Numpy y Pandas

Introducción

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.
- Maneja distintos tipos de datos. Es una librería muy poderosa, se pueden crear redes neuronales desde cero.

Pandas:

Enfocada a la manipulación y analisis 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.

Para importar las librerias se debe realizar el siguiente codigo:

```
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.

Ejemplo:

Importaremos la libreria y crearemos dos arrays uno en una dimension y otra de 3:

```
import numpy as np

lista = [1,2,3,4,5,6,7,8,9]
lista = np.array(lista)

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

matriz = [[1,2,3],[4,5,6],[7,8,9]]
matriz = np.array(matriz)
```

```
# output ---> array([[1, 2, 3],
#                    [4, 5, 6],
#                    [7, 8, 9]])
```

Indexado:

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

```
lista[0]
# output ---> 1

lista[0] + lista[5]
# output ---> 7
```

Slicing:

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].

```
lista[0:3]
# output ---> array([1, 2, 3])

matriz[1:,0:2]
# output ---> 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
# output ---> dtype('int64')
```

como las redes neuronales trabajan mejor con arreglos tipo flotante (numeros decimales) entonces cambiaremos el formato del arreglo anterior con el siguiente codigo:

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

```
# output ---> dtype('float64')
```

otra manera de convertirlo es de la siguiente manera (directamente desde la libreria):

```
arr = arr.astype(np.float64)
arr.dtype
# output ---> dtype('float64')
```

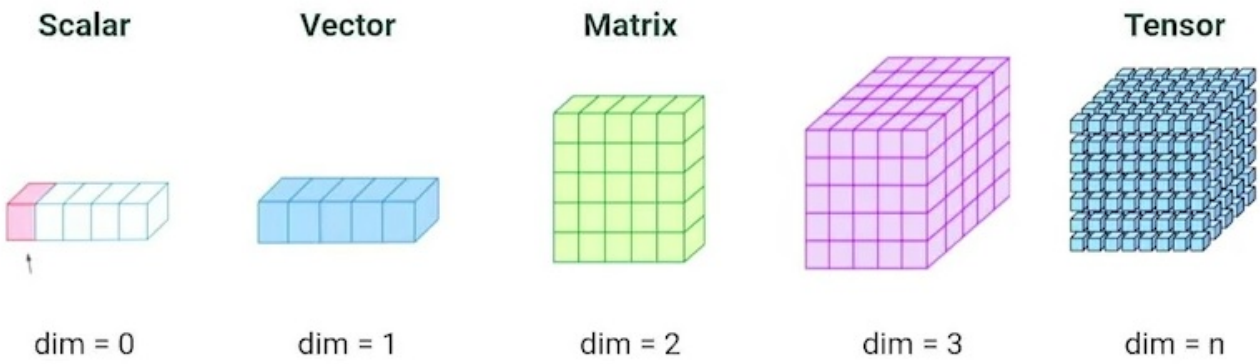
También se puede cambiar a tipo booleano recordando que los números diferentes de 0 se convierten en True.

```
arr = np.array([0, 1, 2, 3, 4])
arr = arr.astype(np.bool_)
arr
# output ---> array([False,  True,  True,  True,  True])
```

el resto de formatos los podemos consultar dando click AQUÍ.

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)
scalar.ndim
# output ---> 42
#        ---> 0
```

Declarando un Vector:

```
vector = np.array([1, 2, 3])
print(vector)
vector.ndim
# output ---> [1 2 3]
#        ---> 1
```

Declarando una matriz:

```
matriz = np.array([[1, 2, 3], [4, 5, 6]])
print(matriz)
matriz.ndim
# output ---> [[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
# output ---> [[[ 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 elimiar dimensiones:

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

```
vector = np.array([1, 2, 3], ndmin = 10)
print(vector)
vector.ndim
# output ---> [[[[[[[[[[1 2 3]]]]]]]]]]
#        ---> 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)
expand.ndim
# output ---> [[1 2 3]]
#        ---> 2
```

Para remover o comprimirdimensiones que no estan siendo usadas

```
print(vector, vector.ndim)
vector_2 = np.squeeze(vector)
print(vector_2, vector_2.ndim)
# output ---> [[[[[[[[[[1 2 3]]]]]]]]]] 10
#        ---> [1 2 3] 1
```

Creando Arrays:

Este métododе NumPy nos permite generar arrays sin definir previamente una lista.

```
np.arange(0,10)
# output ---> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Un tercer argumento permite definir un tamaño de paso.

```
np.arange(0,20,2)
# output ---> array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

np.zeros() nos permite definir estructuras o esquemas.

```
np.zeros(3)
# output ---> array([0., 0., 0.])
np.zeros((10,5))
# output ---> array([[0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.],
#                    [0., 0., 0., 0., 0.]])
```

De iugal forma tenemos np.ones()

```
np.ones(3)
# output ---> array([1., 1., 1.])
```

np.linspace() permite generar una arrary definiendo un incio, un final y cuantas divisiones tendrá.

```
np.linspace(0, 10 , 10)
# output ---> array([ 0.,1.11111111,2.22222222,  3.33333333,  4.44444444,
#                     5.55555556,  6.66666667,  7.77777778,  8.88888889, 10])
```

También podemos crear una matriz con una diagonal de 1 y el resto de 9.

```
np.eye(4)
# output ---> array([[1., 0., 0., 0.],
#                    [0., 1., 0., 0.],
#                    [0., 0., 1., 0.],
#                    [0., 0., 0., 1.]])
```

Otra metódo importante es generar números aleatorios.

```
np.random.rand()
# output --->0.37185218178880153
```

También se pueden generar vectores.

```
np.random.rand(4)
# output ---> array([0.77923054, 0.90495575, 0.12949965, 0.55974303])
```

Y a su vez generar matrices.

```
np.random.rand(4,4)
# output ---> array([[0.26920153, 0.24873544, 0.02278515, 0.08250538],
            #        [0.16755087, 0.59570639, 0.83604996, 0.57717126],
            #        [0.00161574, 0.27857138, 0.33982786, 0.19693596],
            #        [0.69474123, 0.01208492, 0.38613157, 0.609117  ]])
```

NumPy nos permite tambien generar números enteros. En este caso números enteros entre el 1 y 14

```
np.random.randint(1,15)
# output ---> 4
```

También podemos llevarlos a una estructura definida.

```
np.random.randint(1,15, (3,3))
# output ---> array([[ 8,  2,  6],
            #        [ 7,  1,  8],
            #        [11, 14,  4]])
```

Shape y Reshape

shape me indica la forma que tiene un arreglo, es decir, me indica con que estructura de datos estoy trabajando. Reshape transforma el arreglo mientras se mantengan los elementos.

```
arr = np.random.randint(1,10,(3,2))
arr.shape
# output ---> (3, 2)
```

Reshape

con el siguiente arreglo

```
array([[4, 9],
       [9, 2],
       [3, 4]])

# Aplicamos el Reshape

arr.reshape(1,6)

# output ---> array([[4, 9, 9, 2, 3, 4]])
```

otra forma de lograr el reshape

```
np.reshape(arr,(1,6))
# output ---> array([[4, 9, 9, 2, 3, 4]])
```

Se puede hacer un reshape como lo haría el lenguaje C.

```

```

```
np.reshape(arr,(2,3), 'C')
# output ---> array([[5, 6, 4],
#                    [6, 2, 3]])
```

También se puede hacer reshape a como lo haría Fortran.

```
np.reshape(arr,(2,3), 'F')
# output ---> array([[5, 2, 6],
#                    [4, 6, 3]])
```

Además existe la opción de hacer reshape según como esté optimizado nuestro computador. En este caso es como en C.

```
np.reshape(arr,(2,3), 'A')
# output ---> array([[5, 6, 4],
#                    [6, 2, 3]])
```

Funciones principales de NumPy

Trabajaremos con el siguiente vector y la siguiente matriz:

```
arr ---> array([ 4, 19, 16, 12,  5,  6, 19,  6,  1,  8])

matriz ---> array([[ 4, 19, 16, 12,  5],
                   [ 6, 19,  6,  1,  8]])
```

Función max()

```
arr.max()
# output ---> 19
```

si la quiero mostrar en la matriz por eje

```
matriz.max(0)
# output ---> array([ 6, 19, 16, 12,  8])
```

para mostrar en que indice se encuentra el valor maximo (me muestra el indice menor)

```
arr.argmax()
# output ---> 1
```

Función min()

nos entrega el menos valor, contiene las mismas funciones de max pero nos devuelve el menor valor

Función ptp()

me trae la diferencia numerica entre el valor maximo y el valor minimo

```
arr.ptp()
# output ---> 18
```

Función percentile()

Me especifica directamente el percentil que le indico

```
np.percentile(arr, 50)
# output ---> 7.0
```

Función sort()

Me ordena de mayor a menor la información del arreglo o matriz

```
arr.sort()
arr
# output ---> array([ 1,  4,  5,  6,  6,  8, 12, 16, 19, 19])
```

Función median()

Me entrega la mediana de un arreglo

```
np.median(arr)
# output ---> 7.0
```

Función std()

Me calcula la desviación estandar del arreglo

```

```

```
np.std(arr)
# output ---> 6.151422599691879
```

Función var()

Me calcula la varianza del arreglo

```
np.var(arr)
# output ---> 37.84

# desviación estandar ** 2 me da como resultado la varianza

np.std(arr) ** 2
# output ---> 37.84
```

Función mean()

Me calcula la media del arreglo

```
np.mean(arr)
# output ---> 9.6
```

Función concatenate()

me une o concatena la informacion de dos arreglos

trabajaremos con los siguientes arreglos

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

para unir el array **a** con el array **b**

```
np.concatenate((a,b))
# output ---> array([[1, 2],
#                    [3, 4],
#                    [5, 6]])
```

Si queremos concatenarlo de manera que nos quede en una matriz 2x3 debemos trasponer la información del array **b** ya que solo tiene una dimension; quedaria de la siguiente manera:

```
np.concatenate((a,b.T), axis=1)
# output --->array([[1, 2, 5],
#                   [3, 4, 6]])
```

- Existen más funciones estadisticas, para poder consultarlas dar click AQUÍ
- Y más funciones matemáticas dando click AQUÍ

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

arr_copy = arr.copy()
arr_copy[:] = 100

arr_copy
# output ----> array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
arr
# output ----> array([ 0,  0,  0,  0,  0,  0,  6,  7,  8,  9, 10])
```

Condiciones

son un conjunto de parametros que colocamos al hacer un slicing o un idexing dentro de un array con alguna condición, ejemplo quiero extraer del array los numeros pares.

nos permite realizar consultas al array mas especificas que sean dificil lograr con un simple slicing.

```
arr = np.linspace(1,10,10, dtype='int8')
# output ----> array([ 1,  2,  3,  4,  5,  6,  7,  8, 9, 10], dtype=int8)
```

si yo creo una vartiable con una condicion, ejemplo los numeros > a 5 del array

```
indices_cond = arr > 5
# output ----> array([False, False, False, False, False,  True,  True,  True,
True, True])
```

Si yo realizo slicing al array inicial "arr" poniendo como index el "indices_cond" tenemos como resultado la condición planteada dentro de la variable "indices_cond":

```
arr[indices_cond]
# output ----> array([ 6,  7,  8,  9, 10], dtype=int8)
```

otro ejemplo con dos condiciones:

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

De igual forma modificar los valores que cumplan la condición.

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

Operaciones

Cuando realizamos una multiplicacionen donde queremos aplicar el factor a cada elemento en una lista en python, el sistema entiende que lo que queremos es duplicar los datos, ejemplo:

```
lista = [1,2]
lista * 2
# output ---> [1, 2, 1, 2]
```

realicemos las operaciones desde numpy:

```
arr = np.arange(0,10)
arr2 = arr.copy() # realizamos copia para no modificar el array principal

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

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

Con numpy tambien podemos hacer operaciones entre arrays

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

Se pueden hacer operaciones de producto punto entre vectores con las dos siguientes maneras:

```
np.matmul(matriz, matriz2.T)
# output ---> array([[ 30,  80],
#                    [ 80, 255]])
matriz @ matriz2.T
# output ---> array([[ 30,  80],
#                    [ 80, 255]])
```

- para poder conocer el concepto de producto punto haz click AQUÍ
- Para poder ver mas operaciones con numpy hacer click AQUÍ

Pandas

Series y Dataframes

Series: es un objeto de pandas parecido a un array unidimensional ya que puedo indexar, hacer slicing, hacer operaciones manetaticas y admite muchos tipos de datos.

Dataframes: Son estructuras matriciales, ya que tenemos filas y columnas, las filas y las columnas tienen indices. con este podemos tambien indexar, hacer slicing, hacer operaciones manetaticas y admite muchos tipos de datos.

la libreria se debe importar siempre que queramos trabajar con pandas

```
import pandas as pd
```

```
psg_players =pd.Series(['Navas','Mbappe','Neymar','Messi'],
         index=[1,7,10,30])

# Output
# 1      Navas
# 7     Mbappe
# 10    Neymar
# 30     Messi
# dtype: object
```

puedo hacer slicing e indexing

```

```

```
psg_players[1]
# Output ---> 'Navas'

psg_players[0:3]
# Output
# 1      Navas
# 7     Mbappe
# 10    Neymar
# dtype: object
```

En el caso del mismo ejemplo anterior, si queremos realizar un dataframe, vamos a crear un diccionario con las caracteristicas de los jugadores:

```
jugadores_psg ={'Jugadores':['Navas','Mbappe','Neymar','Messi'],
 'Altura':[165.0,185.0,175.0,169.0],
 'Goles':[2,150,200,450]}
```

creamos a continuación del dataframe con el siguiente codigo:

```
df_players = pd.DataFrame(jugadores_psg, index=[1,7,10,30])
```

	Jugadores	Altura	Goles
1	Navas	165.0	2
7	Mbappe	185.0	150
10	Neymar	175.0	200
30	Messi	169.0	450

podemos acceder a los dataframes con indices de filas y columnas, para reconocerlos realizamos los siguientes comandos:

```
df_players.columns
# Output ---> Index(['Jugadores', 'Altura', 'Goles'], dtype='object')

df_players.index
# Output ---> Int64Index([1, 7, 10, 30], dtype='int64')
```

Leer Archivos CSV y JSON con Pandas

Podemos leer archivos externos para convertirlos en dataframes y poder trabajar con ellos en formato de filas y columnas:

```
df_books = pd.read_csv('bestsellers-with-categories.csv', sep=',',header=0)
df_books
```

	Name	Author	User Rating	Reviews	Price	Year	Genre
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction
3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12	2019	Non Fiction
...	...	...	...	...	...	...	...
545	Wrecking Ball (Diary of a Wimpy Kid Book 14)	Jeff Kinney	4.9	9413	8	2019	Fiction
546	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2016	Non Fiction
547	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2017	Non Fiction
548	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2018	Non Fiction
549	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2019	Non Fiction

550 rows × 7 columns

Filtrado con Loc y Iloc

Como con pandas podemos realizar slicing e indexing tal y como se realiza con la libreria de numpy podemos referenciar filas y columnas de la siguiente manera:

```
df_books[0:4]
```

	Name	Author	User Rating	Reviews	Price	Year	Genre
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction
3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction

```
df_books[['Name','Author']]
```

	Name	Author
0	10-Day Green Smoothie Cleanse	JJ Smith
1	11/22/63: A Novel	Stephen King
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson
3	1984 (Signet Classics)	George Orwell
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids
...	...	...
545	Wrecking Ball (Diary of a Wimpy Kid Book 14)	Jeff Kinney
546	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero
547	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero
548	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero
549	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero

550 rows × 2 columns

Al utilizar loc tenemos un slicing con un filtrado de columnas:

```
df_books.loc[0:4, ['Name','Author']]
```

	Name	Author
0	10-Day Green Smoothie Cleanse	JJ Smith
1	11/22/63: A Novel	Stephen King
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson
3	1984 (Signet Classics)	George Orwell
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids

En la siguiente linea de codigo voy a traerme una columna que me dirá que indice tiene como autor Stephen King:

```
df_books.loc[:, ['Author']] == 'Stephen King'
```

por otro lado el iloc me realiza consultas por indice, ejemplo:

```
df_books.iloc[0:5,0:3] # el primer index me trae la cantidad de filas, el segundo
la cantidad de columnas.
```

	Name	Author	User Rating
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7
1	11/22/63: A Novel	Stephen King	4.6
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7
3	1984 (Signet Classics)	George Orwell	4.7
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8

Agregar o eliminar datos con Pandas

Drop columns

```
df_books.drop('Genre', axis=1, inplace=True)
```

- axis=1 me indica que voy a borrar una columna
- inplace=True me borra la columna del dataframe completamente, si no especifico esta función dentro del código solo me hará el drop a la visualización mas no al dataframe.

otra forma de borrar sin usar implace es hacer referencia a la misma variable.

```
df_books = df_books.drop('Year', axis=1)
```

Python tiene su propia función de borrado que también me sirve en este caso pero no es muy recomendado

```
del df_books['Price']
```

Drop Rows

haciendo referencia al axis=0 borramos columnas:

```
df_books.drop(0, axis=0, inplace=True)
```

podemos borrar una lista de indices:

```
df_books.drop([0,1,2], axis=0, inplace=True)
```

borrar una serie de filas con la funcion range:

```
df_books.drop(range(0,10), axis=0, inplace=True)
```

Add Columns

creamos una columna con la funcion nan de numpy (debemos tener la libreria de Numpy activa)

```
df_books['Nueva Columna'] = np.nan
```

	Name	Author	User Rating	Reviews	Price	Year	Genre	Nueva Columna
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction	NaN
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction	NaN
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction	NaN

Podemos añadir una nueva columna con una numeración desde 1 hasta el final del DataFrame

```
data = np.arange(1,df_books.shape[0] + 1)
df_books['row_number'] = data
```

	Name	Author	User Rating	Reviews	Price	Year	Genre	row_number
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction	1
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction	2
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction	3

shape me muestra la cantidad de filas del DataFrame

todo rango que quiera añadir debe tener la misma logitud de filas del DataFrame, de lo contrario tendremos un error.

Add Rows

La siguiente funcion es la mas rapida y eficiente al momento de añadir filas nuevas

```
df_books.append(df_books)
```

con este codigo estamos duplicando la cantidad de datos ya que estamos añadiendo con append el mismo DataFrame.

Manejo de datos Nulos

Tenemos el siguiente DataFrame de ejemplo:

	Col1	Col2	Col3
0	1.0	4.0	a
1	2.0	NaN	b
2	3.0	6.0	c
3	NaN	7.0	None

para visualizar los nulos usamos la siguiente función:

```
df.isnull()
```

	Col1	Col2	Col3
0	False	False	False
1	False	True	False
2	False	False	False
3	True	False	True

Lo llevaremos a un manejo de ceros y unos:

```
df.isnull()*1
```

	Col1	Col2	Col3
0	0	0	0
1	0	1	0
2	0	0	0
3	1	0	1

podemos llenar los valores nulos con un valor para ser reconocidos:

```
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

O tambien podemos llenar los valores numericos con una formula como la media

```
df.fillna(df.mean())
```

	Col1	Col2	Col3
0	1.0	4.000000	a
1	2.0	5.666667	b
2	3.0	6.000000	c
3	2.0	7.000000	None

otra manera de manejar datos numericos nulos

```
df.interpolate()
```

	Col1	Col2	Col3
0	1.0	4.0	a
1	2.0	5.0	b
2	3.0	6.0	c
3	3.0	7.0	None

o eliminar todo lo que esté nulo:

```
df.dropna()
```

Filtrado por Condiciones

Filtrar los libros cuyo año sea mayor a 2016:

```
df_books[df_books['Year'] > 2016]
```

tambien podemos hacerlo creando una vairable

```
mayor2016 = df_books['Year'] > 2016
df_books[mayor2016]
```

de esta manera podriamos crear muchas variables condición y juntarlas en una sola consulta de datos.

para negar una condición se añade el siguiente simbolo a la linea de codigo

```
df_books[~mayor2016]
```

Funciones Principales de Pandas

traer los n primeros registros de un DataFrame

```
df_books.head(5)
```

traer los n ultimos registros de un DataFrame

```
df_books.tail(5)
```

Para conocer la información del DataFrame lo hacemos con:

```
df_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 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
```

De las columnas numericas podría optener datos estadisticos basicos con la siguiente línea de código:

```
df_books.describe()
```

	User Rating	Reviews	Price	Year
count	550.000000	550.000000	550.000000	550.000000
mean	4.618364	11953.281818	13.100000	2014.000000
std	0.226980	11731.132017	10.842262	3.165156
min	3.300000	37.000000	0.000000	2009.000000
25%	4.500000	4058.000000	7.000000	2011.000000
50%	4.700000	8580.000000	11.000000	2014.000000
75%	4.800000	17253.250000	16.000000	2017.000000
max	4.900000	87841.000000	105.000000	2019.000000

Con el siguiente comando podremos ver el peso en memoria de cada una de las columnas del DataFrame, esta información es util para ver que tanto me podria demorar en iterar las columnas pasando un For

```
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
```

Para contar cuantas veces se repite el autor dentro del DataFrame:

```
df_books['Author'].value_counts()
```

```
Jeff Kinney                          12
Rick Riordan                         11
Gary Chapman                         11
Suzanne Collins                      11
American Psychological Association   10
                                     ..
Brené Brown                          1
Hillary Rodham Clinton                1
Chip Heath                            1
Delegates of the Constitutional
       1
Marty Noble                           1
Name: Author, Length: 248, dtype: int64
```

Para borrar duplicados realizamos la siguiente línea de código:

```
df_books.drop_duplicates()
```

Para ordenar los datos de manera descendiente realizamos la siguiente línea de código:

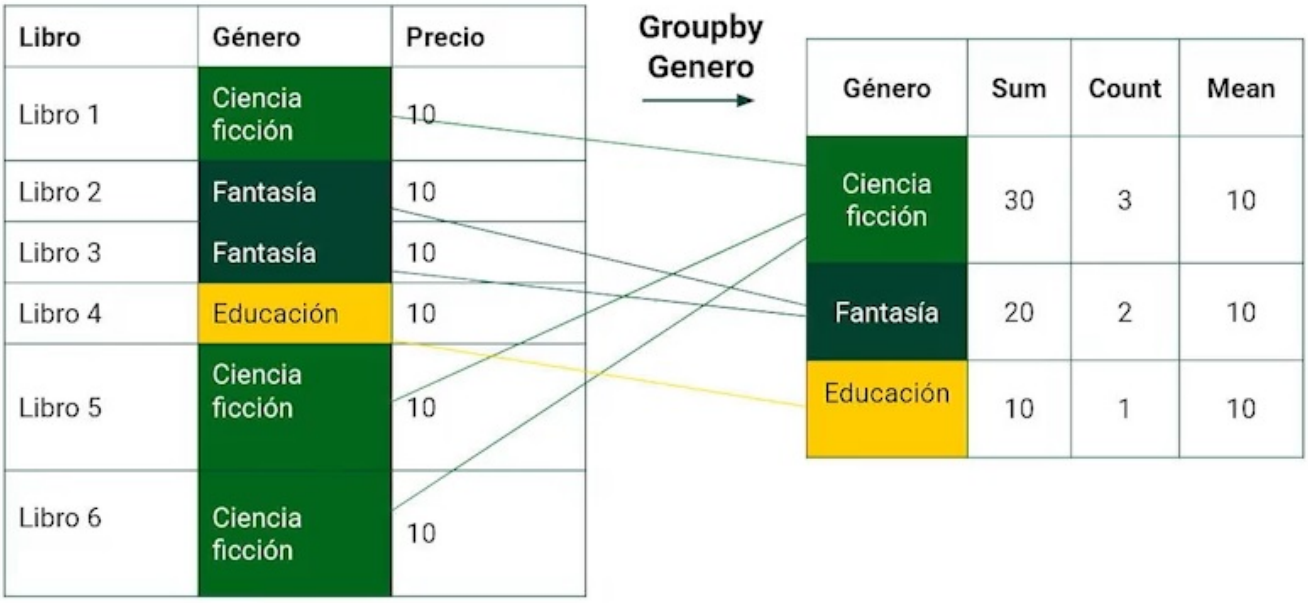
```
df_books.sort_values('Year', ascending=False)
```

El sort tiene por defecto el ordenamiento ascendenter por lo que si queremos ordenarlo de esta manera podriamos omitir el ascending en el código.

si quieres ver más formulas de Pandas da click AQUÍ.

Group by

Función de agrecación que nos sirve para poder visualizar elementos que se repiten en un DataFrame de manera agregada mediante sumas, conteos, promedios etc:




```
df_books.groupby('Author').count()
```

Author	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

Vemos en el resultado anterior que el título del autor quedó más abajo que el resto. Esto es porque despues de ejecutar ese codigo autor queda como index del DataFrame consultado y pdoriamos hacer consultas por label de la siguiente manera:

```
df_books.groupby('Author').count().loc['William Davis']
```

este nos traerá como resultado la información asociada a ese index.

por otra parte podemos hacer que el autor haga parte de otra columna con la siguiente línea de comando adicional:

```
df_books.groupby('Author').count().reset_index()
```

	Author	Name	User Rating	Reviews	Price	Year	Genre
0	Abraham Verghese	2	2	2	2	2	2
1	Adam Gasiewski	1	1	1	1	1	1
2	Adam Mansbach	1	1	1	1	1	1
3	Adir Levy	1	1	1	1	1	1

Si quiero realizar un analisis de más de una métrica podemos realizarlo con el siguiente código:

```
df_books.groupby('Author').agg(['min','max'])
```

	Name		User Rating		Reviews		Price		Year		Genre	
	min	max	min	max	min	max	min	max	min	max	min	max
Author												
Abraham Verghese	Cutting for Stone	Cutting for Stone	4.6	4.6	4866	4866	11	11	2010	2011	Fiction	Fiction
Adam Gasiewski	Milk and Vine: Inspirational Quotes From Class...	Milk and Vine: Inspirational Quotes From Class...	4.4	4.4	3113	3113	6	6	2017	2017	Non Fiction	Non Fiction
Adam Mansbach	Go the F**k to Sleep	Go the F**k to Sleep	4.8	4.8	9568	9568	9	9	2011	2011	Fiction	Fiction
Adir Levy	What Should Danny Do? (The Power to Choose Ser...	What Should Danny Do? (The Power to Choose Ser...	4.8	4.8	8170	8170	13	13	2019	2019	Fiction	Fiction

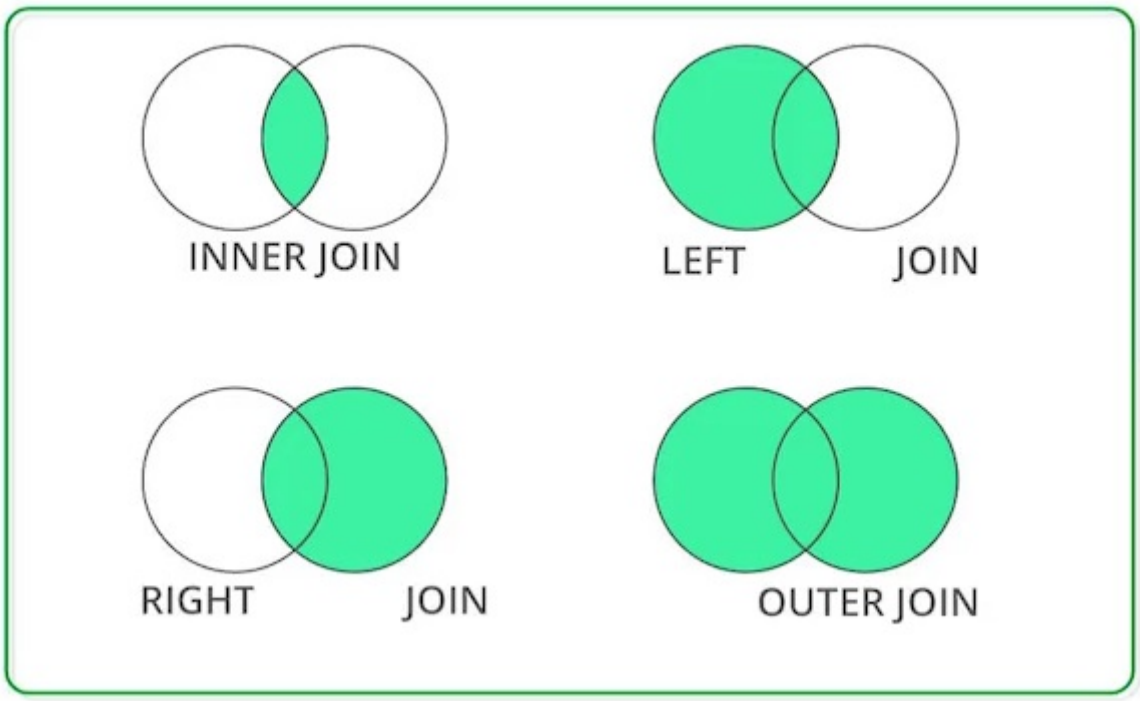
podemos realizar operaciones de agrecación con diferentes variables a analizar, es decir, quiero realizar la misma agregación por autor, pero, quiero sacar el minimo y el maximo de Reviews y el promedio del Rating:

```
df_books.groupby('Author').agg({'Reviews':['min','max'],'User Rating':'mean'})
```

	Reviews		User Rating
	min	max	mean
Author			
Abraham Verghese	4866	4866	4.600000
Adam Gasiewski	3113	3113	4.400000
Adam Mansbach	9568	9568	4.800000
Adir Levy	8170	8170	4.800000

Combinando DataFrames

Cuando queremos realizar una union entre dos dataframes ya que necesitamos información que albergan ambos podemos realizarlo de una manera simila a como se hace desde SQL:



Concat

Dados los siguientes dataframes (df1 y df1)

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

para concatenar los dos dataframes utilizamos el siguiente código

```
pd.concat([df1,df2], ignore_index=True)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

con el ignore_index=True reseteamos el indice del DataFrame ya que si no lo usamos mantendriamos los mismos indices de los DataFrames iniciales.

también se puede concatenar de manera horizontal:

	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	A4	B4	C4	D4
1	A1	B1	C1	D1	A5	B5	C5	D5
2	A2	B2	C2	D2	A6	B6	C6	D6
3	A3	B3	C3	D3	A7	B7	C7	D7

Merge

Dados los siguientes dataframes (izq y der)

	key	A	B
0	k0	A0	B0
1	k1	A1	B1
2	k2	A2	B2
3	k3	A3	B3

	key	C	D
0	k0	C0	D0
1	k1	C1	D1
2	k2	C2	D2
3	k3	C3	D3

```
izq.merge(der, on='key')
```

	key	A	B	C	D
0	k0	A0	B0	C0	D0
1	k1	A1	B1	C1	D1
2	k2	A2	B2	C2	D2
3	k3	A3	B3	C3	D3

si por el contrario tengo columnas con diferentes nombres se deberia unir especificando en el codigo los nombres de las columnas a unir, similar a un ON despues de un JOIN en SQL:

```
izq.merge(der, left_on='key', right_on='key_2')
```

	key	A	B	key_2	C	D
0	k0	A0	B0	k0	C0	D0
1	k1	A1	B1	k1	C1	D1
2	k2	A2	B2	k2	C2	D2
3	k3	A3	B3	k3	C3	D3

si tengo una llave nula en el DataFrame de la derecha la manera de conservar los datos de la columna izquierda es argumentando que quiero hacer un merge por la izquierda ya que por defecto me trae el inner (intersección):

```
izq.merge(der, left_on='key', right_on='key_2', how='left')
```

	key	A	B	key_2	C	D
0	k0	A0	B0	k0	C0	D0
1	k1	A1	B1	k1	C1	D1
2	k2	A2	B2	k2	C2	D2
3	k3	A3	B3	NaN	NaN	NaN

Join

la diferencia del join con el merge es que el join va a hacer la busqueda y el match con los indices de los DataFrames y el merge lo realiza con columnas específicas

Dados los siguientes dataframes (izq y der)

	A	B
k0	A0	B0
k1	A1	B1
k2	A2	B2

	C	D
k0	C0	D0
k2	C1	D1
k3	C2	D2

```
izq.join(der)
```

	A	B	C	D
k0	A0	B0	C0	D0
k1	A1	B1	NaN	NaN
k2	A2	B2	C1	D1

por defecto el join me hace una unión por la izquierda (left), para poder realizar joins diferentes se debe especificar el tipo de join en el código:

```
izq.join(der, how='right')
izq.join(der, how='inner')
izq.join(der, how='outer')
```

Pivot

Pivot transforma los valores de determinadas columnas o filas en los índices de un nuevo DataFrame, y la intersección de estos es el valor resultante.

Basandonos nuevamente el en DataFrame df_books:

```
df_books.pivot_table(index='Author',columns='Genre',values='User Rating')
```

Genre	Fiction	Non Fiction
Author		
Abraham Verghese	4.6	NaN
Adam Gasiewski	NaN	4.400000
Adam Mansbach	4.8	NaN
Adir Levy	4.8	NaN
Admiral William H. McRaven	NaN	4.700000

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.

```
df_books.pivot_table(index='Genre',columns='Year', values='User
Rating',aggfunc='sum')
```

Year	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Genre											
Fiction	110.2	92.3	97.0	94.4	109.1	134.3	79.1	89.6	113.7	99.5	96.4
Non Fiction	119.0	135.6	130.9	132.2	118.6	96.8	153.3	144.3	119.3	133.9	140.6

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.

```
df_books[['Name','Genre']].head(5).melt()
```

	variable	value
0	Name	10-Day Green Smoothie Cleanse
1	Name	11/22/63: A Novel
2	Name	12 Rules for Life: An Antidote to Chaos
3	Name	1984 (Signet Classics)
4	Name	5,000 Awesome Facts (About Everything!) (Natio...
5	Genre	Non Fiction
6	Genre	Fiction
7	Genre	Non Fiction
8	Genre	Fiction
9	Genre	Non Fiction

Apply

Es un método que nos ayudará a aplicar funciones definidas previamente a los DataFrames

por lo que definiremos la siguiente función

```
def two_times(value):
    return value * 2
```

Luego aplicaremos la función a una columna de nuestro DataFrame df_books:

```
df_books['User Rating'].apply(two_times)
```

Esta me multiplicará por 2 cada uno de los 'User Rating' y me lo hará mucho mas rápido que un for.

También me aplicará funciones lambda:

```
df_books['Rating_2'] = df_books['User Rating'].apply(lambda x : x * 3)
```

	Name	Author	User Rating	Reviews	Price	Year	Genre	Rating_2
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction	14.1
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction	13.8
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction	14.1
3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction	14.1
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12	2019	Non Fiction	14.4

Si quiero realizar una función mas compleja en donde el User Rating me lo multiplique por 2 si el genero es Fiction de lo contrario dejar el dato como está:

```
df_books['Rating_2'] = df_books.apply(lambda x : x['User Rating'] * 2 if
x['Genre'] == 'Fiction' else x['User Rating'], axis=1)
```

	Name	Author	User Rating	Reviews	Price	Year	Genre	Rating_2
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction	4.7
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction	9.2
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction	4.7
3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction	9.4
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12	2019	Non Fiction	4.8