**¿Por qué NumPy y Pandas?**

**¿Por qué NumPy?**

Es una librería enfocada al cálculo númerico y manejo de Arrays.

* Es muy veloz, hasta 50 veces más rápido que usar una lista de Python o C.
* Optimiza el almacenamiento en memoria.
* Además, maneja distintos tipos de datos.

Es una librería muy poderosa, se pueden crear redes neuronales desde cero.

**¿Por qué Pandas?**

Pandas está enfocada a la manipulación y análisis de datos.

* Al estar construido sobre NumPy veloz.
* Requiere poco código para manipular los datos.
* Soporta múltiples formatos de archivos.
* Ordena los datos en una alienación inteligente.

Se pueden manejar grandes cantidades de datos, hacer analítica y crear dahsboards.

La forma de importar estas librerías es de la siguiente manera:

import numpy as np

import pandas as pd

## NumPy Array

El array es el principal objeto de la librería. Representa datos de manera estructurada y se puede acceder a ellos a traves del indexado, a un dato específico o un grupo de muchos datos específicos.

lista = [1, 2 , 3, 4, 5, 6, 7, 8, 9]

lista

---> [1, 2, 3, 4, 5, 6, 7, 8, 9]

arr = np.array(lista)

type(arr)

---> numpy.ndarray

matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

matriz = np.array(matriz)

matriz

---> array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

El indexado nos permite acceder a los elementos de los array y matrices  
Los elementos se emepiezan a contar desde 0.

arr[0]

---> 1

Es posible operar directamente con los elementos.

arr[0] + arr[5]

---> 7

En el caso de las matrices al indezar una posición se regresa el array de dicha posición.

matriz[0]

---> array([1, 2, 3])

Para seleccionar un solo elemento de la matriz se especifica la posición del elemento separada por comas.

matriz[0, 2]

---> 3

El slicing nos permite extraer varios datos, tiene un comienzo y un final.  
En este ejemplo se está extrayendo datos desde la posición 1 hasta la 5. [1:6].

arr[1:6]

---> array([2, 3, 4, 5, 6])

Si no se ingresa el valor de Start se toma el incio como la posición 0.

arr[:6]

---> array([1, 2, 3, 4, 5, 6])

En cambio si no se le da una posción de End se regresan todos los elementos hasta el final del array.

arr[2:]

---> array([3, 4, 5, 6, 7, 8, 9])

También se puede trabajar por pasos.  
En este ejemplo de 3 en 3.  
Regresa la posición 0, 0 + 3, 3 + 3 y como no hay posición 6 + 3, no se regrese nada.

arr[::3]

---> array([1, 4, 7])

Cuando se le asigna un valor negativo se regresan los valores comenzando desde la última posición del array.

arr[-1]

---> 9

arr[-3:]

---> array([7, 8, 9])

Para el caso de las matrices sucede algo similar.  
Para acceder a los valores a nivel de filas.

matriz[1:]

---> array([[4, 5, 6],

[7, 8, 9]])

Para acceder a los valores a nivel de filas y columnas.

matriz[1:, 0:2]

---> array([[4, 5],

[7, 8]])

## Tipos de datos

Los arrays de NumPy solo pueden contener un tipo de dato, ya que esto es lo que le confiere las ventajas de la optimización de memoria.

Podemos conocer el tipo de datos del array consultando la propiedad .dtype.

arr = np.array([1, 2, 3, 4])

arr.dtype

---> dtype('int64')

Si queremos usar otro tipo de dato lo podemos definir en la declaración del array.

arr = np.array([1, 2, 3, 4], dtype = 'float64')

arr.dtype

---> dtype('float64')

Ahora vemos que los valores están con punto decimal.

arr

---> array([1., 2., 3., 4.])

Si ya se tiene el array definido se usa el método .astype() para convertir el tipo de dato.

arr = np.array([1, 2, 3, 4])

arr = arr.astype(np.float64)

arr

---> array([1., 2., 3., 4.])

También se puede cambiar a tipo booleano recordando que los números diferentes de 0 se convierten en True.

arr = np.array([0, 1, 2, 3, 4])

arr = arr.astype(np.bool\_)

arr

---> array([False, True, True, True, True])

También podemos convertir los datos en tipo string.

arr = np.array([0, 1, 2, 3, 4])

arr = arr.astype(np.string\_)

arr

---> array([b'0', b'1', b'2', b'3', b'4'], dtype='|S21')

De igual forma se puede pasar de string a numero.

arr = np.array(['0', '1', '2', '3', '4'])

arr = arr.astype(np.int8)

arr

---> array([0, 1, 2, 3, 4], dtype=int8)

Si un elemento no es de tipo número el método falla.

arr = np.array(['hola','0', '1', '2', '3', '4'])

arr = arr.astype(np.int8)

arr

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-30-b9bb95861c7b> in <module>()

1 # DSi un elemento no es de tipo número el método falla.

2 arr = np.array(['hola','0', '1', '2', '3', '4'])

----> 3 arr = arr.astype(np.int8)

4 arr

ValueError: invalid literal for int() with base

10: 'hola'

**Dimensiones**

* scalar: dim = 0 Un solo dato o valor
* vector: dim = 1 Listas de Python
* matriz: dim = 2 Hoja de cálculo
* tensor: dim > 3 Series de tiempo o Imágenes

Declarando un escalar.

scalar = np.array(42)

print(scalar) ----> 42

scalar.ndim ----> 0

Declarando un vector.

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

print(vector) ----> [1 2 3]

vector.ndim ----> 1

Declarando una matriz.

matriz = np.array([[1, 2, 3], [4, 5, 6]])

print(matriz)

matriz.ndim

---->[[1 2 3]

[4 5 6]]

----> 2

Declarando un tensor.

tensor = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]],[[13, 13, 15], [16, 17, 18], [19, 20, 21], [22, 23, 24]]])

print(tensor)

tensor.ndim

----> [[[ 1 2 3]

[ 4 5 6]

[ 7 8 9]

[10 11 12]]

[[13 13 15]

[16 17 18]

[19 20 21]

[22 23 24]]]

----> 3

**Agregar o eliminar dimensiones**

Se puede definir el número de dimensiones desde la declaración del array

vector = np.array([1, 2, 3], ndmin = 10)

print(vector) ----> [[[[[[[[[[1 2 3]]]]]]]]]]

vector.ndim ----> 10

Se pueden expandir dimensiones a los array ya existentes.  
Axis = 0 hace refencia a las filas, mientras que axis = 1 a las columnas.

expand = np.expand\_dims(np.array([1, 2, 3]), axis = 0)

print(expand) ----> [[1 2 3]]

expand.ndim ----> 2

Remover/comprimir las dimensiones que no están siendo usadas.

print(vector, vector.ndim) ----> [[[[[[[[[[1 2 3]]]]]]]]]] 10

vector\_2 = np.squeeze(vector)

print(vector\_2, vector\_2.ndim) ----> [1 2 3] 1

**Reto**

1. Definir un tensor de 5 dimensiones
2. Sumar una dimensión en algún eje
3. Borrar dimensiones que no se usen

tensor5 = np.array([[[[1, 2],[3, 4]], [[5, 6],[7, 8]]], [[[1, 2],[3, 4]], [[5, 6],[7, 8]]]], ndmin = 5)

print(tensor5, tensor5.ndim)

[[[[[1 2]

[3 4]]

[[5 6]

[7 8]]]

[[[1 2]

[3 4]]

[[5 6]

[7 8]]]]] 5

expand5 = np.expand\_dims(tensor5, axis=1)

print(expand5, expand5.ndim)

[[[[[[1 2]

[3 4]]

[[5 6]

[7 8]]]

[[[1 2]

[3 4]]

[[5 6]

[7 8]]]]]] 6

print(expand5, expand5.ndim)

reduced5 = np.squeeze(expand5)

print(reduced5, reduced5.ndim)

[[[[[[1 2]

[3 4]]

[[5 6]

[7 8]]]

[[[1 2]

[3 4]]

[[5 6]

[7 8]]]]]] 6

[[[[1 2]

[3 4]]

[[5 6]

[7 8]]]

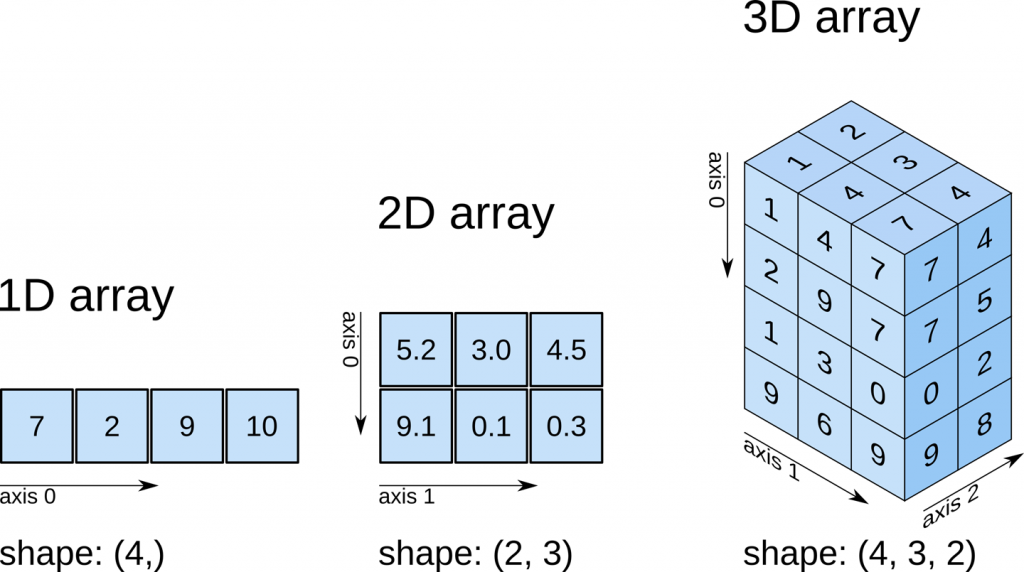
[[[1 2]

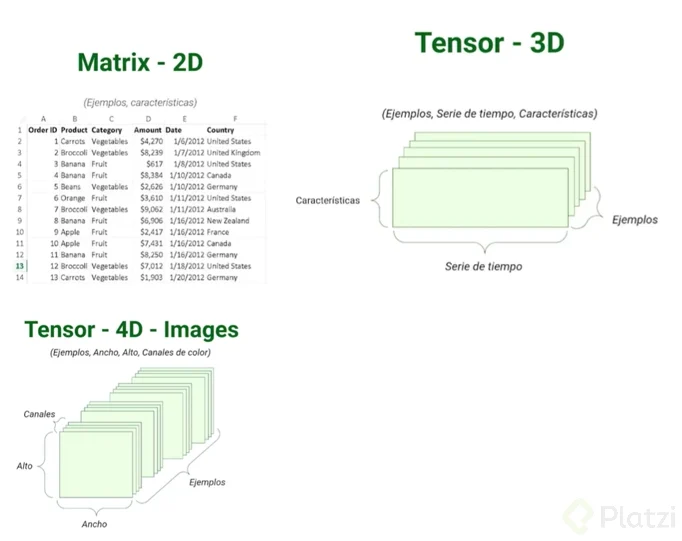
[3

4]]

[[5 6]

[7 8]]]] 4





#### Creando Arrays

1 ✅ np.**arange** (Start,Ens,Steps) → es como el list( range(0,10)) pero como arrange  
2 ✅ np.**zeros**(n)  
3 ✅ np.**ones**(n)  
4 ✅ np.**linspace**(Start, End, Cant n de Start a End)  
5 ✅ np.**eye**(n) ·· Matriz identidad

**Arrays con numeros randoms**

☑️ np.**random.rand**(Columnas, Filas, mas dimensiones ) ·· Ambos con numeros randoms  
☑️ np.**random.randint**(Start, End, Dimensiones) ·· N random entre Start y End y tupla dims

#### Shape y Reshape

Forma ····Como es la estructura del array···· y reforma del array

✅ arr\_shape = np.random.randint(Start, End,(3,2)) #Randoms entre 1 y 10 en una matriz (n, n)  
✅ arr\_shape.reshape(dim, dim)

#### Reto

#1 Crear un tensor / array con cualq dimensionalidad

#2 Intercambiar valores de array

#3 Crear un valor que no respete los valores originales

print('Generando la matriz...')

arr\_reto = np.random.randint(1,100,(4,6))

print(f'La matriz es esta\n{arr\_reto}\ntiene {arr\_reto.ndim}')

print()

print('Intercambiando las dimensiones + ajuste dimensiones')

print('A')

arr\_reto\_dims = np.squeeze(arr\_reto.reshape(1,24)) #Concatenando Squeeze con el reshape

print(f'La matriz es esta\n{arr\_reto\_dims}\ntiene {arr\_reto\_dims.ndim}')

print()

print('B')

arr\_reto\_dims =

np.squeeze(arr\_reto.reshape(8,3)) #Concatenando Squeeze con el reshape

print(f'La matriz es esta\n{arr\_reto\_dims}\ntiene {arr\_reto\_dims.ndim}')

print()

print('Generando un fallo intencional por incoherencia de dimensiones')

print('arr\_reto.reshape(4,10) --> le paso 40 elementos y son 24')

**Funciones principales de NumPy**

arr = np.random.randint(1, 20, 10)

matriz = arr.reshape(2,5)

matriz

----> array([[18, 8, 3, 11, 1],

[15, 10, 13, 9, 15]])

arr

----> array([18, 8, 3, 11, 1, 15, 10, 13, 9, 15])

arr.max() ----> 18

matriz.max() ----> 18

Podemos regresar los máximos de cada fila o columna especificando el eje.

matriz.max(1) ----> array([18, 15])

matriz.max(0) ---->array([18, 10, 13, 11, 15])

Tambien tenemos .argmax() que nos devuelve la posición del elemento

arr.argmax() ----> 0

matriz.argmax(0) ----> array([0, 1, 1, 0, 1])

De forma análoga tenemos .min()

arr.min() ----> 1

arr.argmin() ----> 4

matriz.min(0) ----> array([15, 8, 3, 9, 1])

matriz.argmin(1) ----> array([4, 3])

Podemos saber la diferencia de valor más bajo con el más alto.

arr.ptp() # 18 - 1 ----> 17

matriz.ptp(0) ----> array([ 3, 2, 10, 2, 14])

Para hacer análisis estádistico se tienen la siguientes funciones.

Ordenar los elementos:

arr.sort() ----> array([1, 3, 8, 9,10, 11,13, 15,15,18])

Obtener un percentil:

np.percentile(arr, 0) ----> 1.0

Mediana:

np.median(arr) ----> 10.5

Desviación estándar:

np.std(arr) ----> 5.080354

Varianza:

np.var(arr) ----> 25.81000

Promedio

np.mean(arr) ----> 10.3

Lo mismo aplica para las matrices.

np.median(matriz, 1) ----> array([ 8., 15.])

Se pueden unir dos arrays por medio de la concatenación

a = np.array([[1,2], [3,4]])

b= np.array([5, 6])

np.concatenate((a,b), axis = 0)

----------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-213-97c6fb2c6837> in <module>()

----> 1 np.concatenate((a,b), axis = 0)

<\_\_array\_function\_\_ internals> in concatenate(\*args, \*\*kwargs)

ValueError: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array at index 1 has 1 dimension(s)

El error anterior es debido a a tiene 2 dimensiones mientras que b tiene 1.

print(a.ndim) ----> 2

b.ndim ----> 1

b = np.expand\_dims(b, axis = 0)

np.concatenate((a,b), axis = 0)

----> array([[1, 2],

[3, 4],

[5, 6]])

De igual forma podemos agregarlo en el otro eje

np.concatenate((a,b), axis = 1)

-----------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-217-3ae7237876ab> in <module>()

1 # De igual forma podemos agregarlo en el otro eje

----> 2 np.concatenate((a,b), axis = 1)

<\_\_array\_function\_\_ internals> in concatenate(\*args, \*\*kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at index 0 has size 2 and the array at index 1 has size 1

Como b es una fila y no una columna, no se puede concatenar a menos que se aplique la transpuesta.

np.concatenate((a,b.T), axis = 1)

-

---> array([[1, 2, 5],

[3, 4, 6]])

**Copy**

.copy() nos permite copiar un array de NumPy en otra variable de tal forma que al modificar el nuevo array los cambios no se vean reflejados en array original.

arr = np.arange(0, 11)

arr ----> array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

arr[0:6] ----> array([0, 1, 2, 3, 4, 5])

trozo\_de\_arr = arr[0:6]

trozo\_de\_arr[:] = 0

trozo\_de\_arr ----> array([0, 0, 0, 0, 0, 0])

Se han modificado los datos del array original porque seguía haciendo referencia a esa variable.

arr ----> array([ 0, 0, 0, 0, 0, 0, 6, 7, 8, 9, 10])

arr\_copy = arr.copy()

arr\_copy[:] = 100

arr\_copy ----> array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100])

arr ----> array([ 0, 0, 0, 0, 0, 0, 6, 7, 8, 9, 10])

**Condiciones**

Las condiciones nos permiten hacer consultas más específicas.

arr = np.linspace(1,10,10, dtype = 'int8')

arr ----> array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=int8)

Regresa un array de booleanos donde la condiciones se cumple.

indices\_cond = arr > 5

indices\_cond

----> array([False, False, False, False, False, True, True, True, True, True])

Regresa los valores para donde la condiciones True.

arr[indices\_cond] ----> array([ 6, 7, 8, 9, 10], dtype=int8)

Se pueden agregar múltiples condiciones.

arr[(arr > 5) & (arr < 9)] ----> array([6, 7, 8], dtype=int8)

De igual forma modificar los valores que cumplan la condición.

arr[arr > 5] = 99

arr ----> array([ 1, 2, 3, 4, 5, 99, 99, 99, 99, 99], dtype=int8)

Otra funcion muy util de numpy tambien es np.where(condicion, valor si, valor si la condicion no se cumple), ejemplo:

matriz = array([[19, 4, 43],

[ 8, 96, 80],

[ 6, 99, 35]])

np.where(matriz > 50, 0, 1)

-->

array([[1, 1, 1],

[1, 0, 0],

[1, 0, 1]])

**Operaciones**

Existen diferentes operaciones que se pueden usar para los arrays de NumPy.

lista = [1,2]

lista ----> [1, 2]

Una lista de Python entiende que quieres duplicar los datos. Cosa que no buscamos.

lista \* 2 ----> [1, 2, 1, 2]

arr = np.arange(0,10)

arr2 = arr.copy()

arr ----> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

Ahora multiplicamos por un escalar:

arr \* 2 ----> array([ 0, 2, 4, 6, 8, 10, 12, 14, 16, 18])

Operación suma de escalar:

arr + 2 ----> array([ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

División con un escalar  
Como en este caso la primera posición del array es 0, muestra un error pero no detiene el proceso.

1/arr

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:1: RuntimeWarning: divide by zero encountered in true\_divide

"""Entry point for launching an IPython kernel.

array([ inf, 1. , 0.5 , 0.33333333, 0.25 ,

0.2 , 0.16666667, 0.14285714, 0.125 , 0.11111111])

Elevar a un escalar:

arr\*\*2 ----> array([ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81])

Sumar dos arrays de igual dimensiones las hace elemento por elemento:

arr + arr2 ----> array([ 0, 2, 4, 6, 8, 10, 12, 14, 16, 18])

Lo mismo aplica para matrices.

matriz = arr.reshape(2,5)

matriz2 = matriz.copy()

matriz

----> array([[0, 1, 2, 3, 4],

[5, 6, 7, 8, 9]])

matriz - matriz2

----> array([[0, 0, 0, 0, 0],

[0, 0, 0, 0, 0]])

Una operación importante es la de punto por punto, aquí dos formas de hacerla:

np.matmul(matriz, matriz2.T)

----> array([[ 30, 80],

[ 80, 255]])

matriz @ matriz2.T

----> array([[ 30, 80],

[

80, 255]])

**PANDAS**

Manipulación y análisis de datos. El nombre viene de “Panel data”.  
• Velocidad  
• Poco código  
• Múltiples formatos de archivos  
• Alineación inteligente

**Pandas Series**

Es muy parecido a un array de una dimensión (o vector) de NumPy.  
• Arreglo unidimensional indexado  
• Búsqueda por índice  
• Slicing  
• Operaciones aritméticas  
• Distintos tipos de datos

**Pandas DataFrame**

Muy parecido a las estructuras matriciales trabajadas con NumPy.  
• Estructura principal  
• Arreglo de dos dimensiones  
• Búsqueda por índice (columnas o filas)  
• Slicing  
• Operaciones aritméticas  
• Distintos tipos de datos  
• Tamaño variable

* Leer un archivo CSV. Primero debemos cargarlo al sistema de archivo del Notebooks que estamos utilizando (Deepnote o Jupiter). Luego hacemos el llamado con la siguiente linea:

import pandas as pd

df\_books = pd.read\_csv('ruta/nombre\_archivo')

df\_books -----> #muestra el archivo en formato DataFrame

* Si el .CSV esta separado por otro simbolo que no sea la coma, por ejemplo por punto y coma, podemos configurarlo para que el separador sea ;

df\_books = pd.read\_csv('ruta/nombre\_archivo', sep=';')

df\_books ------> #muestra el archivo, separado por (;), en formato dataFrame

* Cambiar el encabezado del archivo por una de sus columnas

df\_books = pd.read\_csv('ruta/nombre\_archivo', sep=',', header=1)

df\_books ------> #muestra el dataFrame con la primera columna como encabezado

df\_books = pd.read\_csv('ruta/nombre\_archivo', sep=',', header=0)

df\_books ------> #muestra el dataFrame con el encabezado que trae el archivo por defecto

* Sustituye los nombres del encabezado con una lista que contenga los nuevos nombres

df\_books = pd.read\_csv('ruta/nombre\_archivo', sep=',', header=0, names = ['title1', 'title2', 'titleN'])

df\_books -----> #muestra la dataFrame con los nuevos nombres de columnas dados

* Abrir un archivo json en formato dataFrame

df\_personajes = pd.read\_json('ruta/nombre\_archivo')

df\_personajes -----> #muestra el archivo json en formato dataFrame

* Abrir un archivo json en formato de Series

df\_personajes = pd.read\_json('ruta/nombre\_archivo', typ='Series')

df\_personajes ------> #muestra el archivo json en formato de Series

#### Iloc y loc

Permiten filtrar datos de manera mas específica. Loc filtra segun un label mientras que iloc lo hace mediante indices.

* Mostrar el dataFrame con loc

import pandas as pd

df\_books = pd.read\_csv('bestsellers-with-categories.csv', sep=',', header=0)

df\_books.loc[:] ----> #muestra todos los datos del dataFrame

* Mostrar un rango de filas tomando en cuenta el start y el end

df\_books.loc[0:4] ----> #muestra los datos de la fila 0 a la fila 4

* Filtrando por filas y columnas

df\_books.loc[0:4, ['Name', 'Author']]

----> #filtra los datos de la fila que va de 0 a 4 y de las columnas Name y Author

* Podemos modificar los valores de una columna especifica del dataFrame

df\_books.loc[:, ['Reviews']] \* -1

----> #multiplica por -1 todos los valores de la columna Reviews

* Filtrar datos que cumplan una condición determinada

df\_books.loc[:, ['Author']] == 'JJ Smith'

----> #muestra la columna Author con True en los valores que cumplen la condicion y False para los que no la cumplen

* Mostrar el dataFrame con iloc

df\_books.iloc[:] ---> #muestra todos los datos del dataframe

* Filtra datos segun los indices de las filas y las columnas

df\_books.iloc[:4, 0:2] ---> #muestra los datos de las filas que van de 0 a 3 y las columnas con indices 0 y 1

* Tambien podemos buscar un dato especifico con iloc

df\_boo

ks.iloc[1,3] ---> #muestra el dato alojado en la fila 1 columna 3

#### Agr

#### egar o eliminar datos de Pandas

* Mostrar las primeras 5 filas del dataFrame

import pandas as pd

import numpy as np

df\_books = pd.read\_csv('bestsellers-with-categories.csv', sep=',', header=0)

df\_books.head() ---> #muestra las primeras 5 lineas del dataFrame

* Eliminar columnas de la salida pero no del dataFrame

df\_books.drop('Genre', axis=1).head() #axis 1 = columnas. axis 2 = filas

----> #elimina la columna Genre de la salida pero no del dataFrame

* Eliminar una columna del dataFrame

del df\_books['Price']

----> #elimina la columna Price del dataFrame

* Eliminar filas del dataFrame

df\_books.drop(0, axis=0)

----> #elimina la fila 0 del dataFrame

* Eliminar un conjunto de filas mediante una lista

df\_books.drop([0,1,2], axis=0)

----> #elimina las filas 0, 1 y 2 del dataFrame

* Elimina un conjunto de filas mediante un rango

df\_books.drop(range(0,10), axis=0)

----> #elimina las primeras 10 filas del dataFrame

* Agregar una nueva columna con valores Nan

df\_books['Nueva\_columna'] = np.nan

----> #Crea una nueva columna con el nombre de Nueva\_columna de valores Nan

* Mostrar el numero de filas(0) o columnas(1) que tiene un dataFrame

df\_books.shape[0]

----> #Muestra el numero de filas que posee el dataFrame

* Agregar valores a una nueva columna del dataFrame

#Creamos una array con un numero de valores igual al numero de filas del dataFrame

data = np.arange(0, df\_books.shape[0])

#Creamos una nueva columna y agregamos los valores almacenados en el array

df\_books['Rango'] = data

----> #Crea una nueva columna llamada Rango con los valores del array

* Para añadir filas se utiliza la funcion append de python añadiendo como parametro una lista, diccionario o añadiendo los valores manualmente

df\_books.append(df\_books)

----> #Duplica las filas del dataFrame

#### Manejo de valores nulos

* Creamos un dataFrame con algunos valores nulos

import pandas as pd

import numpy as np

dict = {'Col1':[1,2,3,np.nan],

'Col2':[4, np.nan,6,7],

'Col3':['a','b','c', None]}

df = pd.DataFrame(dict)

----> Col1 Col2 Col3

0 1 4 a

1 2 nan b

2 3 6 c

3 nan 7 None

* Identificar valores nulos en un dataFrame

df.isnull()

----> Col1 Col2 Col3

0 false false false

1 false true false

2 false false false

3 true false true

* Identificar valores nulos con un valor numerico

df.isnull()\*1

----> Col1 Col2 Col3

0 0 0 0

1 0 1 0

2 0 0 0

3 1 0 1

* Sustituir los valores nulos por una cadena

df.fillna('Missing')

----> Col1 Col2 Col3

0 1.0 4.0 a

1 2.0 Missing b

2 3.0 6.0 c

3 Missing 7.0 Missing

* Sustituir valores nulos por una medida estadisticas realizada con los valores de las columnas

df.fillna(df.mean())

----> Col1 Col2 Col3

0 1 4 a

1 2 5.667 b

2 3 6 c

3 2 7 None

* Sustituir valores nulos por valores de interpolacion

df.interpolate()

----> Col1 Col2 Col3

0 1 4 a

1 2 5 b

2 3 6 c

3 3 7 None

* Eliminar valores nulos

df.dropna()

----> Col1 Col2 Col3

0 1 4 a

2 3 6 c

#### Filtrado por condiciones

Llamamos los datos de un archivo csv para manejarlos

import pandas as pd

df\_books = pd.read\_csv('bestsellers-with-categories.csv', sep=',', header=0)

df\_books.head(2) ---> #muestra los primeros dos registros del dataFrame

* Mostrar datos que sean mayores a cierto valor

mayor2016 = df\_books['Year'] > 2016

mayor2016

---> #muestra el dataFrame con valores booleanos. True para libros publicados desde el 2017

* Filtrar datos que sean mayores a cierto valor

df\_books[mayor2016]

---> #filtra los datos que cumplen con la condicion

* Tambien se puede colocar la condicion directamente como parametro

df\_books[df\_books['Year'] > 2016]

---> #filtra los datos que cumplen con la condicion

* Mostrar los datos que sean igual a cierto valor

genreFiction = df\_books['Genre'] == 'Fiction'

genreFiction ---> #muestra el dataFrame con valores booleanos. True para libros de tipo Fiction

* filtrado con varias condiciones

df\_books[genreFiction & mayor2016]

---> #Filtra aquellos libros que sean de tipo Fiction y que hayan sido publicado desde 2017

* Filtrado con negacion

df\_books[~mayor2016]

---> #Filtra aquellos libros publicado antes de 2017

### Funciones principales de Pandas 🐼

✅ **.head()** → trae los primeros datos  
✅ **.info()** → Columnas, indices, cuales noson nulos, tipo de dato que maneja  
✅ **.describe()** → Solo de las columnas numericas me arroja datos estadisticos [ media,max,miun,mediana,etc ]  
✅ **.memory\_usage()** → memoria utilizada  
✅ **.value\_counts()** → cuenta valores de una columna  
✅ **.drop\_duplicates()** → elimina los valores repetidos  
✅ **.sort\_values( columna para ordenar )** → Se puede ordenar de forma descendiente con la bandera **ascending=False**

#### Funciones principales de Pandas

* Mostrar las primeras dos lineas de registro

import pandas as pd

df\_books = pd.read\_csv('bestsellers-with-categories.csv', sep=',', header=0)

df\_books.head(2) ---> #muestra los primeros dos registros del dataFrame

* Mostrar los diferentes datos que contiene el dataFrame

df\_books.info()

--->

RangeIndex: 550 entries, 0 to 549 #numero de registro

Data columns (total 7 columns): #total de columnas

# Column Non-Null Count Dtype #tipos de cada columna

--- ------ -------------- -----

0 Name 550 non-null object

1 Author 550 non-null object

2 User Rating 550 non-null float64

3 Reviews 550 non-null int64

4 Price 550 non-null int64

5 Year 550 non-null int64

6 Genre 550 non-null object

dtypes: float64(1), int64(3), object(3)

memory usage: 30.2+ KB

* Obtener diferentes datos estadisticos de las columnas numericas.

df\_books.describe()

---> User.Rating Reviews Price Year

count 550 550 550 550

mean 4.618 11953.281 13.1 2014

std 0.226 11731.132 10.84 3.165

min 3.3 37 0 2009

25% 4.5 4058 7 2011

50% 4.7 8580 11 2014

75% 4.8 17253.25 16 2017

max 4.9 87841 105 2019

* Mostrar los ultimos 5 registros del dataFrame

df\_books.tail()

---> #muestra los ultimos 5 registros

* Obtener el uso de la memoria de cada columna

df\_books.memory\_usage(deep=True)

--->

Index 128

Name 59737

Author 39078

User Rating 4400

Reviews 4400

Price 4400

Year 4400

Genre 36440

dtype: int64

* Obtener la informacion de un registro especifico

df\_books.iloc[0]

---> #obtiene la informacion del registro con indice 0

Name 10-Day Green Smoothie Cleanse

Author JJ Smith

User Rating 4.7

Reviews 17350

Price 8

Year 2016

Genre Non Fiction

Name: 0, dtype: object

* Eliminar registros duplicados

df\_books.drop\_duplicates()

* Elimina el primer registro duplicado del dataFrame (orden ascendente)

df\_books.drop\_duplicates(keep='last')

* Ordenar los registros segun valores de la columna (orden ascendente)

df\_books.sort\_values('Year')

---> #ordena los valores de menor a mayor segun el año

* Ordenar los registros segun valores de la columna(orden descendente)

df\_books.sort\_values('Year', ascending=False)

---> #ordena lo

s valores de mayor a menor segun el año

#### Groupby

Permite agrupar datos en funcion de los demas . Es decir, hacer el analisis del dataframe en funcion de una de las columnas.

* llamamos el dataFrame que vamos a manipular

import pandas as pd

df\_books = pd.read\_csv('bestsellers-with-categories.csv', sep=',', header=0)

df\_books.head(2) ---> #muestra las dos primeras lineas del dataFrame

* Agrupar por Author y mostrar el conteo de los datos de las demas columnas

df\_books.groupby('Author').count()

---> Name User Rating Reviews Price Year Genre

\*\*Abraham Verghese 2 2 2 2 2 2

Adam Gasiewski 1 1 1 1 1 1

Adam Mansbach 1 1 1 1 1 1

Adir Levy 1 1 1 1 1 1\*\*

* Agrupar por Author y mostrar la media de los datos de las demas columnas

df\_books.groupby('Author').median()

---> User Rating Reviews Price Year

\*\*Abraham Verghese 4.6 4866 11 2010.5

Adam Gasiewski 4.4 3113 6 2017

Adam Mansbach 4.8 9568 9 2011

Adir Levy 4.8 8170 13 2019\*\*

* La columna Author, en los casos anteriores, pasa a ser el indice. Podemos usar loc y acceder a un dato especifico del dataFrame. Agrupar por autor y mostrar la suma de los valores de las demas columnas para William Davis

df\_books.groupby('Author').sum().loc['William Davis']

--->

User Rating 8.8

Reviews 14994.0

Price 12.0

Year 4025.0

Name: William Davis, dtype: float64

* Abrupar por author y mostrar la suma de los valores de las demas columnas. Colocar los indices que el dataFrame trae por defecto

df\_books.groupby('Author').sum().reset\_index()

---> Author User Rating Reviews Price Year

0 Abraham Verghese 9.2 9732 22 4021

1 Adam Gasiewski 4.4 3113 6 2017

2 Adam Mansbach 4.8 9568 9 2011

3 Adir Levy 4.8 8170 13 2019

* La funcion agg() permite aplicar varias funciones al dataFrame una vez agrupado segun una columna especifica. Agrupar por Author y mostrar el minimo y maximo de las demas columnas

df\_books.groupby('Author').agg(['min','max'])

---> #muestra cada columna dividida en dos: min y max. Estas contienen los valores maximo y minimo de la columna para cada Author

* Agrupar por Author, obtener el minimo y maximo de la columna Reviews y sumar los valores de la columna User Rating

df\_books.groupby('Author').agg({'Reviews':['min','max'], 'User Rating':'sum'})

---> Reviews min Reviews max User Rating

Abraham Verghese 4866 4866 9.2

Adam Gasiewski 3113 3113 4.4

Adam Mansbach 9568 9568 4.8

Adir Levy 8170 8170 4.8

* Agrupar por Author - Year y contar los valores de las demas columnas

df\_books.groupby(['Author','Year']).count()

---> Name User Rating Reviews Price Genre

('Abraham Verghese', 2010) 1 1 1 1 1

('Abraham Verghese', 2011) 1 1 1 1 1

('Adam Gasiewski', 2017) 1 1 1 1 1

('Adam Mans

bach', 2011) 1 1 1 1 1

#### Combinando dataFrames

Existen diferentes formas de fusionar dos dataFrames. Esto se hace a traves de la lógica de combinación como se muestra a continuacion

* **Left join**: da prioridad al dataFrame de la izquierda. Trae siempre los datos de la izquierda y las filas en comun con el dataFrame de la derecha.
* **Right join**: da prioridad al dataFrame de la derecha. Trae siempre los datos de la derecha y las filas en comun con el dataFrame de la izquierda.
* **Inner join**: Trae solamente aquellos datos que son común en ambos dataFrame
* **Outer join**: Trae los datos tanto del dataFrame de la izquierda como el de la derecha incluyendo los datos que comparten ambos.
* Concat - Axis 0: permite combinar dos dataframes a nivel de filas. Crecimiento vertical
* Concat - Axis 1: permite combinar dos dataframes a nivel de columnas. La organizacion por columnas no va a ser la misma para ambos dataFrames, por tanto, se crearan valores NaN para rellenar los espacios vacios. Crecimiento horizontal

# Merge y Concat

* **Aquí los diccionarios:**
* **CONCAT**
* df1 = pd.DataFrame({'A':['A0', 'A1', 'A2','A3'],
* 'B':['B0', 'B1', 'B2','B3'],
* 'C':['C0', 'C1', 'C2','C3'],
* 'D':['D0', 'D1', 'D2','D3']})
* df2 = pd.DataFrame({'A':['A4', 'A5', 'A6','A7'],
* 'B':['B4', 'B5', 'B6','B7'],
* 'C':['C4', 'C5', 'C6','C7'],
* 'D':['D4', 'D5', 'D6','D7']})
* **MERGE 1**
* izq = pd.DataFrame({'key' : ['k0', 'k1', 'k2','k3'],
* 'A' : ['A0', 'A1', 'A2','A3'],
* 'B': ['B0', 'B1', 'B2','B3']})
* der = pd.DataFrame({'key' : ['k0', 'k1', 'k2','k3'],
* 'C' : ['C0', 'C1', 'C2','C3'],
* 'D': ['D0', 'D1', 'D2','D3']})
* **MERGE 2**
* izq = pd.DataFrame({'key' : ['k0', 'k1', 'k2','k3'],
* 'A' : ['A0', 'A1', 'A2','A3'],
* 'B': ['B0', 'B1', 'B2','B3']})
* der = pd.DataFrame({'key\_2' : ['k0', 'k1', 'k2','k3'],
* 'C' : ['C0', 'C1', 'C2','C3'],
* 'D': ['D0', 'D1', 'D2','D3']})
* **MERGE 3**
* izq = pd.DataFrame({'key' : ['k0', 'k1', 'k2','k3'],
* 'A' : ['A0', 'A1', 'A2','A3'],
* 'B': ['B0', 'B1', 'B2','B3']})
* der = pd.DataFrame({'key\_2' : [
* 'k0', 'k1', 'k2',np.nan],
* 'C' : ['C0', 'C1', 'C2','C3'],
* 'D': ['D0', 'D1', 'D2','D3']})

Aporte para crear los diccionarios usados en la clase. Es un método mucho más complejo que simplemente copiar, pegar y modificar.

Sin embargo, lo aporto como ejemplo del uso de list y dictionary comprehensions en python nivel intermedio.

# genera lista con keys: 'A', 'B', 'C', 'D'

keys = list('ABCD')

# values será un list comprehension que contendrá una lista de listas:

# ['A0', 'A1', ..., 'A3'], ['B0', 'B1', ..., 'B3'], ...

# es un list comprehension anidado

values = [

# list comprehension anidado

# crea una lista del tipo: ['A0', 'A1', ..., 'A3']

[f'{letter}{number}' for number in range(4)]

# para las letras A, B, C, D, E, F

for letter in list('ABCDEF')

]

df3 = pd.DataFrame(

# dictionary comprehension:

# uso zip para crear tuplas de dos elementos que suministren

# pares de valores k:v al diccionario

{k:v for k,v in zip(keys, values)}

)

Diccionarios ocupados en esta clase

izq = pd.DataFrame({'A': ['A0','A1','A2'],

'B':['B0','B1','B2']},

index=['k0','k1','k2'])

der =pd.DataFrame({'C': ['C0','C1','C2'],

'D':['D0','D1','D2']},

index=['k0','k2','k3'])

**pivot\_table**

Esta función puede traer recuerdos a las personas interesadas en el mundo del SQL, ya que Oracle, PostgreSQL y otros motores de bases de datos la tienen implementada desde hace muchos años. Pivot, básicamente, transforma los valores de determinadas columnas o filas en los índices de un nuevo DataFrame, y la intersección de estos es el valor resultante.

Entiendo que esto puede sonar algo confuso, pero no te preocupes, todo queda mucho más claro con un ejemplo.

1. Para comenzar, crea un nuevo Jupyter Notebooks, puedes usar Google Colab o la notebook de tu preferencia que estés utilizando para este curso.
2. Carga el DataFrame que hemos usado en el curso:

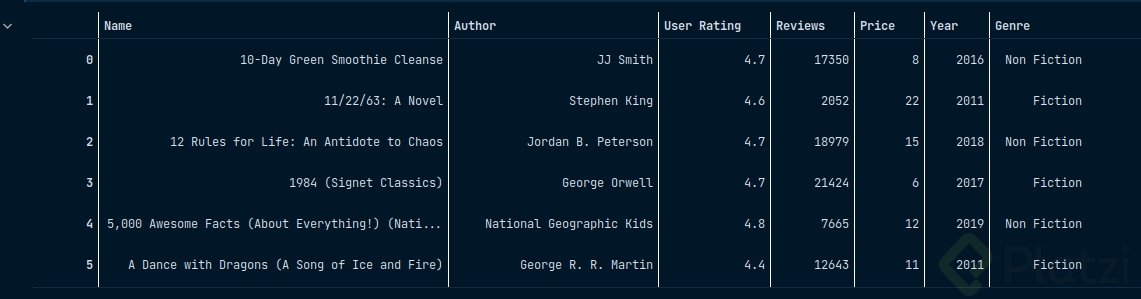
df\_books = pd.read\_csv('bestsellers with categories.csv',sep=',',header=0)

1. Explóralo viendo sus primeras 5 filas:

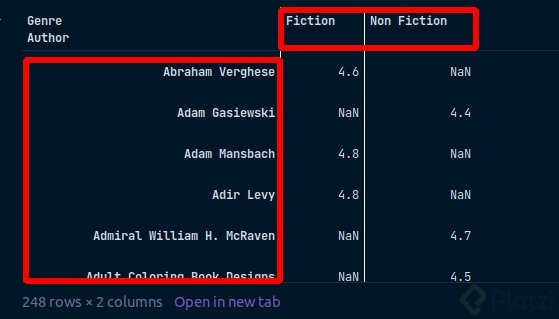
df\_books.head()

1. Aplica pivot\_table:

df\_books.pivot\_table(index='Author',columns='Genre',values='User Rating')



Como resultado, los valores de Author pasan a formar el índice por fila y los valores de Genre pasan a formar parte de los índices por columna, y el User Rating se mantiene como valor.

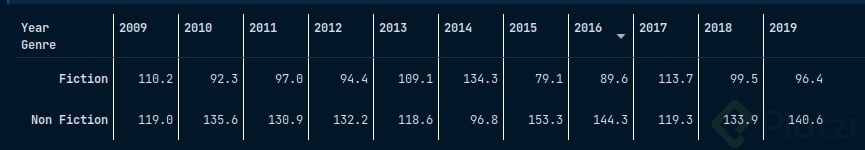


Por supuesto, para este caso, un Author suele tener un solo género literario, así que no es una transformación muy útil, pero veamos si podemos lograr algo mejor.

1. Ejecuta la siguiente variación:

df\_books.pivot\_table(index='Genre',columns='Year', values='User Rating',aggfunc='sum')

En este caso tenemos por cada género, la suma a lo largo de los años. Esto es mucho más interesante, ¿verdad? La mejor noticia es que no solo podemos obtener la suma, también podemos obtener la media, la desviación estándar, el conteo, la varianza, etc. Únicamente con cambiar el parámetro aggfunc que traduce función de agrupamiento.



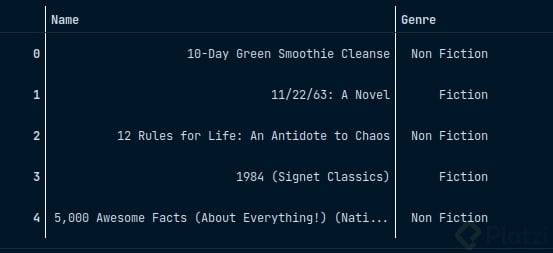
**melt**

El método melt toma las columnas del DataFrame y las pasa a filas, con dos nuevas columnas para especificar la antigua columna y el valor que traía.

Por ejemplo, simplemente al imprimir las cinco primeras filas del DataFrame con las columnas de Name y Genre se tiene este resultado.

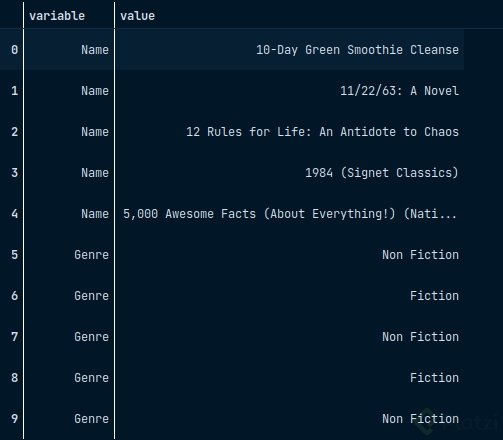
1. Para ello ejecuta la siguiente línea en tu Jupyter Notebook:

df\_books[['Name','Genre']].head(5)



1. Aplica melt de la siguiente manera:

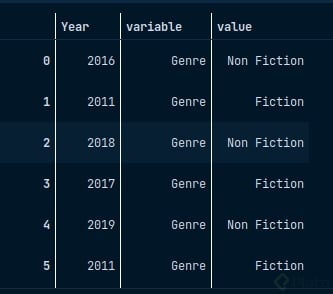
df\_books[['Name','Genre']].head(5).melt()



Ahora cada resultado de las dos columnas pasa a una fila de este modo a tipo **llave:valor**.

1. En el siguiente ejemplo ejecutemos melt de esta manera:

df\_books.melt(id\_vars='Year',value\_vars='Genre')



Simplemente, podemos seleccionar las columnas que no quiero hacer melt usando el parámetro id\_vars. Para este caso Year y también la única columna que quiero aplicar el melt, para este caso Genre con la propiedad value\_vars.