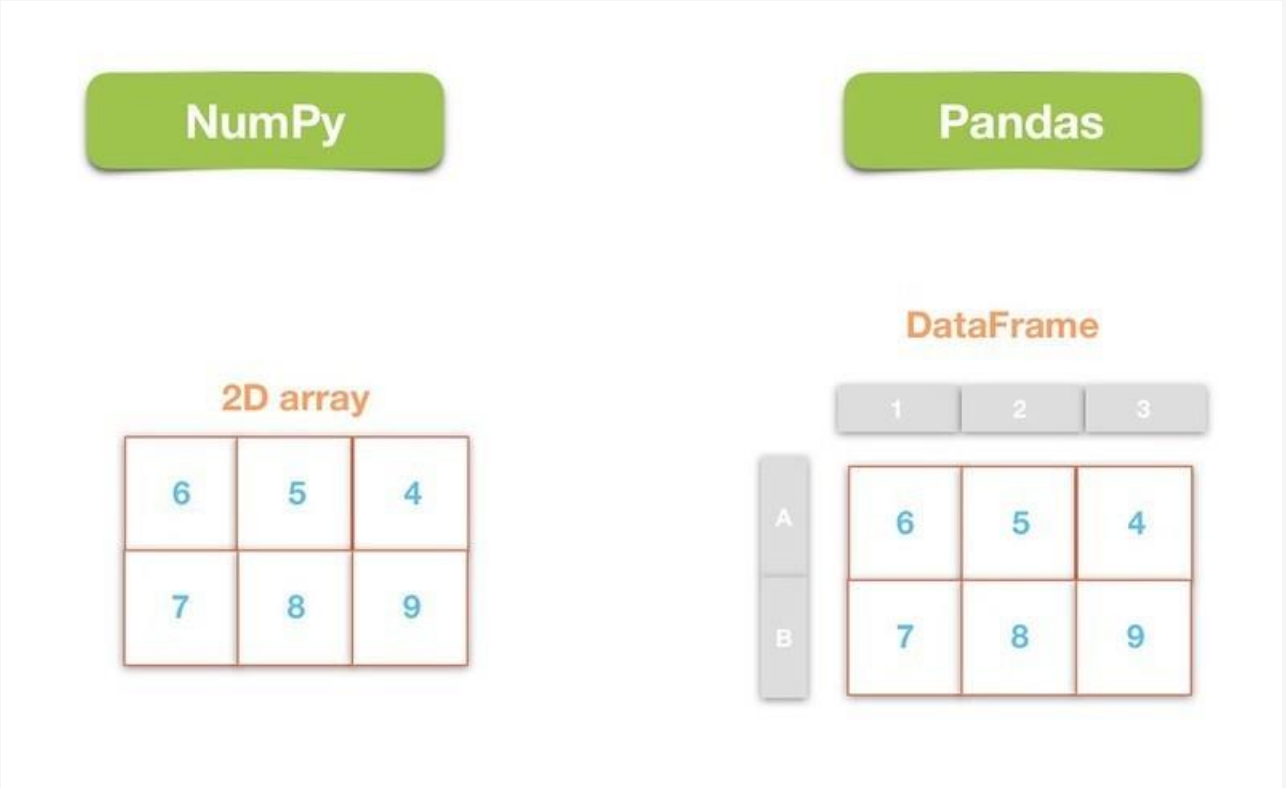
Usando esta librería podemos lograr cinco pasos típicos en el procesamiento y análisis de datos, independientemente del origen de los datos: cargar, preparar, manipular, modelar y analizar.

Pero hablemos de que se trata un DataFrame que es la estructura fundamental de Pandas, estos son estructuras de datos etiquetados bidimensionales con columnas de tipos potencialmente diferentes. Los Pandas DataFrame consta de tres componentes principales: **los datos, el índice y las columnas.**

Numpy vs Pandas

DataFrame son estructuras de dos dimensiones con columnas potencialmente diferentes que cuentan con índices tanto en las columnas como en las filas y que, por supuesto, pueden ser manipuladas por el programador.

Otras características importantes de los DataFrame es que tiene datos heterogéneos y el tamaño y los datos pueden ser mutables.



Características principales de la librería Pandas:

- Objeto DataFrame rápido y eficiente con indexación predeterminada y personalizada.
- Herramientas para cargar datos en objetos de datos en memoria desde diferentes formatos de archivo.
- Alineación de datos y manejo integrado de datos faltantes.
- Remodelación y giro de conjuntos de fechas.

Características principales de la librería Pandas:

- Etiquetado, corte, indexación y subconjunto de grandes conjuntos de datos.
- Las columnas de una estructura de datos se pueden eliminar o insertar.
- Agrupa por datos para agregación y transformaciones.
- Alto rendimiento de fusión y unión de datos.
- Funcionalidad de la serie de tiempo.

Instalación

La instalación estándar de Python no viene con la librería Pandas instalada por lo que se utilizar el paquete de instalación, pip.

En caso de que estés utilizando el paquete de Python Anaconda, no te tienes que preocupar, esta viene instalada por defecto.

pip install pandas

Importar

Para utilizar Pandas en tus programas debes importarla utilizando el alias pd, esta es el estándar que se utiliza para esta librería, como lo es np, para la librería NumPy.

Recuerda que cuando codificas en tu propio entorno de Machine Learning no debes olvidar este paso tan importante.

```
importar pandas como pd
```

DataFrames

Crear los DataFrames es el primer paso en cualquier proyecto de Machine Learning con Python, por lo que para crear una trama de datos desde una matriz NumPy debes pasarla a la función DataFrame() en el argumento de datos.

```
import numpy as np
import pandas as pd
data = np.array([['','Col1','Col2'], ['Fila1',11,22], ['Fila2',33,44]])
print(pd.DataFrame(data = data[1:,1:], index = data[1:,0], columns = data[0,1:]))
```

DataFrames

Si observamos el resultado de este código, se fragmenta los elementos seleccionados de la matriz NumPy para construir el DataFrame, primero se selecciona los valores que figuran en las listas que comienza con Fila1 y Fila2, luego selecciona el índice o los números de fila Fila1 y Fila2 y luego los nombres de las columnas Col1 y Col2.

La manera en que creamos este DataFrame será la misma para todas las estructuras.

```
df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]))
print('DataFrame:')
print(df)
```

Series

A su vez crear un serie es sencillo.

```
import pandas as pd
series = pd.Series({"Argentina":"Buenos Aires",
                    "Chile":"Santiago de Chile", "Colombia":"Bogotá",
                    "Perú":"Lima"})
print('Series:')
print(series)
```