**Contenido**

[1. CIENCIA DE DATOS, CON PANDAS, NUMPY, MATPLOTLIB, SEARBORN, MACHINE LEARNING. 4](#_Toc206786399)

[1.1 Pandas 4](#_Toc206786400)

[1.1.1 Características principales 5](#_Toc206786401)

[1.1.2 Tipos de datos de Pandas 5](#_Toc206786402)

[1.1.3 La clase de objetos Series 5](#_Toc206786403)

[1.1.4 Creación de series 6](#_Toc206786404)

[1.1.5 Creación de una serie a partir de una lista 6](#_Toc206786405)

[1.1.6 Creación de una serie a partir de un diccionario 6](#_Toc206786406)

[1.1.7 Atributos de una serie 7](#_Toc206786407)

[1.1.8 Acceso a los elementos de una serie 7](#_Toc206786408)

[1.1.9 Resumen descriptivo de una serie 8](#_Toc206786409)

[1.1.10 Aplicar funciones a una serie 11](#_Toc206786410)

[1.1.11 Filtrar una serie 11](#_Toc206786411)

[1.1.12 Ordenar una serie 12](#_Toc206786412)

[1.1.13 Eliminar los dados desconocidos en una serie 12](#_Toc206786413)

[1.1.14 La clase de objetos DataFrame 13](#_Toc206786414)

[1.1.15 Creación de un DataFrame 13](#_Toc206786415)

[1.1.16 Creación de un DataFrame a partir de una lista de listas 15](#_Toc206786416)

[1.1.17 Creación de un DataFrame a partir de una lista de diccionarios 15](#_Toc206786417)

[1.1.18 Creación de un DataFrame a partir de un array 16](#_Toc206786418)

[1.1.19 Creación de un DataFrame a partir de un fichero CSV o Excel 17](#_Toc206786419)

[1.1.20 Exportación de ficheros 17](#_Toc206786420)

[1.1.21 Atributos de un DataFrame 18](#_Toc206786421)

[1.1.22 Renombrar los nombres de las filas y columnas 22](#_Toc206786422)

[1.1.23 Cambiar el índice de un DataFrame 24](#_Toc206786423)

[1.1.24 Reindexar un DataFrame 25](#_Toc206786424)

[1.1.25 Acceso a los elementos de un DataFrame 26](#_Toc206786425)

[1.1.26 Operaciones con las columnas de un DataFrame 29](#_Toc206786426)

[1.1.27 Operaciones sobre columnas 30](#_Toc206786427)

[1.1.28 Aplicar funciones a columnas 32](#_Toc206786428)

[1.1.29 Convertir una columna al tipo datetime 32](#_Toc206786429)

[1.1.30 Resumen descriptivo de un DataFrame 33](#_Toc206786430)

[1.1.31 Eliminar columnas de un DataFrame 35](#_Toc206786431)

[1.1.32 Operaciones con las filas de un DataFrame 37](#_Toc206786432)

[1.1.33 Eliminar filas de un DataFrame 37](#_Toc206786433)

[1.1.34 Filtrar las filas de un DataFrame 39](#_Toc206786434)

[1.1.35 Ordenar un DataFrame 40](#_Toc206786435)

[1.1.36 Eliminar las filas con datos desconocidos en un DataFrame 41](#_Toc206786436)

[1.1.37 Agrupación de un DataFrame 42](#_Toc206786437)

[1.1.38 Dividir un DataFrame en grupos 42](#_Toc206786438)

[1.1.39 Aplicar una función de agregación por grupos 45](#_Toc206786439)

[1.1.40 Reestructurar un DataFrame 46](#_Toc206786440)

[1.1.41 Convertir un DataFrame a formato largo 46](#_Toc206786441)

[1.1.42 Convertir un DataFrame a formato ancho 47](#_Toc206786442)

[1.1.43 Combinar varios DataFrames 48](#_Toc206786443)

[1.1.44 Concatenación de DataFrames 48](#_Toc206786444)

[1.1.45 Mezcla de DataFrames 50](#_Toc206786445)

[1.1.46 Tablas dinámicas. 52](#_Toc206786446)

[1.2 Numpy. 53](#_Toc206786447)

[1.2.1 ¿Qué es Numpy? 53](#_Toc206786448)

[1.2.2 \*\*¿Qué ofrece Numpy?\*\* 53](#_Toc206786449)

[1.2.3 ¿Por qué utilizar Numpy?.\*\*. 53](#_Toc206786450)

[1.2.4 ¿Por qué Numpy es más rápido que las listas?. 54](#_Toc206786451)

[1.2.5 ¿En qué lenguaje está escrito Numpy?\*\*. 54](#_Toc206786452)

[1.2.6 ⚙️ Principales características. 54](#_Toc206786453)

[1.2.7 Creación de arrays 54](#_Toc206786454)

[1.2.8 Filtrado de elementos de un array 61](#_Toc206786455)

[1.2.9 Operaciones matemáticas con arrays 61](#_Toc206786456)

[1.2.10 Módulo de un vector 62](#_Toc206786457)

[1.2.11 Producto de dos matrices 62](#_Toc206786458)

[1.2.12 Matriz traspuesta 62](#_Toc206786459)

[1.2.13 Traza de una matriz 62](#_Toc206786460)

[1.2.14 Determinante de una matriz 63](#_Toc206786461)

[1.2.15 Matriz inversa 63](#_Toc206786462)

[1.2.16 Autovalores de una matriz 63](#_Toc206786463)

[1.2.17 Autovectores de una matriz 63](#_Toc206786464)

[1.2.18 Solución de un sistema de ecuaciones 63](#_Toc206786465)

[1.3 Matplotlib. 64](#_Toc206786466)

[1.3.1 🎯 ¿Para qué sirve?\*\* 64](#_Toc206786467)

[1.3.2 ⚙️ ¿Cómo funciona?\*\* 64](#_Toc206786468)

[1.3.3 Gráfico de línea. 65](#_Toc206786469)

[1.3.4 Grafica de pastel. 66](#_Toc206786470)

[1.3.5 Gráficos de líneas usando numpy. 67](#_Toc206786471)

[1.3.6 Graficas de barras verticales. 68](#_Toc206786472)

[1.3.7 Gráficas horizontales de barras. 68](#_Toc206786473)

[1.3.8 Graficas de dispersion. 69](#_Toc206786474)

[1.3.9 Histogramas 70](#_Toc206786475)

[1.3.10 gráficos de cajas. 71](#_Toc206786476)

[1.3.11 Gráficos a partir de una tabla dinámica 73](#_Toc206786477)

[1.3.12 Pandas y SQL. 75](#_Toc206786478)

[1.4 Polars. 76](#_Toc206786479)

[1.4.1 ¿Por qué usar polares? 77](#_Toc206786480)

[1.4.2 ¿Qué es Polars? 77](#_Toc206786481)

[1.4.3 ¿Para qué sirve? 77](#_Toc206786482)

[1.4.4 Optimización De Consultas 77](#_Toc206786483)

[1.4.5 Expresión Select 78](#_Toc206786484)

[1.4.6 Expresión Filter 79](#_Toc206786485)

[1.4.7 Expresión with columns 80](#_Toc206786486)

[1.4.8 Expresión group by 82](#_Toc206786487)

[1.4.9 Combinación De Expresiones 83](#_Toc206786488)

[1.4.10 Traducción De Expresiones Pandas A Polars 84](#_Toc206786489)

[1.4.11 Selección De Columnas 85](#_Toc206786490)

[1.4.12 Selección De Filas Por Índice 85](#_Toc206786491)

[2. Selección De Filas Por Filtro 85](#_Toc206786492)

[3. Creación De Columnas 85](#_Toc206786493)

[4. Unión De DataFrames 85](#_Toc206786494)

[5. Ordenar DataFrames 86](#_Toc206786495)

[6. Más Funciones Interesantes 86](#_Toc206786496)

[6.1.1 Ventajas de Polars. 87](#_Toc206786497)

[6.1.2 Desventajas de Polars 87](#_Toc206786498)

[6.2 Searborn. 88](#_Toc206786499)

[6.3 Machine Learning 88](#_Toc206786500)

# CIENCIA DE DATOS, CON PANDAS, NUMPY, MATPLOTLIB, SEARBORN, MACHINE LEARNING.

## Pandas

**¿Qué es pandas?**

¡Claro! Pandas es una biblioteca de Python muy popular utilizada para análisis y manipulación de datos.

Si tienes una pregunta específica sobre Pandas, como cómo usar una función, cómo leer un archivo Excel o CSV, o cómo filtrar o agrupar datos, dime y con gusto te ayudo.

**¿Por qué utilizar Pandas?**

Pandas nos permite analizar grandes datos y sacar conclusiones basadas en teorías estadísticas.

Pandas puede limpiar conjuntos de datos desordenados y hacerlos legibles y relevantes.

Los datos relevantes son muy importantes en la ciencia de datos.

### Características principales

* DataFrame: estructura bidimensional (como una hoja de Excel) con etiquetas en filas y columnas.
* Series: estructura unidimensional, como una columna de datos.
* Lectura de datos: desde CSV, Excel, JSON, SQL, etc.
* Filtrado y selección: accede a datos específicos con condiciones.
* Agrupación y agregación: ideal para análisis estadístico.
* Visualización: se integra con Matplotlib para crear gráficos.
* Permite leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos SQL.
* Permite acceder a los datos mediante índices o nombres para filas y columnas.
* Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
* Permite trabajar con series temporales.
* Realiza todas estas operaciones de manera muy eficiente.

### Tipos de datos de Pandas

Pandas dispone de tres estructuras de datos diferentes:

* Series: Estructura de una dimensión.
* DataFrame: Estructura de dos dimensiones (tablas).
* Panel: Estructura de tres dimensiones (cubos).

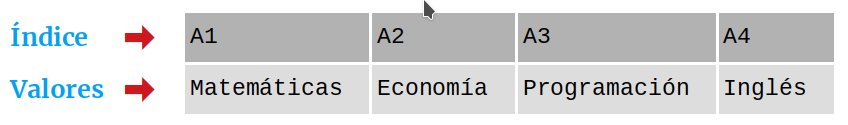
Estas estructuras se construyen a partir de **arrays** de la librería **Numpy**, añadiendo nuevas funcionalidades.

### La clase de objetos Series

Son estructuras similares a los **arrays** de una dimensión. Son homogéneas, es decir, sus elementos tienen que ser del mismo tipo, y su tamaño es inmutable, es decir, no se puede cambiar, aunque si su contenido.

Dispone de un índice que asocia un nombre a cada elemento de la serie, a través de la cual se accede al elemento.

Ejemplo. La siguiente serie contiene las asignaturas de un curso.



### Creación de series

### Creación de una serie a partir de una lista

**Series(data=lista, index=indices, dtype=tipo)** : Devuelve un objeto de tipo Series con los datos de la lista lista, las filas especificadas en la lista índices y el tipo de datos indicado en tipo. Si no se pasa la lista de índices se utilizan como índices los enteros del 0 al , done  es el tamaño de la serie. Si no se pasa el tipo de dato se infiere.

**Ejemplo básico**

Creación de series de tipo String.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

#Crear una lista

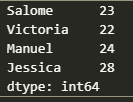
nombres = ['Salome', 'Victoria', 'Manuel', 'Jessica']

edades = [23, 22, 24, 28]

#Covertir la lista a una serie de datos.

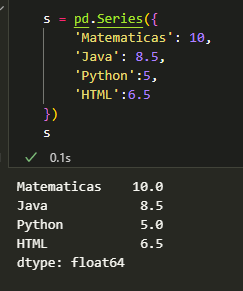
personas = pd.Series(edades, *index*=nombres)

personas



### Creación de una serie a partir de un diccionario

Series(data=diccionario, index=indices): Devuelve un objeto de tipo Series con los valores del diccionario diccionario y las filas especificadas en la lista índices. Si no se pasa la lista de índices se utilizan como índices las claves del diccionario.



### Atributos de una serie

Existen varias propiedades o métodos para ver las características de una serie.

* **s.size** : Devuelve el número de elementos de la serie s.
* **s.index :** Devuelve una lista con los nombres de las filas del DataFrame s.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* **s.dtype** : Devuelve el tipo de datos de los elementos de la serie s.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

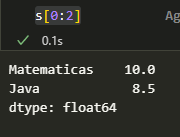
### Acceso a los elementos de una serie

El acceso a los elementos de un objeto del tipo Series puede ser a través de posiciones o través de índices (nombres).

**Acceso por posición**

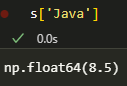
Se realiza de forma similar a como se accede a los elementos de un array.

* **s[i] :** Devuelve el elemento que ocupa la posición i+1 en la serie s.
* **s[posiciones]:** Devuelve otra serie con los elementos que ocupan las posiciones de la lista posiciones.

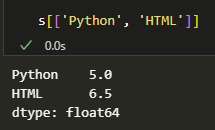


**Acceso por índice**

* **s[nombre] :** Devuelve el elemento con el nombre del índice.



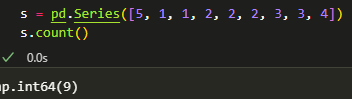
* **s[nombres] :** Devuelve otra serie con los elementos correspondientes a los nombres indicadas en la lista nombres en el índice.



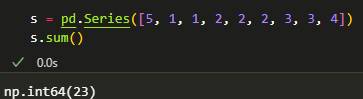
### Resumen descriptivo de una serie

Las siguientes funciones permiten resumir varios aspectos de una serie:

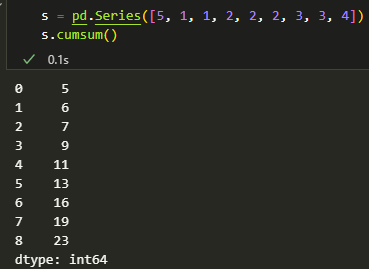
* s.count() : Devuelve el número de elementos que no son nulos ni NaN en la serie s.



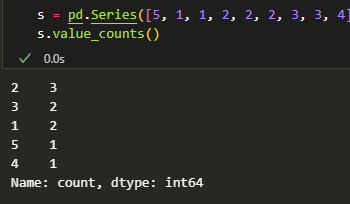
* s.sum() : Devuelve la suma de los datos de la serie s cuando los datos son de un tipo numérico, o la concatenación de ellos cuando son del tipo cadena str.



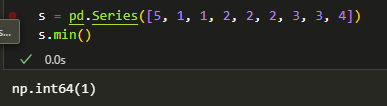
* s.cumsum() : Devuelve una serie con la suma acumulada de los datos de la serie s cuando los datos son de un tipo numérico.



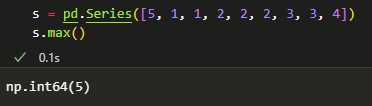
* s.value\_counts() : Devuelve una serie con la frecuencia (número de repeticiones) de cada valor de la serie s.



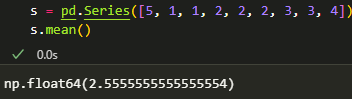
* s.min() : Devuelve el menor de los datos de la serie s.



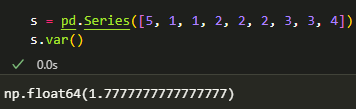
* s.max() : Devuelve el mayor de los datos de la serie s.



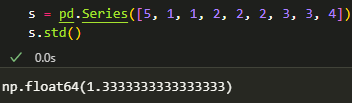
* s.mean() : Devuelve la media de los datos de la serie s cuando los datos son de un tipo numérico.



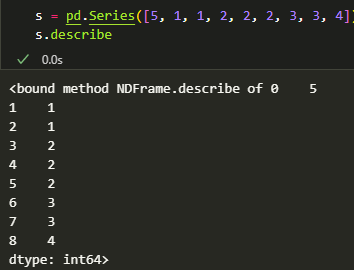
* s.var() : Devuelve la varianza de los datos de la serie s cuando los datos son de un tipo numérico.



* s.std() : Devuelve la desviación típica de los datos de la serie s cuando los datos son de un tipo numérico.

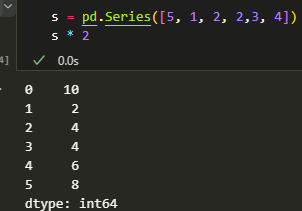


* s.describe(): Devuelve una serie con un resumen descriptivo que incluye el número de datos, su suma, el mínimo, el máximo, la media, la desviación típica y los cuartiles.



**Aplicar operaciones a una serie**

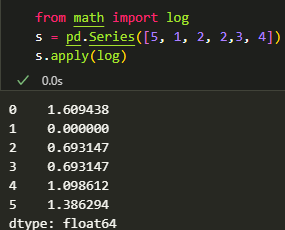
Los operadores binarios **(+, \*, /, etc.)** pueden utilizarse con una serie, y devuelven otra serie con el resultado de aplicar la operación a cada elemento de la serie.



### Aplicar funciones a una serie

También es posible aplicar una función a cada elemento de la serie mediante el siguiente método:

* s.apply(f) : Devuelve una serie con el resultado de aplicar la función f a cada uno de los elementos de la serie s.



*dtype: float64*

*>>> s = pd.Series(['a', 'b', 'c'])*

*>>> s.apply(str.upper)*

*0 A*

*1 B*

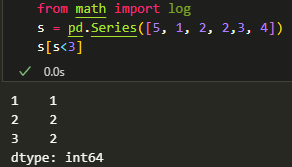
*2 C*

*dtype: object*

### Filtrar una serie

Para filtrar una serie y quedarse con los valores que cumplen una determinada condición se utiliza el siguiente método:

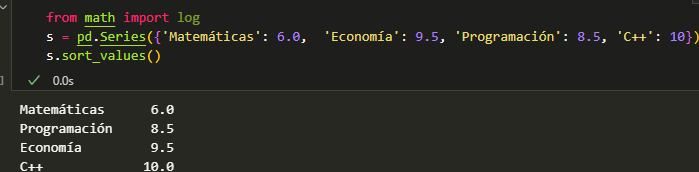
* s[condición] : Devuelve una serie con los elementos de la serie s que se corresponden con el valor True de la lista booleana condición. condición debe ser una lista de valores booleanos de la misma longitud que la serie.



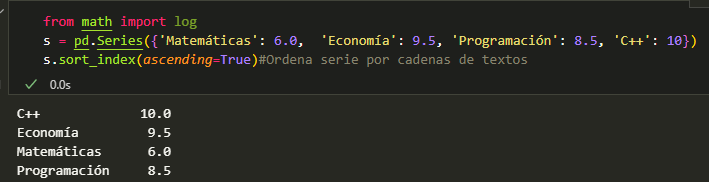
### Ordenar una serie

Para ordenar una serie se utilizan los siguientes métodos:

* s.sort\_values(ascending=booleano) : Devuelve la serie que resulta de ordenar los valores la serie s. Si argumento del parámetro ascending es True el orden es creciente y si es False decreciente.



* df.sort\_index(ascending=booleano) : Devuelve la serie que resulta de ordenar el índice de la serie s. Si el argumento del parámetro ascending es True el orden es creciente y si es False decreciente.



### Eliminar los dados desconocidos en una serie

Los datos desconocidos representan en Pandas por NaN y los nulos por None. Tanto unos como otros suelen ser un problema a la hora de realizar algunos análisis de datos, por lo que es habitual eliminarlos. Para eliminarlos de una serie se utiliza el siguiente método:

* s.dropna() : Elimina los datos desconocidos o nulos de la serie s.

**>>> import** pandas **as** pd

**>>> import** numpy **as** np

**>>>** s = pd.Series(['a', 'b', None, 'c', np.NaN, 'd'])

**>>>** s

0 a

1 b

2 None

3 c

4 NaN

5 d

dtype: object

**>>>** s.dropna()

0 a

1 b

3 c

5 d

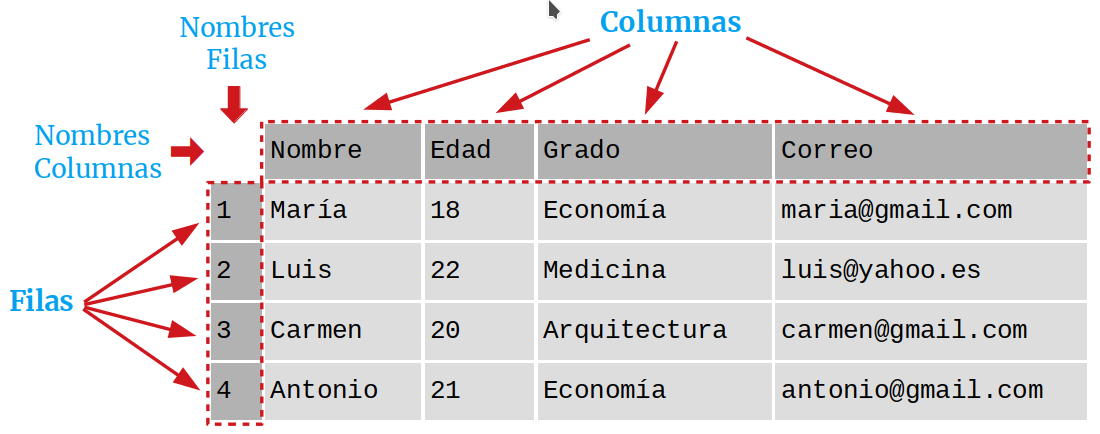
dtype: object

### La clase de objetos DataFrame

Un objeto del tipo DataFrame define un conjunto de datos estructurado en forma de tabla donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contender datos de distintos tipos.

Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas.

**Ejemplo.** El siguiente DataFrame contiene información sobre los alumnos de un curso. Cada fila corresponde a un alumno y cada columna a una variable.



### Creación de un DataFrame

**Creación de un DataFrame a partir de un diccionario de listas**

Para crear un DataFrame a partir de un diccionario cuyas claves son los nombres de las columnas y los valores son listas con los datos de las columnas se utiliza el método:

*DataFrame(data=diccionario, index=filas, columns=columnas, dtype=tipos) : Devuelve un objeto del tipo DataFrame cuyas columnas son las listas contenidas en los valores del diccionario diccionario, los nombres de filas indicados en la lista filas, los nombres de columnas indicados en la lista columnas y los tipos indicados en la lista tipos. La lista filas debe tener el mismo tamaño que las listas del diccionario, mientras que las listas columnas y tipos deben tener el mismo tamaño que el diccionario. Si no se pasa la lista de filas se utilizan como nombres los enteros empezando en 0. Si no se pasa la lista de columnas se utilizan como nombres las claves del diccionario. Si no se pasa la lista de tipos, se infiere.*

 Los valores asociados a las claves del diccionario deben ser listas del mismo tamaño.

#Crear un dataframe con un diccionario de informacion.

datos = {

    'id': [1, 2, 3, 4, 5, 6],

    'nombre': ['Juan', 'Ana', 'Camila', 'Salome', 'Sharis Góngora', 'Melissa'],

    'edad': [25, 30, 22, 23, 34, 21],

    'estado': ['Tabasco', 'Tabasco', 'Chiapas', 'Veracruz', 'Tabasco', 'Veracruz'],

    'calificación': [6, 9.5, 8.9, 6, 9, 10],

    'salario': [10000, 20000, 15000, 12000, 30000, 18000],

    'impuesto': [0.16, 0.16, 0.16, 0.16, 0.16, 0.16], # 16% de IVA

    'peso':[70, 60, 55, 58, 65, 50],  # en kg

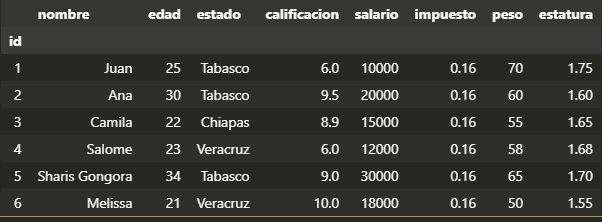
     'estatura':[1.75, 1.60, 1.65, 1.68, 1.70, 1.55]  # en metros

}

#Convertirlo a un dataframe

tabla\_datos = pd.DataFrame(datos).set\_index('id')

tabla\_datos

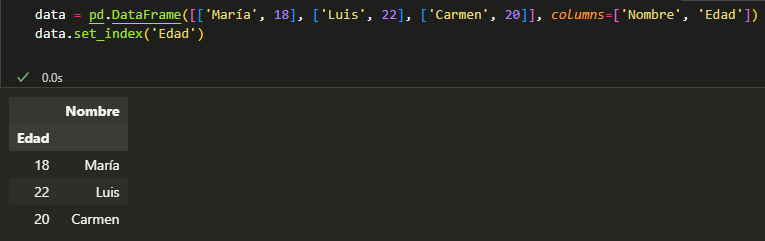


### Creación de un DataFrame a partir de una lista de listas

Para crear un DataFrame a partir de una lista de listas con los datos de las columnas se utiliza el siguiente método:

* DataFrame(data=listas, index=filas, columns=columnas, dtype=tipos) : Devuelve un objeto del tipo DataFrame cuyas columnas son los valores de las listas de la lista listas, los nombres de filas indicados en la lista filas, los nombres de columnas indicados en la lista columnas y los tipos indicados en la lista tipos. La lista filas, debe tener el mismo tamaño que la lista listas mientras que las listas columnas y tipos deben tener el mismo tamaño que las listas anidadas en listas. Si no se pasa la lista de filas o de columnas se utilizan enteros empezando en 0. Si no se pasa la lista de tipos, se infiere.

Si las listas anidadas en listas no tienen el mismo tamaño, las listas menores se rellenan con valores NaN.



### Creación de un DataFrame a partir de una lista de diccionarios

Para crear un DataFrame a partir de una lista de diccionarios con los datos de las filas, se utiliza el siguiente método:

* DataFrame(data=diccionarios, index=filas, columns=columnas, dtype=tipos) : Devuelve un objeto del tipo DataFrame cuyas filas contienen los valores de los diccionarios de la lista diccionarios, los nombres de filas indicados en la lista filas, los nombres de columnas indicados en la lista columnas y los tipos indicados en la lista tipos. La lista filas debe tener el mismo tamaño que la lista lista. Si no se pasa la lista de filas se utilizan enteros empezando en 0. Si no se pasa la lista de columnas se utilizan las claves de los diccionarios. Si no se pasa la lista de tipos, se infiere.

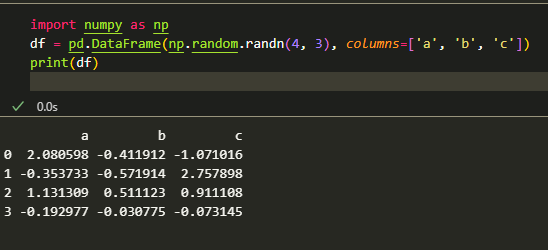
 Si los diccionarios no tienen las mismas claves, las claves que no aparecen en el diccionario se rellenan con valores NaN.



### Creación de un DataFrame a partir de un array

Para crear un DataFrame a partir de un array de NumPy se utiliza el siguiente método:

* DataFrame(data=array, index=filas, columns=columnas, dtype=tipo) : Devuelve un objeto del tipo DataFrame cuyas filas y columnas son las del array array, los nombres de filas indicados en la lista filas, los nombres de columnas indicados en la lista columnas y el tipo indicado en tipo. La lista filas debe tener el mismo tamaño que el número de filas del array y la lista columnas el mismo tamaño que el número de columnas del array. Si no se pasa la lista de filas se utilizan enteros empezando en 0. Si no se pasa la lista de columnas se utilizan las claves de los diccionarios. Si no se pasa la lista de tipos, se infiere.



### Creación de un DataFrame a partir de un fichero CSV o Excel

Dependiendo del tipo de fichero, existen distintas funciones para importar un DataFrame desde un fichero.

* read\_csv(fichero.csv, sep=separador, header=n, index\_col=m, na\_values=no-validos, decimal=separador-decimal) : Devuelve un objeto del tipo DataFrame con los datos del fichero CSV fichero.csv usando como separador de los datos la cadena separadora. Como nombres de columnas se utiliza los valores de la fila n y como nombres de filas los valores de la columna m. Si no se indica m se utilizan como nombres de filas los enteros empezando en 0. Los valores incluidos en la lista no-validos se convierten en NaN. Para los datos numéricos se utiliza como separador de decimales el carácter indicado en separador-decimal.
* read\_excel(fichero.xlsx, sheet\_name=hoja, header=n, index\_col=m, na\_values=no-validos, decimal=separador-decimal) : Devuelve un objeto del tipo DataFrame con los datos de la hoja de cálculo hoja del fichero Excel fichero.xlsx. Como nombres de columnas se utiliza los valores de la fila n y como nombres de filas los valores de la columna m. Si no se indica m se utilizan como nombres de filas los enteros empezando en 0. Los valores incluidos en la lista no-validos se convierten en NaN. Para los datos numéricos se utiliza como separador de decimales el carácter indicado en separador-decimal.



### Exportación de ficheros

También existen funciones para exportar un DataFrame a un fichero con diferentes formatos.

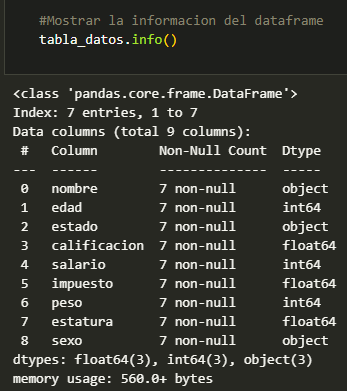
df.to\_csv(fichero.csv, sep=separador, columns=booleano, index=booleano) : Exporta el DataFrame df al fichero fichero.csv en formato CSV usando como separador de los datos la cadena separadora. Si se pasa True al parámetro columns se exporta también la fila con los nombres de columnas y si se pasa True al parámetro index se exporta también la columna con los nombres de las filas.

df.to\_excel(fichero.xlsx, sheet\_name = hoja, columns=booleano, index=booleano) : Exporta el DataFrame df a la hoja de cálculo hoja del fichero fichero.xlsx en formato Excel. Si se pasa True al parámetro columns se exporta también la fila con los nombres de columnas y si se pasa True al parámetro index se exporta también la columna con los nombres de las filas.

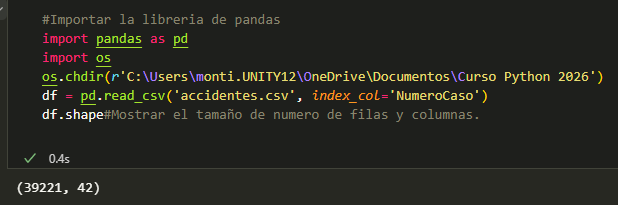
### Atributos de un DataFrame

Existen varias propiedades o métodos para ver las características de un DataFrame.

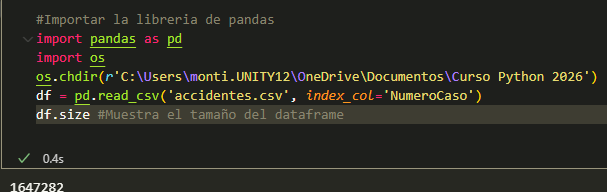
* df.info() : Devuelve información (número de filas, número de columnas, índices, tipo de las columnas y memoria usado) sobre el DataFrame df.



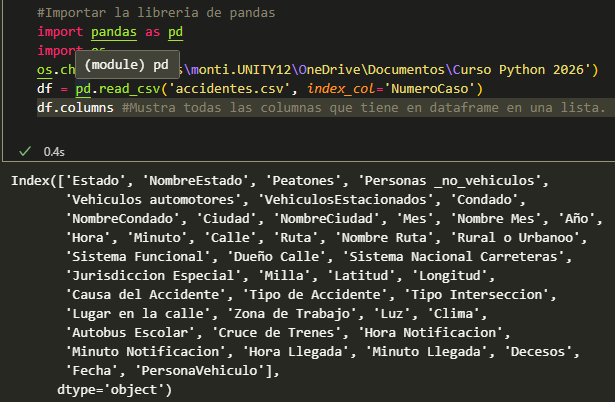
* df.shape : Devuelve una tupla con el número de filas y columnas del DataFrame df.



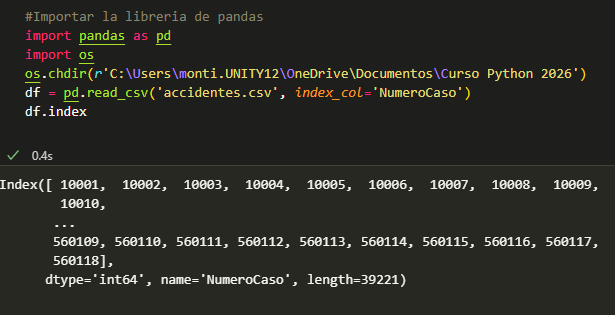
* df.size : Devuelve el número de elementos del DataFrame.



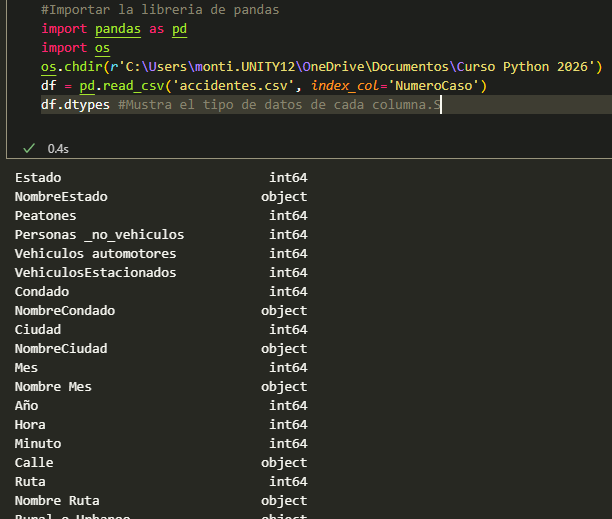
* df.columns : Devuelve una lista con los nombres de las columnas del DataFrame df.



* **df.index** : Devuelve una lista con los nombres de las filas del DataFrame df.



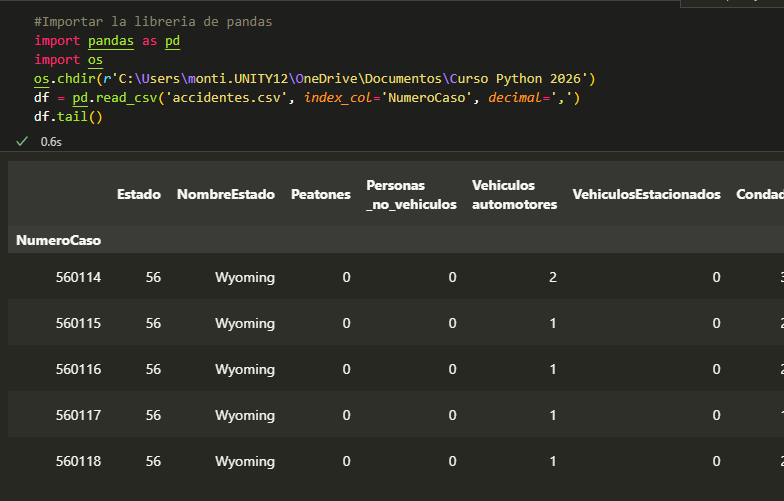
* df.dtypes : Devuelve una serie con los tipos de datos de las columnas del DataFrame df.



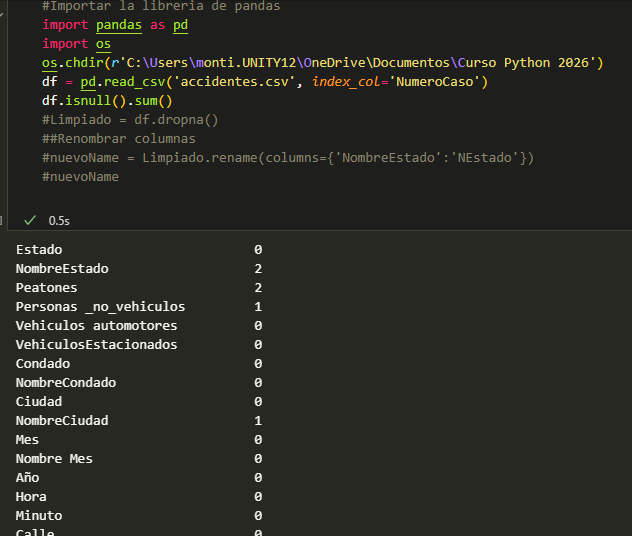
* df.head(n) : Devuelve las n primeras filas del DataFrame df.



* df.tail(n) : Devuelve las n últimas filas del DataFrame df.



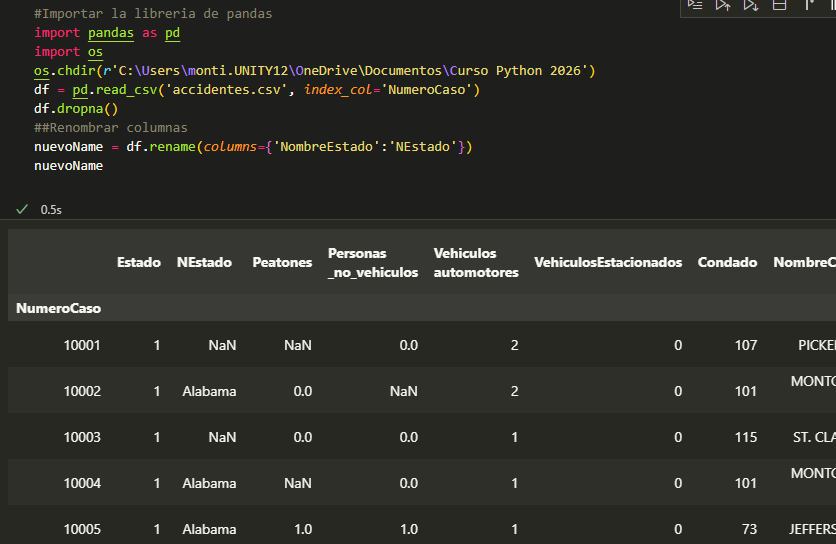
**Df.isnull().sum()** #Cuentas cuantos valores nulos hay en cada columna



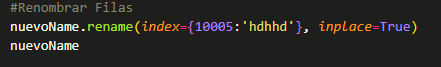
### Renombrar los nombres de las filas y columnas

Para cambiar el nombre de las filas y las columnas de un DataFrame se utiliza el siguiente método:

df.rename(columns=columnas, index=filas): Devuelve el DataFrame que resulta de renombrar las columnas indicadas en las claves del diccionario columnas con sus valores y las filas indicadas en las claves del diccionario filas con sus valores en el DataFrame df.



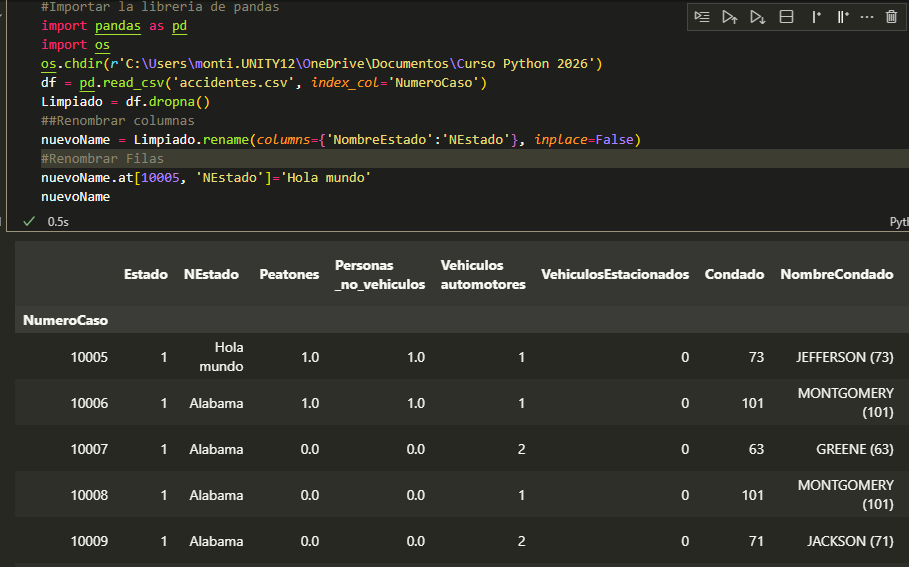
#Renombrar informacion de cualquier fila.



#Renombrar filas por el índice y nombre de columna



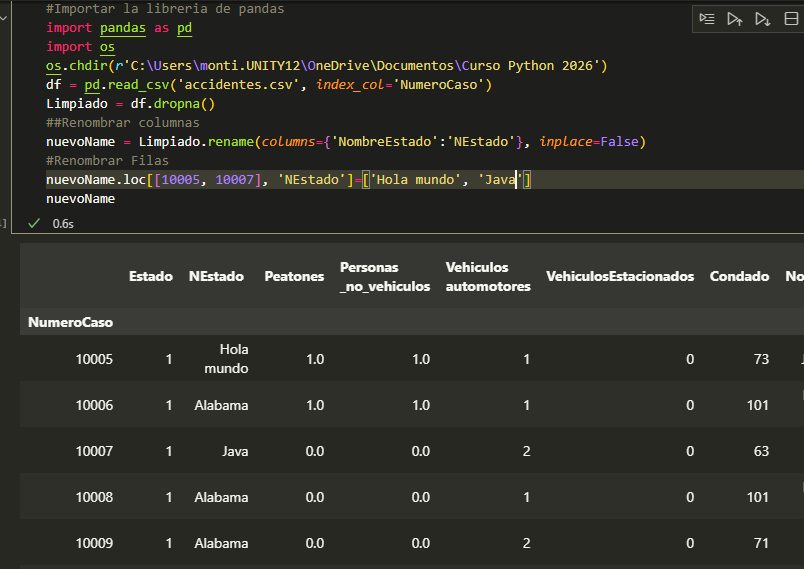
Forma 2 usando **df.at[10005, ‘NEstado’]=’Hola mundo’**



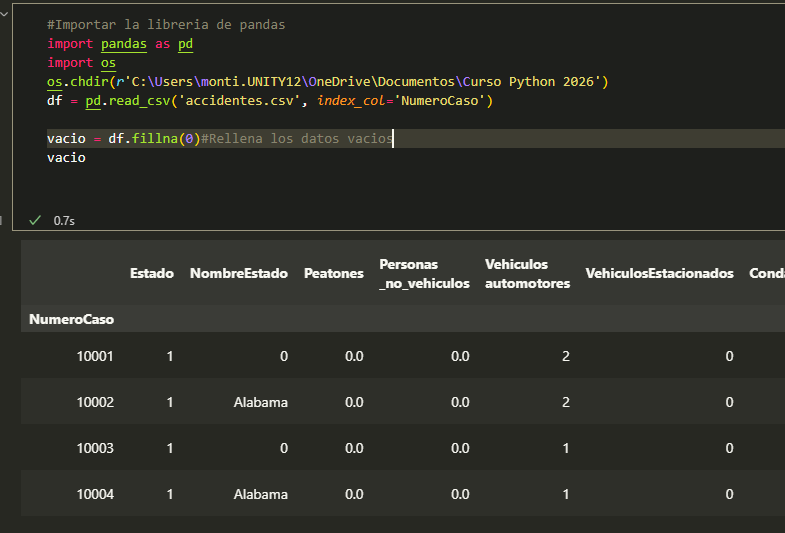
**Renombrar la Filas de una sola columna**

* nuevoName.loc[[10005, 10007], 'NEstado']=['Hola mundo', 'Java']

nuevoName



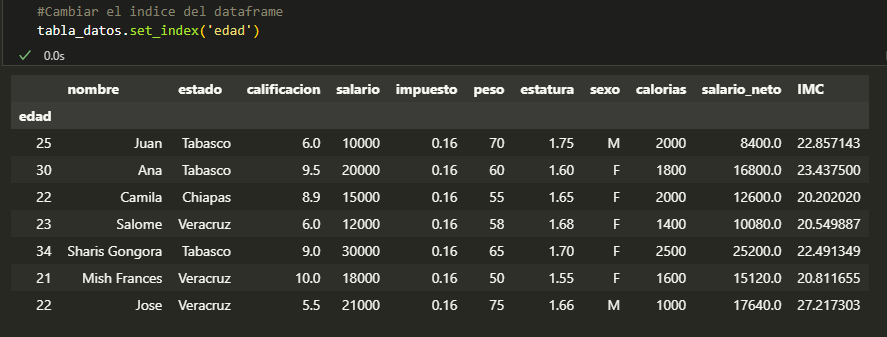
**Rellenar filas de datos vacío con valores por defecto.**



### Cambiar el índice de un DataFrame

Aunque el índice de un DataFrame suele fijarse en la creación del mismo, en ocasiones puede ser necesario cambiar el índice una vez creado el DataFrame. Para ello se utiliza el siguiente método:

* df.set\_index(keys = columnas, verify\_integrity = bool): Devuelve el DataFrame que resulta de eliminar las columnas de la lista columnas y convertirlas en el nuevo índice. El parámetro verify\_integrity recibe un booleano (False por defecto) y realiza una comprobación para evitar duplicados en la clave cuando recibe True.



### Reindexar un DataFrame

Para reordenar los índices de las filas y las columnas de un DataFrame, así como añadir o eliminar índices, se utiliza el siguiente método:

* df.reindex(index=filas, columns=columnas, fill\_value=relleno) : Devuelve el DataFrame que resulta de tomar del DataFrame df las filas con nombres en la lista filas y las columnas con nombres en la lista columnas. Si alguno de los nombres indicados en filas o columnas no existía en el DataFrame df, se crean filan o columnas nuevas rellenas con el valor relleno.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

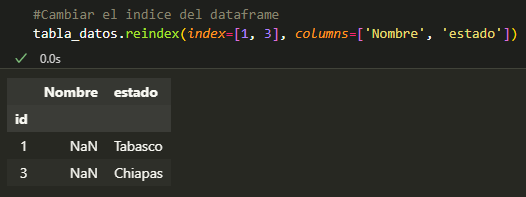
**>>>** print(df.reindex(index=[4, 3, 1], columns=['nombre', 'tensión', 'colesterol']))

nombre tensión colesterol

4 Marisa López Collado NaN 148.0

3 Carmen López Pinzón NaN 200.0

1 Rosa Díaz Díaz NaN 232.0



### Acceso a los elementos de un DataFrame

El acceso a los datos de un DataFrame se puede hacer a través de posiciones o través de los nombres de las filas y columnas.

**Accesos mediante posiciones**

df.iloc[i, j] : Devuelve el elemento que se encuentra en la fila i y la columna j del DataFrame df. Pueden indicarse secuencias de índices para obtener partes del **DataFrame.**

* df.iloc[filas, columnas] : Devuelve un DataFrame con los elementos de las filas de la lista filas y de las columnas de la lista columnas.
* df.iloc[i] : Devuelve una serie con los elementos de la fila i del DataFrame df.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** print(df.iloc[1, 3])

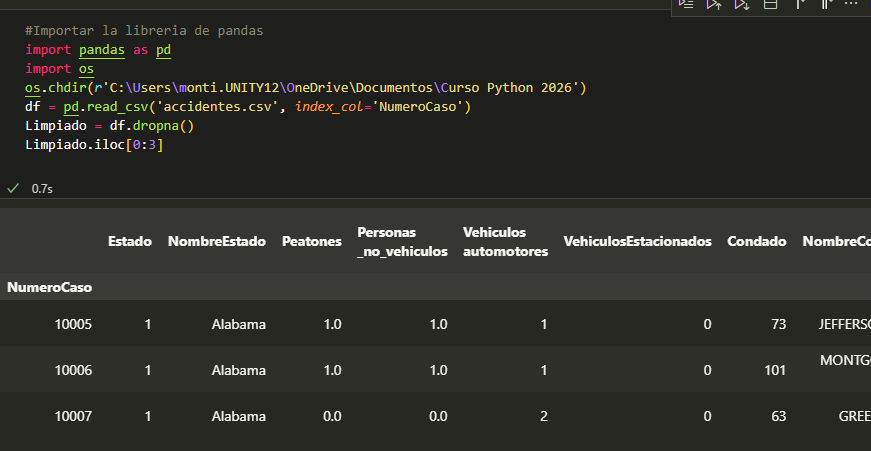
65

**>>>** print(df.iloc[1, :2])

nombre Rosa Díaz Díaz

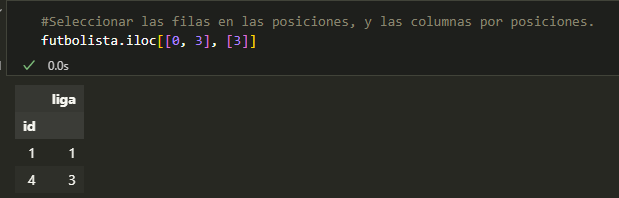
edad 32





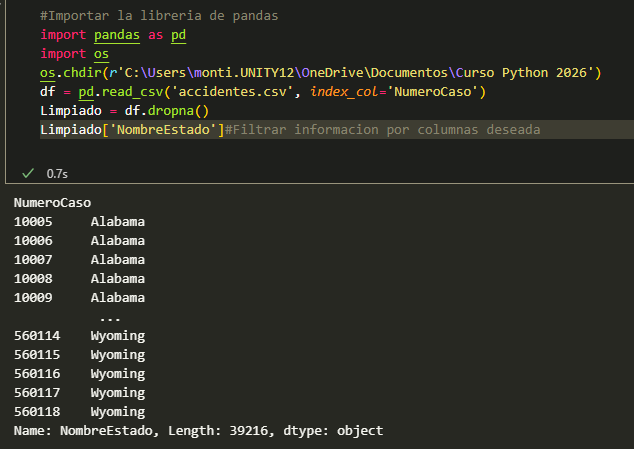
Interfaz de usuario gráfica, Texto, Sitio web

El contenido generado por IA puede ser incorrecto.

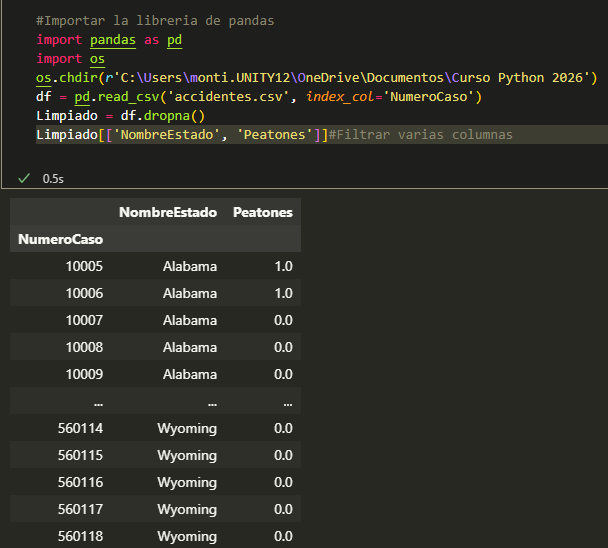


Acceso a los elementos mediante nombres

* df.loc[fila, columna] : Devuelve el elemento que se encuentra en la fila con nombre fila y la columna de con nombre columna del DataFrame df.
* df.loc[filas, columnas] : Devuelve un DataFrame con los elementos que se encuentra en las filas con los nombres de la lista filas y las columnas con los nombres de la lista columnas del DataFrame df.
* df[columna] : Devuelve una serie con los elementos de la columna de nombre columna del DataFrame df.



* df.columna : Devuelve una serie con los elementos de la columna de nombre columna del DataFrame df. Es similar al método anterior pero solo funciona cuando el nombre de la columna no tiene espacios en blanco.



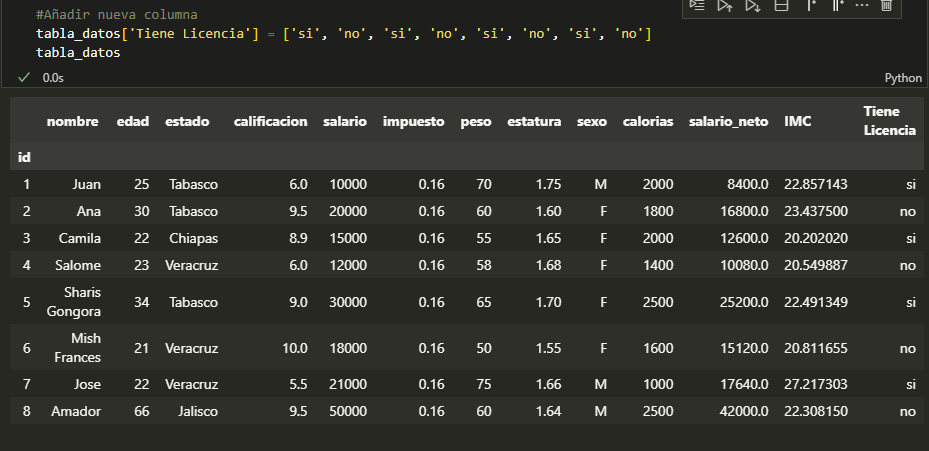
### Operaciones con las columnas de un DataFrame

Añadir columnas a un DataFrame

El procedimiento para añadir una nueva columna a un DataFrame es similar al de añadir un nuevo par a un diccionario, pero pasando los valores de la columna en una lista o serie.

* d[nombre] = lista: Añade al DataFrame df una nueva columna con el nombre nombre y los valores de la lista lista. La lista debe tener el mismo tamaño que el número de filas de df.
* d[nombre] = serie: Añade al DataFrame df una nueva columna con el nombre nombre y los valores de la serie serie. Si el tamaño de la serie es menor que el número de filas de df se rellena con valores NaN mientras que si es mayor se recorta.

**Nueva Columna.**

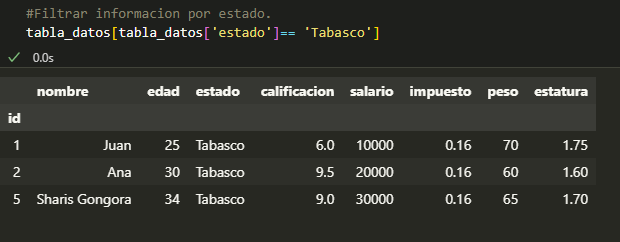


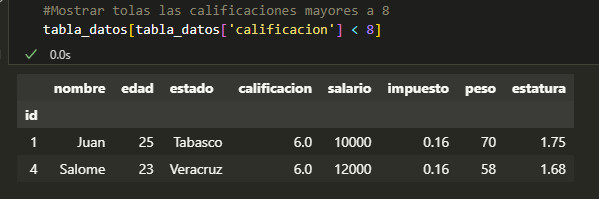
Nueva fila.

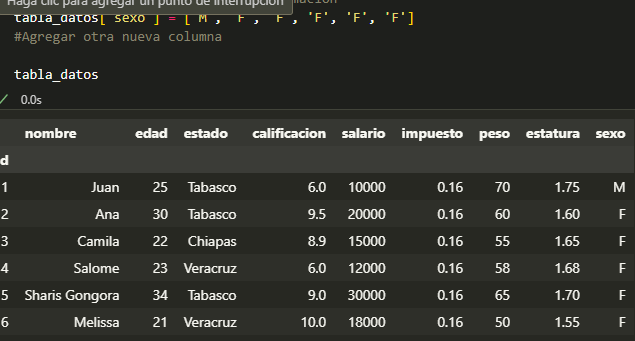


### Operaciones sobre columnas

Puesto que los datos de una misma columna de un DataFrame son del mismo tipo, es fácil aplicar la misma operación a todos los elementos de la columna.







Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

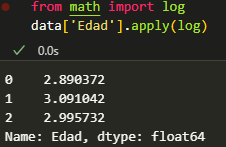
Una pantalla de un video juego

El contenido generado por IA puede ser incorrecto.

### Aplicar funciones a columnas

Para aplicar funciones a todos los elementos de una columna se utiliza el siguiente método:

* df[columna].apply(f) : Devuelve una serie con los valores que resulta de aplicar la función f a los elementos de la columna con nombre columna del DataFrame df.



**>>> import** pandas **as** pd

**>>> from** math **import** log

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** print(df['altura'].apply(log))

0 0.582216

1 0.548121

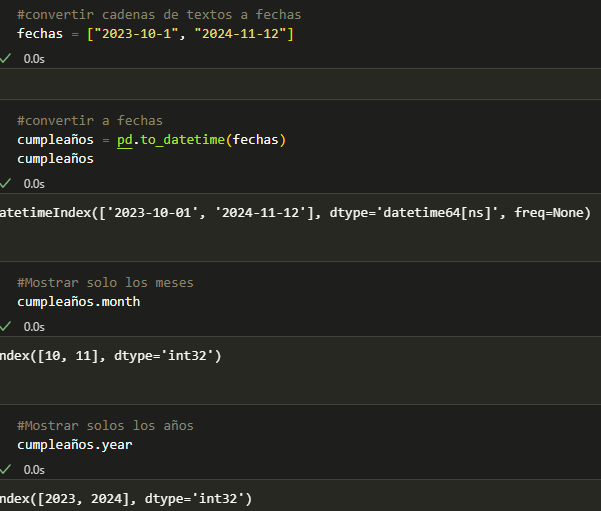
2 0.593327

...

### Convertir una columna al tipo datetime

A menudo una columna contiene cadenas que representan fechas. Para convertir estas cadenas al tipo datetime se utiliza el siguiente método:

* to\_datetime(columna, formato): Devuelve la serie que resulta de convertir las cadenas de la columna con el nombre columna en fechas del tipo datetime con el formado especificado en formato.



**>>> import** pandas **as** pd

**>>>** df = pd.DataFrame({'Name': ['María', 'Carlos', 'Carmen'], 'Nacimiento':['05-03-2000', '20-05-2001', '10-12-1999']})

**>>>** print(pd.to\_datetime(df.Nacimiento, format = '%d-%m-%Y'))

0 2000-03-05

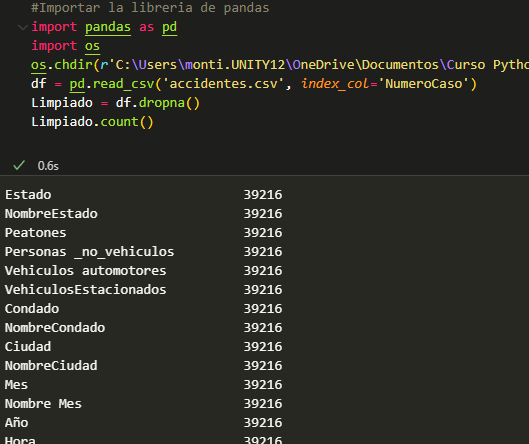
1 2001-05-20

2 1999-12-10

Name: Nacimiento, dtype: datetime64[ns]

### Resumen descriptivo de un DataFrame

Al igual que para las series, los siguientes métodos permiten resumir la información de un DataFrame por columnas:

* df.count() : Devuelve una serie con el número de elementos que no son nulos ni NaN en cada columna del DataFrame df.
* 
* df.sum() : Devuelve una serie con la suma de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico, o la concatenación de ellos cuando son del tipo cadena str.
* df.cumsum() : Devuelve un DataFrame con la suma acumulada de los datos de las columnas del DataFrame df cuando los datos son de un tipo numérico.
* df.min() : Devuelve una serie con los menores de los datos de las columnas del DataFrame df.
* df.max() : Devuelve una serie con los mayores de los datos de las columnas del DataFrame df.
* df.mean() : Devuelve una serie con las medias de los datos de las columnas numéricas del DataFrame df.
* df.var() : Devuelve una serie con las varianzas de los datos de las columnas numéricas del DataFrame df.
* df.std() : Devuelve una serie con las desviaciones típicas de los datos de las columnas numéricas del DataFrame df.
* df.cov() : Devuelve un DataFrame con las covarianzas de los datos de las columnas numéricas del DataFrame df.
* df.corr() : Devuelve un DataFrame con los coeficientes de correlación de Pearson de los datos de las columnas numéricas del DataFrame df.
* df.describe(include = tipo) : Devuelve un DataFrame con un resumen estadístico de las columnas del DataFrame df del tipo tipo. Para los datos numéricos (number) se calcula la media, la desviación típica, el mínimo, el máximo y los cuartiles. Para los datos no numéricos (object) se calcula el número de valores, el número de valores distintos, la moda y su frecuencia. Si no se indica el tipo solo se consideran las columnas numéricas.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

>>>df.edad.count() *# Tamaño muestral*

14

**>>>** print(df.edad.mean()) *# Media*

38.214285714285715

**>>>** print(df.edad.var()) *# Varianza*

244.02747252747255

**>>>** print(df.edad.std()) *# Desviación típica*

15.62137870123737

**>>>** df.cov() *# Matriz de covarianzas*

edad peso altura colesterol

edad 244.027473 -69.891026 -0.326593 279.717949

peso -69.891026 260.076923 1.764615 -2.424242

altura -0.326593 1.764615 0.013229 0.563269

colesterol 279.717949 -2.424242 0.563269 1587.858974

**>>>** df.corr() *# Matriz de correlación*

edad peso altura colesterol

edad 1.000000 -0.276185 -0.181774 0.452391

peso -0.276185 1.000000 0.918984 -0.003621

altura -0.181774 0.918984 1.000000 0.122694

colesterol 0.452391 -0.003621 0.122694 1.000000

**>>>** print(df.describe()) *# Resumen descriptivo*

edad peso altura colesterol

count 14.000000 13.000000 14.000000 13.000000

mean 38.214286 70.923077 1.768571 220.230769

std 15.621379 16.126901 0.115016 39.847948

min 18.000000 51.000000 1.580000 148.000000

25% 24.750000 61.000000 1.705000 194.000000

50% 35.000000 65.000000 1.755000 210.000000

75% 49.750000 78.000000 1.840000 249.000000

max 68.000000 109.000000 1.980000 280.000000

**>>>** print(df.describe(include='object'))

nombre sexo

count 14 14

unique 14 2

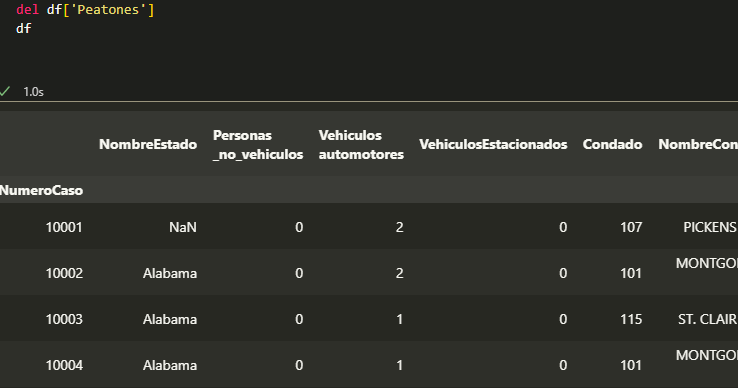
top Antonio Fernández Ocaña H

freq 1 8

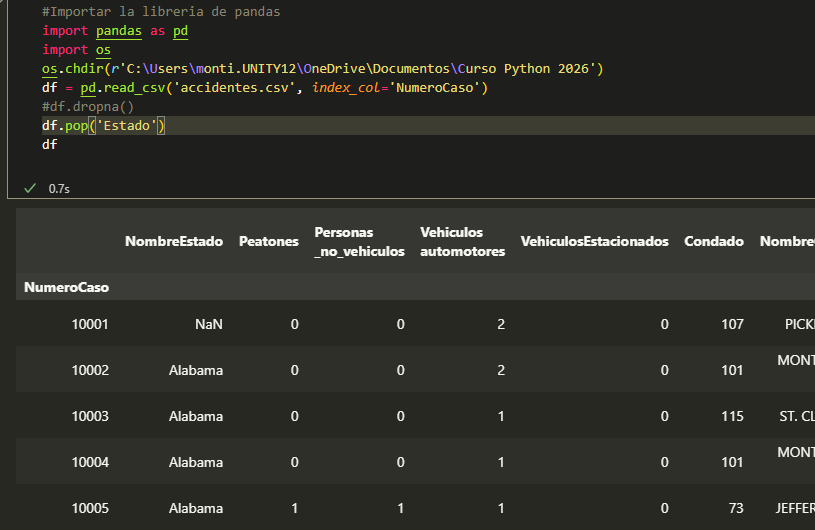
### Eliminar columnas de un DataFrame

Para eliminar columnas de un DataFrame se utilizan los siguientes métodos:

* del d[nombre] : Elimina la columna con nombre nombre del DataFrame df.

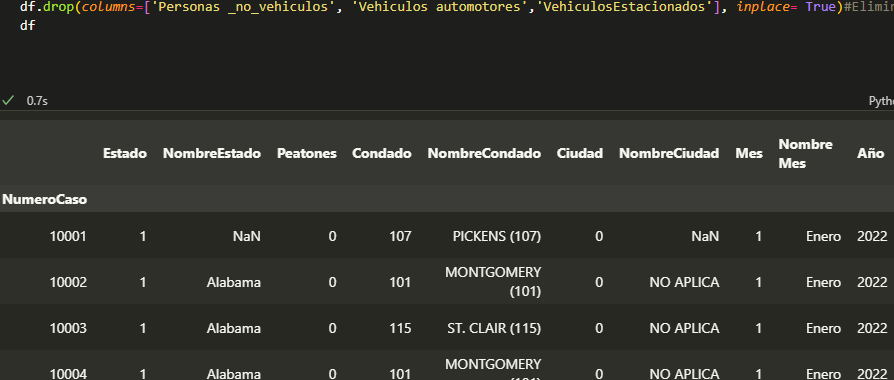


* df.pop(nombre) : Elimina la columna con nombre nombre del DataFrame df y la devuelve como una serie.



Permite eliminar varias columnas al mismo tiempo

*df.drop(columns=['Personas \_no\_vehiculos', 'Vehiculos automotores','VehiculosEstacionados'], inplace= True)*



### Operaciones con las filas de un DataFrame

Añadir una fila a un DataFrame

Para añadir una fila a un DataFrame se utiliza el siguiente método:

* df.append(serie, ignore\_index=True) : Devuelve el DataFrame que resulta de añadir una fila al DataFrame df con los valores de la serie serie. Los nombres del índice de la serie deben corresponderse con los nombres de las columnas de df. Si no se pasa el parámetro ignore\_index entonces debe pasarse el parámetro name a la serie, donde su argumento será el nombre de la nueva fila.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** df = df.append(pd.Series(['Carlos Rivas', 28, 'H', 89.0, 1.78, 245.0], index=['nombre','edad','sexo','peso','altura','colesterol']), ignore\_index=True)

**>>>** print(df.tail())

nombre edad sexo peso altura colesterol

10 Macarena Álvarez Luna 53 M 55.0 1.62 262.0

11 José María de la Guía Sanz 58 H 78.0 1.87 198.0

12 Miguel Angel Cuadrado Gutiérrez 27 H 109.0 1.98 210.0

13 Carolina Rubio Moreno 20 M 61.0 1.77 194.0

14 Carlos Rivas 28 H 89.0 1.78 245.0

### Eliminar filas de un DataFrame

Para eliminar filas de un DataFrame se utilizan el siguiente método:

* df.drop(filas) : Devuelve el DataFrame que resulta de eliminar las filas con los nombres indicados en la lista filas del DataFrame df.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** print(df.drop([1, 3]))

nombre edad sexo peso altura colesterol

0 José Luis Martínez Izquierdo 18 H 85.0 1.79 182.0

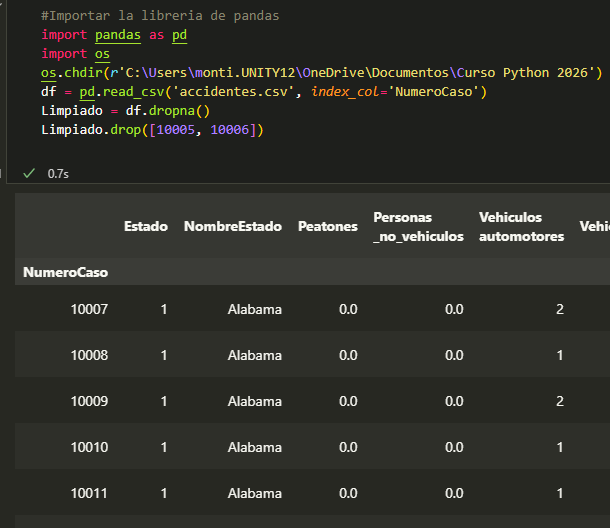
2 Javier García Sánchez 24 H NaN 1.81 191.0

4 Marisa López Collado 46 M 51.0 1.58 148.0

...



Elimina filas por listas.



### Filtrar las filas de un DataFrame

Una operación bastante común con un DataFrame es obtener las filas que cumplen una determinada condición.

* df[condicion] : Devuelve un DataFrame con las filas del DataFrame df que se corresponden con el valor True de la lista booleana condicion. condicion debe ser una lista de valores booleanos de la misma longitud que el número de filas del DataFrame.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** print(df[(df['sexo']=='H') & (df['colesterol'] > 260)])

nombre edad sexo peso altura colesterol

6 Antonio Fernández Ocaña 51 H 62.0 1.72 276.0

9 Santiago Reillo Manzano 46 H 75.0 1.85 280.0

Una pantalla de un video juego

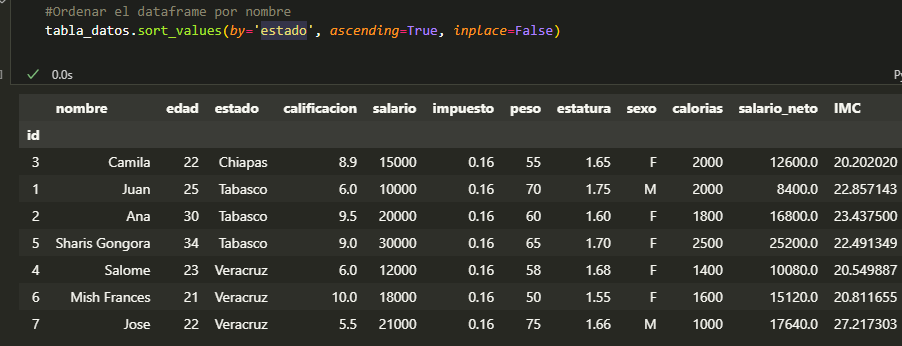
El contenido generado por IA puede ser incorrecto.

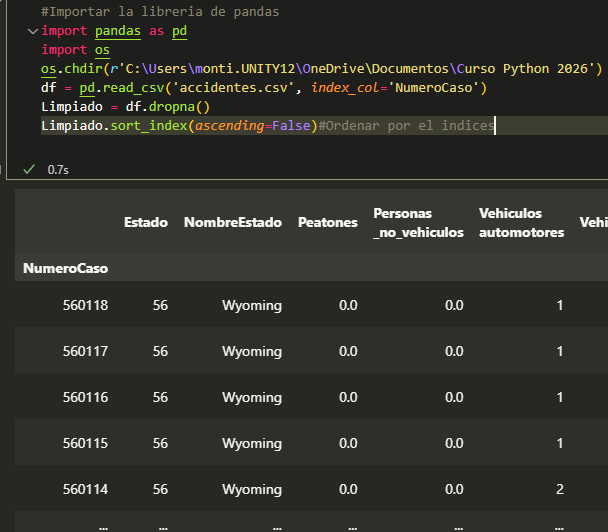
### Ordenar un DataFrame

Para ordenar un DataFrame de acuerdo con los valores de una determinada columna se utilizan los siguientes métodos:

* df.sort\_values(columna, ascending=booleano) : Devuelve el DataFrame que resulta de ordenar las filas del DataFrame df según los valores de la columna con nombre columna. Si argumento del parámetro ascending es True el orden es creciente y si es False decreciente.
* df.sort\_index(ascending=booleano) : Devuelve el DataFrame que resulta de ordenar las filas del DataFrame df según los nombres de las filas. Si el argumento del parámetro ascending es True el orden es creciente y si es False decreciente.

**Ordenar informacion por estado.**

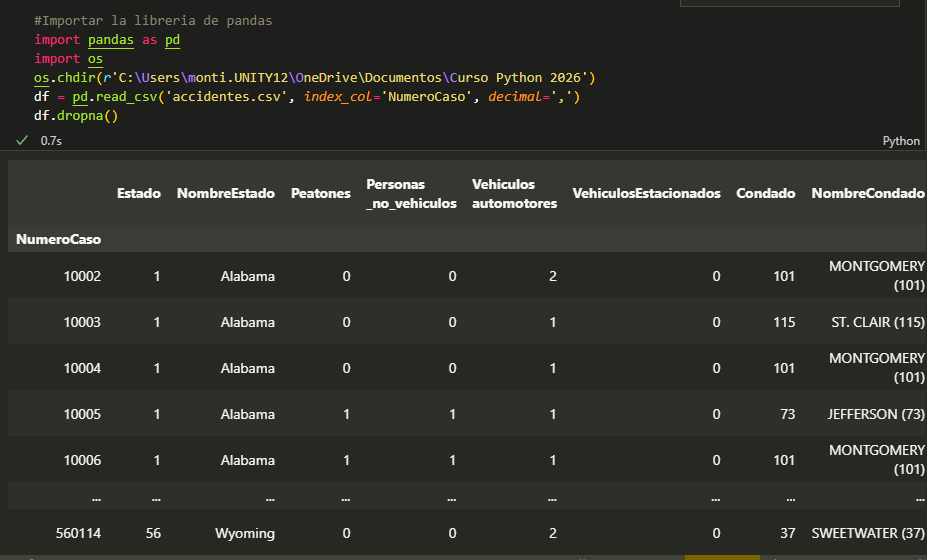




### Eliminar las filas con datos desconocidos en un DataFrame

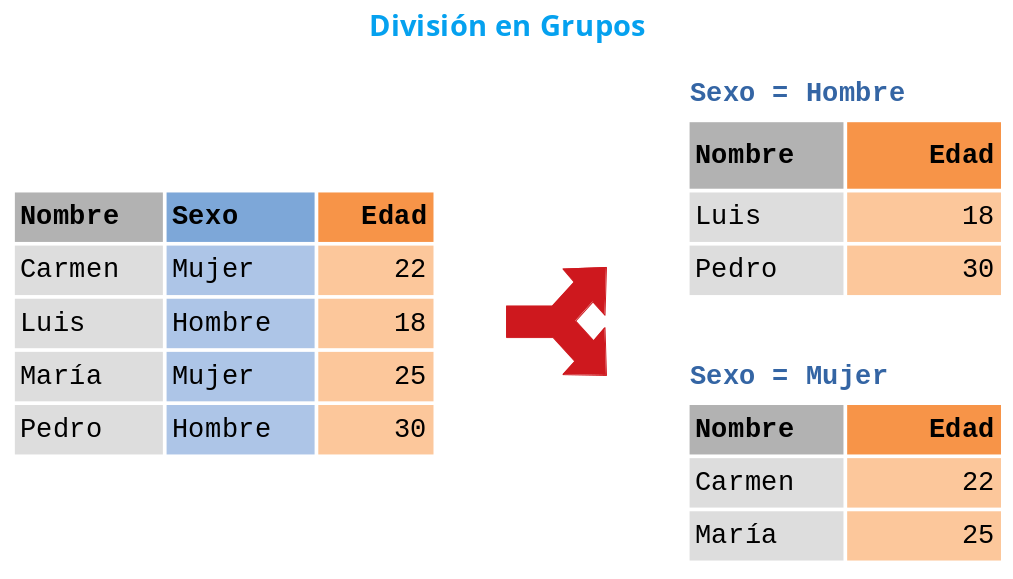
Para eliminar las filas de un DataFrame que contienen datos desconocidos NaN o nulos None se utiliza el siguiente método:

* s.dropna(subset=columnas) : Devuelve el DataFrame que resulta de eliminar las filas que contienen algún dato desconocido o nulo en las columnas de la lista columna del DataFrame df. Si no se pasa un argumento al parámetro subset se aplica a todas las columnas del DataFrame.



### Agrupación de un DataFrame

En muchas aplicaciones es útil agrupar los datos de un DataFrame de acuerdo con los valores de una o varias columnas (categorías), como por ejemplo el sexo o el país.



### Dividir un DataFrame en grupos

Para dividir un DataFrame en grupos se utiliza el siguiente método:

* df.groupby(columnas).groups : Devuelve un diccionario con cuyas claves son las tuplas que resultan de todas las combinaciones de los valores de las columnas con nombres en la lista columnas, y valores las listas de los nombres de las filas que contienen esos valores en las correspondientes columnas del DataFrame df.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

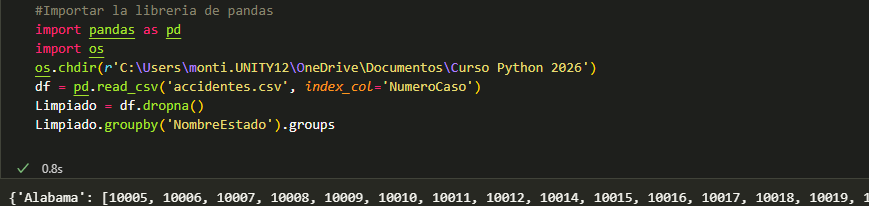
'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

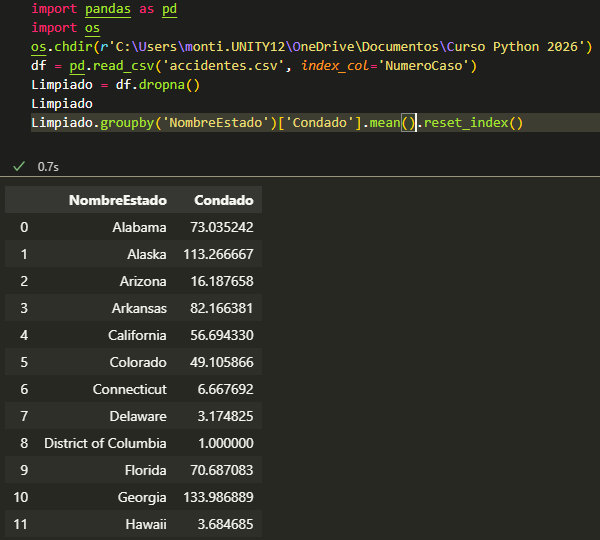
**>>>** print(df.groupby('sexo').groups)

{'H': Int64Index([0, 2, 5, 6, 8, 9, 11, 12], dtype='int64'), 'M': Int64Index([1, 3, 4, 7, 10, 13], dtype='int64')}

**>>>** print(df.groupby(['sexo','edad']).groups)

{('H', 18): Int64Index([0], dtype='int64'), ('H', 24): Int64Index([2], dtype='int64'), ('H', 27): Int64Index([12], dtype='int64'), ('H', 35): Int64Index([8], dtype='int64'), ('H', 46): Int64Index([9], dtype='int64'), ('H', 51): Int64Index([6], dtype='int64'), ('H', 58): Int64Index([11], dtype='int64'), ('H', 68): Int64Index([5], dtype='int64'), ('M', 20): Int64Index([13], dtype='int64'), ('M', 22): Int64Index([7], dtype='int64'), ('M', 32): Int64Index([1], dtype='int64'), ('M', 35): Int64Index([3], dtype='int64'), ('M', 46): Int64Index([4], dtype='int64'), ('M', 53): Int64Index([10], dtype='int64')}





Para obtener un grupo concreto se utiliza el siguiente método:

* df.groupby(columnas).get\_group(valores) : Devuelve un DataFrame con las filas del DataFrame df que cumplen que las columnas de la lista columnas presentan los valores de la tupla valores. La lista columnas y la tupla valores deben tener el mismo tamaño.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** print(df.groupby('sexo').get\_group('M'))

nombre edad sexo peso altura colesterol

1 Rosa Díaz Díaz 32 M 65.0 1.73 232.0

3 Carmen López Pinzón 35 M 65.0 1.70 200.0

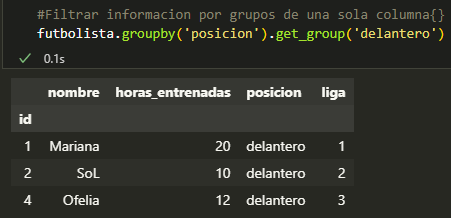
4 Marisa López Collado 46 M 51.0 1.58 148.0

7 Pilar Martín González 22 M 60.0 1.66 NaN

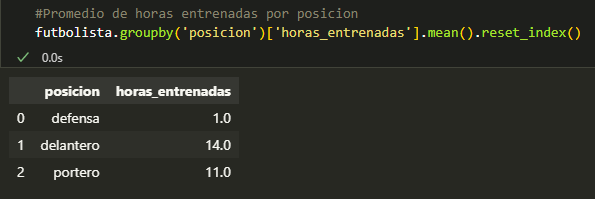
10 Macarena Álvarez Luna 53 M 55.0 1.62 262.0

13 Carolina Rubio Moreno 20 M 61.0 1.77 194.0





**Mostrar las horas entrenadas por posición.**



### Aplicar una función de agregación por grupos

Una vez dividido el DataFame en grupos, es posible aplicar funciones de agregación a cada grupo mediante el siguiente método:

* df.groupby(columnas).agg(funciones) : Devuelve un DataFrame con el resultado de aplicar las funciones de agregación de la lista funciones a cada uno de los DataFrames que resultan de dividir el DataFrame según las columnas de la lista columnas.

Una función de agregación toma como argumento una lista y devuelve un único valor. Algunas de las funciones de agregación más comunes son:

* np.min : Devuelve el mínimo de una lista de valores.
* np.max : Devuelve el máximo de una lista de valores.
* np.count\_nonzero : Devuelve el número de valores no nulos de una lista de valores.
* np.sum : Devuelve la suma de una lista de valores.
* np.mean : Devuelve la media de una lista de valores.
* np.std : Devuelve la desviación típica de una lista de valores.

**>>> import** pandas **as** pd

**>>>** df = pd.read\_csv(

'https://raw.githubusercontent.com/asalber/manual-python/master/datos/colesterol.csv')

**>>>** print(df.groupby('sexo').agg(np.mean))

edad peso altura colesterol

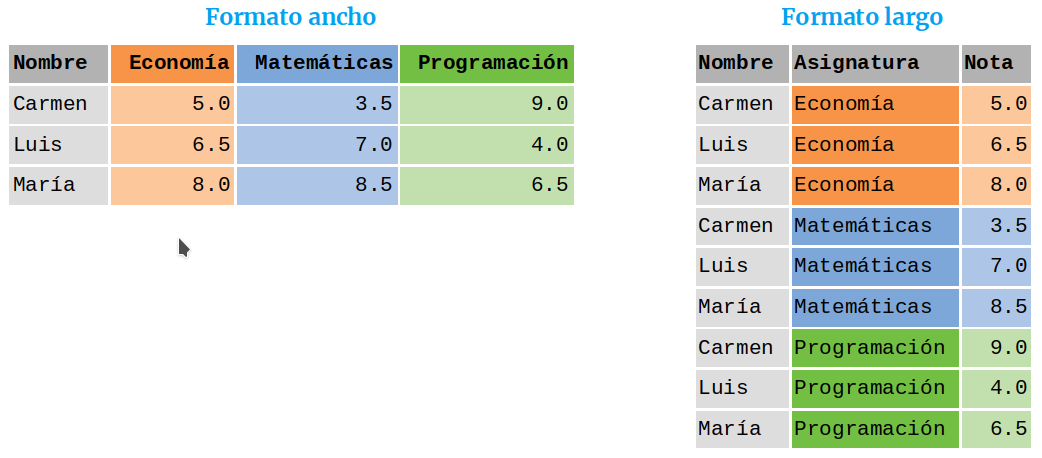
sexo

H 40.875000 80.714286 1.837500 228.375

M 34.666667 59.500000 1.676667 207.200

### Reestructurar un DataFrame

A menudo la disposición de los datos en un DataFrame no es la adecuada para su tratamiento y es necesario reestructurar el DataFrame. Los datos que contiene un DataFrame pueden organizarse en dos formatos: ancho y largo.



### Convertir un DataFrame a formato largo

Para convertir un DataFrame de formato ancho a formato largo (columnas a filas) se utiliza el siguiente método:

* df.melt(id\_vars=id-columnas, value\_vars=columnas, var\_name=nombre-columnas, var\_value=nombre-valores) : Devuelve el DataFrame que resulta de convertir el DataFrame df de formato ancho a formato largo. Todas las columnas de lista columnas se reestructuran en dos nuevas columnas con nombres nombre-columnas y nombre-valores que contienen los nombres de las columnas originales y sus valores, respectivamente. Las columnas en la lista id-columnas se mantienen sin reestructurar. Si no se pasa la lista columnas entonces se reestructuran todas las columnas excepto las columnas de la lista id-columnas.

**>>> import** pandas **as** pd

**>>>** datos = {'nombre':['María', 'Luis', 'Carmen'],

**...** 'edad':[18, 22, 20],

**...** 'Matemáticas':[8.5, 7, 3.5],

**...** 'Economía':[8, 6.5, 5],

**...** 'Programación':[6.5, 4, 9]}

**>>>** df = pd.DataFrame(datos)

**>>>** df1 = df.melt(id\_vars=['nombre', 'edad'], var\_name='asignatura', value\_name='nota')

**>>>** print(df1)

nombre edad asignatura nota

0 María 18 Matemáticas 8.5

1 Luis 22 Matemáticas 7.0

2 Carmen 20 Matemáticas 3.5

3 María 18 Economía 8.0

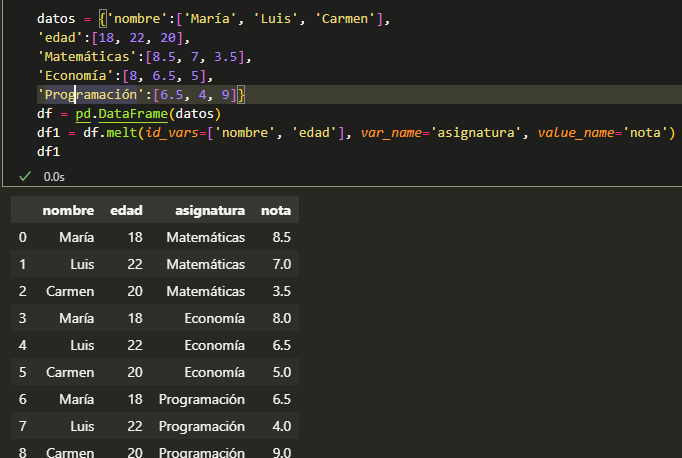
4 Luis 22 Economía 6.5

5 Carmen 20 Economía 5.0

6 María 18 Programación 6.5

7 Luis 22 Programación 4.0

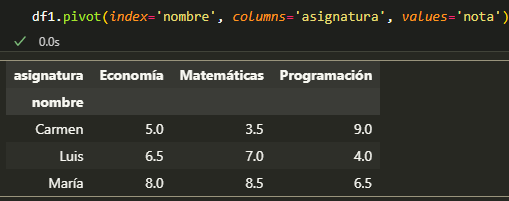
8 Carmen 20 Programación 9.0



### Convertir un DataFrame a formato ancho

Para convertir un DataFrame de formato largo a formato ancho (filas a columnas) se utiliza el siguiente método:

* df.pivot(index=filas, columns=columna, values=valores) : Devuelve el DataFrame que resulta de convertir el DataFrame df de formato largo a formato ancho. Se crean tantas columnas nuevas como valores distintos haya en la columna columna. Los nombres de estas nuevas columnas son los valores de la columna columna mientras que sus valores se toman de la columna valores. Los nombres del índice del nuevo DataFrame se toman de los valores de la columna filas.



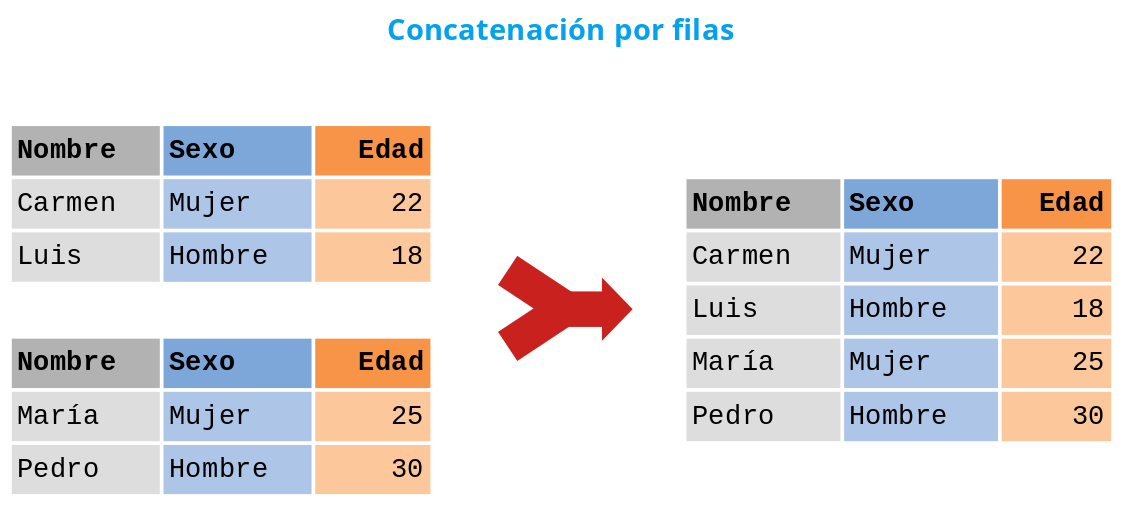
### Combinar varios DataFrames

Dos o más DataFrames pueden combinarse en otro DataFrame. La combinación puede ser de varias formas:

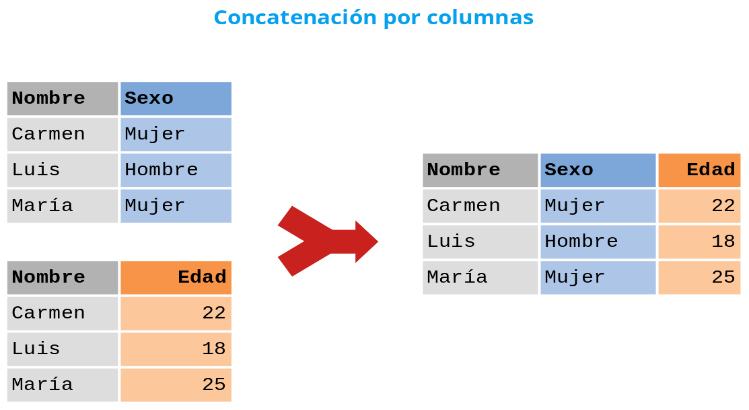
* Concatenación: Combinación de varios DataFrames concatenando sus filas o columnas.
* Mezcla: Combinación de varios DataFrames usando columnas o índices comunes.

### Concatenación de DataFrames

* Concatenación de filas. Las filas de los DataFrames se concatenan unas a continuación de las otras para formar el nuevo DataFrame. Para ello es necesario que los DataFrames que se combinen tengan el mismo índice de columnas.



* Concatenación de columnas. Las columnas de los DataFrames se concatenan unas a continuación de las otras para formar el nuevo DataFrame. Para ello es necesario que los DataFrames que se combinen tengan el mismo índice de filas.

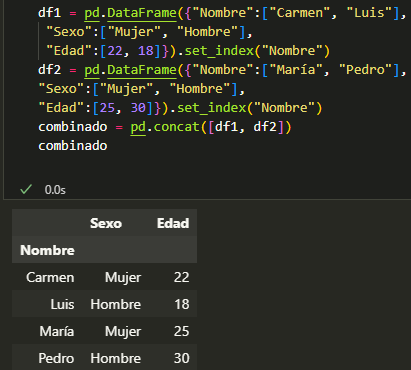


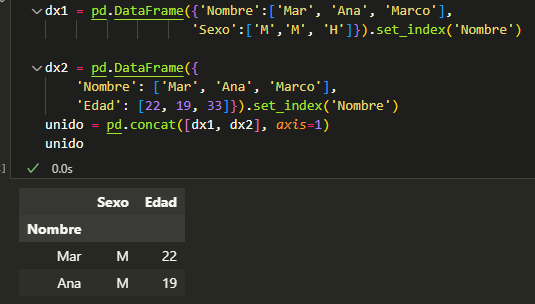
Para concatenar dos o más DataFrames se utiliza el siguiente método:

* df.concat(dataframes, axis = eje): Devuelve el DataFrame que resulta de concatenar los DataFrames de la lista dataframes. Si eje es 0 (valor por defecto) la concatenación se realiza por filas, y si eje es 1 se realiza por columnas.

Si los DataFrames que se concatenan por filas no tienen el mismo índice de columnas, el DataFrame resultante incluirá todas las columnas existentes en los DataFrames y rellenará con valores NaN los datos no disponibles. Si los DataFrames que se concatenan por columnas no tienen el mismo índice de filas, el DataFrame resultante incluirá todas las filas existentes en los DataFrames y rellenará con valores NaN los datos no disponibles.

Ejemplo 1 de unión de dataframes





### Mezcla de DataFrames

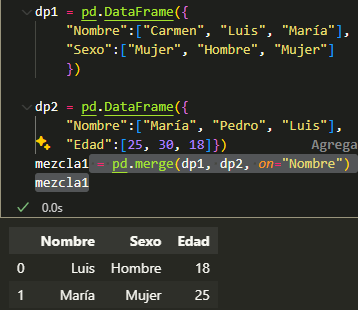
La mezcla de DataFrames permite integrar filas de dos DataFrames que contienen información en común en una o varias columnas o índices que se conocen como clave.

Para mezclar dos DataFrames se utiliza el siguiente método:

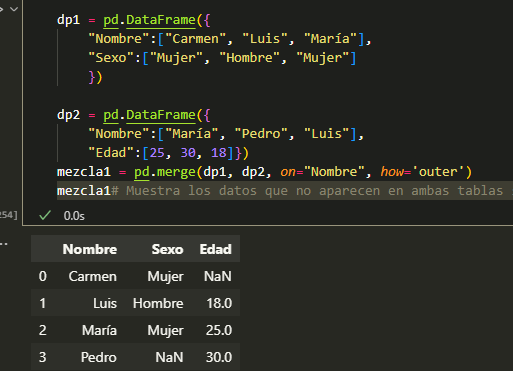
* df.merge(df1, df2, on = clave, how = tipo): Devuelve el DataFrame que resulta de mezclar el DataFrame df2 con el DataFrame df1, usando como claves las columnas de la lista clave y siguiendo el método de mezcla indicado por tipo.

El tipo de mezcla puede ser

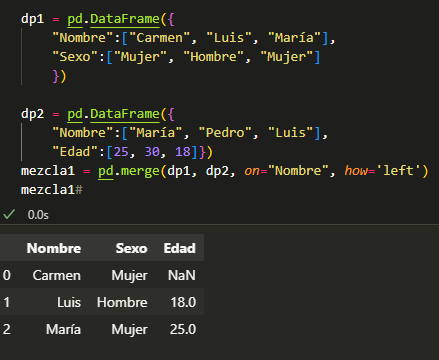
"inner" (por defecto): El DataFrame resultante solo contiene las filas cuyos valores en la clave están en los dos DataFrames. Es equivalente a la intersección de conjuntos.



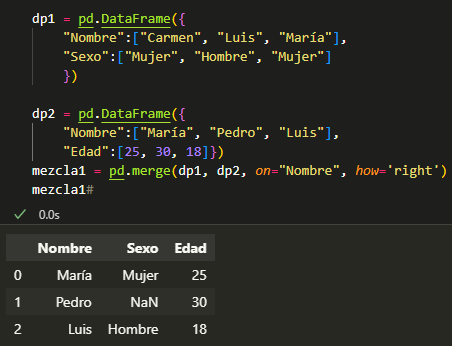
* "outer": El DataFrame resultante contiene todas las filas de los dos DataFrames. Si una fila de un DataFrame no puede emparejarse con otra los mismos valores en la clave en el otro DataFrame, la fila se añade igualmente al DataFrame resultante rellenando las columnas del otro DataFrame con el valor NaN. Es equivalente a la unión de conjuntos.



* "left": El DataFrame resultante contiene todas las filas del primer DataFrame y descarta las filas del segundo DataFrame que no pueden emparejarse con alguna fila del primer DataFrame a través de la clave.



* "right": El DataFrame resultante contiene todas las filas del segundo DataFrame y descarta las filas del primer DataFrame que no pueden emparejarse con alguna fila del segundo DataFrame a través de la clave.



### Tablas dinámicas.

*tabla\_dinamica = pd.pivot\_table(*

*df,*

*values='Ventas', # Columna que queremos agregar*

*index='Producto', # Fila de la tabla*

*columns='Mes', # Columnas de la tabla*

*aggfunc='sum', # Función de agregación: sum, mean, count, etc.*

*fill\_value=0 # Rellena los valores faltantes con 0*

*)*

*print(tabla\_dinamica)*

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

## Numpy.

### ¿Qué es Numpy?

Numpy es una biblioteca de Python utilizada para trabajar con matrices.

También tiene funciones para trabajar en el dominio del álgebra lineal, transformada de Fourier y matrices.

Numpy fue creado en 2005 por Travis Oliphant. Es un proyecto de código abierto y se puede usar libremente.

Numpy significa Python numérico.

### \*\*¿Qué ofrece Numpy?\*\*

Arrays eficientes (ndarray). Operaciones matemáticas avanzadas

Álgebra lineal, transformadas de Fourier, estadísticas, etc. Integración con C/C++ y otras bibliotecas científicas, Herramientas para generar datos aleatorios.

Numpy es una biblioteca fundamental para la computación científica en Python.

Permite trabajar con arrays (arreglos) de forma eficiente y ofrece herramientas poderosas para el cálculo numérico.

### ¿Por qué utilizar Numpy?.\*\*.

En Python tenemos listas que cumplen la función de matrices, pero son lentas de procesar.

Numpy tiene como objetivo proporcionar un objeto de matriz que sea hasta 50 veces más rápido que las listas tradicionales de Python.

El objeto de matriz en Numpy se llama ndarray, proporciona muchas funciones de soporte que hacen que trabajar con él ndarraysea muy fácil.

Las matrices se utilizan con mucha frecuencia en la ciencia de datos, donde la velocidad y los recursos son muy importantes.

\*Ciencia de Datos:\* es una rama de la informática donde estudiamos cómo almacenar, utilizar y analizar datos para derivar información de ellos.

### ¿Por qué Numpy es más rápido que las listas?.

Las matrices Numpy se almacenan en un lugar continuo en la memoria, a diferencia de las listas,

por lo que los procesos pueden acceder a ellas y manipularlas de manera muy eficiente.

Este comportamiento se denomina localidad de referencia en informática.

Esta es la razón principal por la que Numpy es más rápido que las listas. Además, está optimizado para funcionar con las arquitecturas de CPU más recientes

.

### ¿En qué lenguaje está escrito Numpy?\*\*.

Numpy es una biblioteca de Python y está escrita parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C++.

### ⚙️ Principales características.

\*\*ndarray:\*\* Objeto de arreglo multidimensional muy rápido y versátil.

Operaciones vectorizadas: Realiza cálculos sin necesidad de usar bucles manuales.

\*\*Broadcasting:\*\* Aplica operaciones entre arrays de distintas formas automáticamente.

\*\*Álgebra lineal:\*\* Multiplicación de matrices, determinantes, valores propios, etc.

\*\*Generación de números aleatorios:\*\* Para simulaciones, pruebas o análisis estadístico.

\*\*Transformadas de Fourier:\*\* Útiles para procesar señales o análisis de frecuencias.

La ventaja de Numpy frente a las listas predefinidas en Python es que el procesamiento de los arrays se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones.

### Creación de arrays

Para crear un array se utiliza la siguiente función de Numpy

* np.array(lista) : Crea un array a partir de la lista o tupla lista y devuelve una referencia a él. El número de dimensiones del array dependerá de las listas o tuplas anidadas en lista:
* Para una lista de valores se crea un array de una dimensión, también conocido como vector.
* Para una lista de listas de valores se crea un array de dos dimensiones, también conocido como matriz.
* Para una lista de listas de listas de valores se crea un array de tres dimensiones, también conocido como cubo.
* Y así sucesivamente. No hay límite en el número de dimensiones del array más allá de la memoria disponible en el sistema.

 Los elementos de la lista o tupla deben ser del mismo tipo.

**Otras funciones útiles que permiten generar arrays son:**

* np.empty(dimensiones) : Crea y devuelve una referencia a un array vacío con las dimensiones especificadas en la tupla dimensiones.
* np.zeros(dimensiones) : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla dimensiones cuyos elementos son todos ceros.
* np.ones(dimensiones) : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla dimensiones cuyos elementos son todos unos.
* np.full(dimensiones, valor) : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla dimensiones cuyos elementos son todos valor.
* np.identity(n) : Crea y devuelve una referencia a la matriz identidad de dimensión n.
* np.arange(inicio, fin, salto) : Crea y devuelve una referencia a un array de una dimensión cuyos elementos son la secuencia desde inicio hasta fin tomando valores cada salto.
* np.linspace(inicio, fin, n) : Crea y devuelve una referencia a un array de una dimensión cuyos elementos son la secuencia de n valores equidistantes desde inicio hasta fin.
* np.random.random(dimensiones) : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla dimensiones cuyos elementos son aleatorios.

**Atributos de un array**

Existen varios atributos y funciones que describen las características de un array.

* a.ndim : Devuelve el número de dimensiones del array a.
* a.shape : Devuelve una tupla con las dimensiones del array a.
* a.size : Devuelve el número de elementos del array a.
* a.dtype: Devuelve el tipo de datos de los elementos del array a.

**Ejemplos prácticos.**

*#Importar la librería de numpy.*

*import numpy as np*

*import matplotlib.pyplot as plt*

*#verificar que versión de numpy tenemos instalado*

*print('Version de Numpy: ', np.\_\_version\_\_)*

#Version de Numpy: 2.3.1

**#Crear mi primer array de 1 dimensión en numpy**

mi\_array = np.array([1, 2, 3, 4, 5, 6])#Array de 1 dimensión

mi\_array

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

#sumar 2 valores al array

mi\_array + 2

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

#Agregar 2 nuevos valores al array

mi\_array = np.append(mi\_array, [10, 11, 12, 13, 14, 15, 2, 3, 1, 5, 4, 3, 5, 4])

mi\_array

array([ 1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 15, 2, 3, 1, 5, 4,

3, 5, 4])

#Acceder a un elemento del array

mi\_array[4]

#Acceder a varios elementos del array

mi\_array[5:10]

array([ 6, 10, 11, 12, 13])

#Eliminar un elemento del array

mi\_array = np.delete(mi\_array, 1)

mi\_array

#Eliminar varios elementos.

#mi\_array= np.delete(mi\_array, [1, 2, 3])

array([ 1, 3, 4, 5, 6, 10, 11, 12, 13, 14, 15, 2, 3, 1, 5, 4, 3,

5, 4])

#filtrar elementos menores de 5

mi\_array[mi\_array < 5]

array([1, 3, 4, 2, 3, 1, 4, 3, 4])

#número mínimo del array

mi\_array.min()

#Número máximo del array

mi\_array.max()

#obtener la media del array

mi\_array.mean()

np.float64(6.368421052631579)

#Obtener la desviación estándar del array

mi\_array.std() np.float64(4.451026213024416)

#Mostrar los valores únicos de array

mi\_array\_unique = np.unique(mi\_array)

mi\_array\_unique

#Mostrar el tamaño del array

mi\_array.size

#Suma total del array

mi\_array.sum()

#mostrar el tipo de array

mi\_array.dtype

#Devuelve una tupla con las dimensiones del array a.shape

mi\_array.shape

#a.ndim : Devuelve el número de dimensiones del array a.

mi\_array.ndim

#Array que cumplen una condición que sean pares

mi\_array[(mi\_array % 2==0)]

array([ 4, 6, 10, 12, 14, 2, 4, 4])

#Crear 1 array de 2 dimensiones

D2\_array = [[4,7,16, 13],

            [1,5,6, 14],

            [2,8,9, 15],

            [10,11, 12, 3]]

D2\_array = np.array(D2\_array)

D2\_array.ndim

#D2\_array.shape

#Ordenar el array

D2\_array.sort(*axis*=0)

D2\_array

#Crea y devuelve una referencia a un array vacío con las dimensiones especificadas en la tupla dimensiones.

np.empty((4, 8))

*array([[1. , 0. , 0.16, 1. , 1. , 0. , 0. , 1. ],*

*[1. , 1. , 0. , 1. , 0. , 1. , 0. , 1. ],*

*[0. , 1. , 1. , 1. , 0. , 0. , 1. , 1. ],*

*[1. , 0. , 1. , 1. , 1. , 0. , 0.75, 1. ]])*

#Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla dimensiones cuyos

# elementos son todos ceros.

ceros = np.zeros((4,4))

ceros

*array([[0., 0., 0., 0.],*

*[0., 0., 0., 0.],*

*[0., 0., 0., 0.],*

*[0., 0., 0., 0.]])*

#Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla dimensiones cuyos

# elementos son todos unos.

unos = np.ones((3,5))

unos

*array([[1., 1., 1., 1., 1.],*

*[1., 1., 1., 1., 1.],*

*[1., 1., 1., 1., 1.]])*

#Crea y devuelve una referencia a un array de una dimensión cuyos elementos son la secuencia de n valores

# equidistantes desde inicio hasta fin.

secuencia = np.linspace(0, 10, 5)

secuencia

*array([ 0. , 2.5, 5. , 7.5, 10. ])*

# Crea y devuelve una referencia a un array de una dimensión cuyos elementos son la secuencia desde inicio hasta

# fin tomando valores cada salto.

np.arange(1, 5, 1)

*array([1, 2, 3, 4])*

#np.full(dimensiones, valor) : Crea y devuelve una referencia a un array con las dimensiones especificadas en la

# tupla dimensiones cuyos elementos son todos valor.

np.full((2, 4), 3)

*array([[3, 3, 3, 3],*

*[3, 3, 3, 3]])*

from numpy import pi

rad = np.linspace(0, 2\*pi, 100)

rad

#Obtener el seno y coseno de los valores del array rad

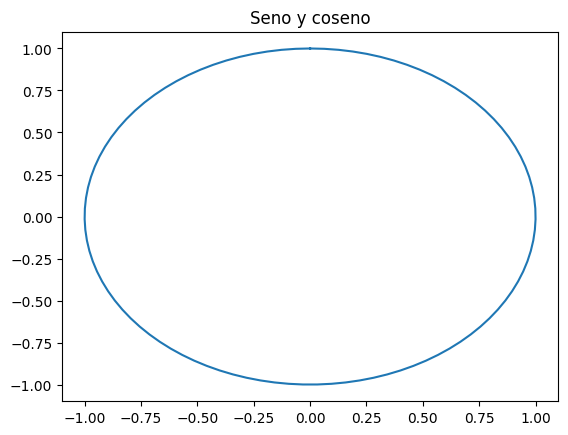
seno = np.sin(rad)

coseno = np.cos(rad)

plt.plot(seno, coseno)

plt.title('Seno y coseno')

plt.show()

**

# # Matriz identidad

np.eye(4)

#Crear una matriz de 3 dimensiones

D3\_array = [[[1,2,3],

             [4,5,6]]]

D3\_array = np.array(D3\_array)

D3\_array.ndim

#Contenar array

a = np.linspace(10, 20, 6)

b = np.linspace(5, 25, 6)

print(a + b)

#concatenar

unión = np.concatenate((a, b))

union

*array([10., 12., 14., 16., 18., 20., 5., 9., 13., 17., 21., 25.])*

#rg.random para generar ndarray de números aleatorios

rg = np.random.default\_rng(2)

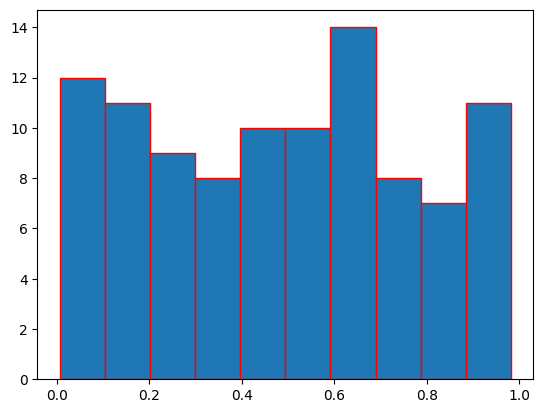
aleatorio = rg.random(100)

aleatorio

#crear un histograma de los valores aleatorios

plt.hist(aleatorio, *bins*=10, *edgecolor*='red')

plt.show()

**

#Generar números enteros aleatorios

enteros = rg.integers(20, *size*=1000)

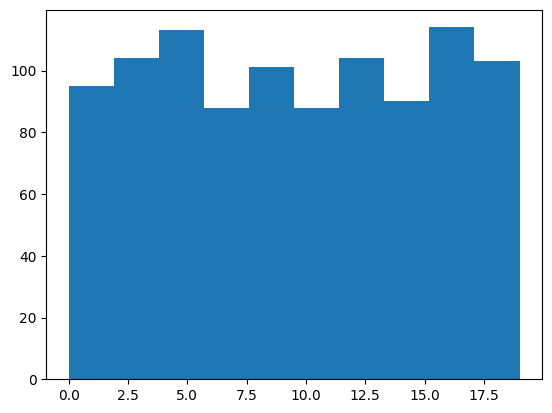
#enteros

#Mostrar los números enteros en un histograma

plt.hist(enteros)

plt.yticks(np.arange(0, 120, 20))

plt.show()

**

#Numeros aleatorios sin repetir

rg.choice(26, *size*= 10, *replace*= False)

*array([20, 15, 9, 16, 8, 25, 18, 0, 3, 14])*

estadístico = rg.integers(20, *size*=6)

estadistico.std()

#Ejemplo

np1 = rg.integers(20, *size*=(3, 3))

np2 = rg.integers(20, *size*=(3, 3))

print(np1, "\n\n", np2)

#Utilizar vstack para unir np1 y np2 verticalmente.

np.vstack((np1, np2))

### Filtrado de elementos de un array

Una característica muy útil de los arrays es que es muy fácil obtener otro array con los elementos que cumplen una condición.

* a[condicion] : Devuelve una lista con los elementos del array a que cumplen la condición condicion.
* **>>>** a = np.array([[1, 2, 3], [4, 5, 6]])
* **>>>** print(a[(a % 2 == 0)])
* [2 4 6]
* **>>>** print(a[(a % 2 == 0) & (a > 2)])
* [2 4]

### Operaciones matemáticas con arrays

Existen dos formas de realizar operaciones matemáticas con arrays: a nivel de elemento y a nivel de array.

Las operaciones a nivel de elemento operan los elementos que ocupan la misma posición en dos arrays. Se necesitan, por tanto, dos arrays con las mismas dimensiones y el resultado es una array de la misma dimensión.

Los operadores mamemáticos +, -, \*, /, %, \*\* se utilizan para la realizar suma, resta, producto, cociente, resto y potencia a nivel de elemento.

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

**>>>** b = np.array([[1, 1, 1], [2, 2, 2]])

**>>>** print(a + b )

[[2 3 4]

[6 7 8]]

**>>>** print(a / b)

[[1. 2. 3. ]

[2. 2.5 3. ]]

**>>>** print(a \*\* 2)

[[ 1 4 9]

[16 25 36]]

Álgebra matricial

Numpy incorpora funciones para realizar las principales operaciones algebraicas con vectores y matrices. La mayoría de los métodos algebráicos se agrupan en el submódulo linalg.

Producto escalar de dos vectores

Para realizar el producto escalar de dos vectores se utiliza el operador @ o el siguiente método:

* u.dot(v): Devuelve el producto escalar de los vectores u y v.
* **>>> import** numpy **as** np
* **>>>** a = np.array([1, 2, 3])
* **>>>** b = np.array([1, 0, 1])
* **>>>** print(a @ b)
* 4
* **>>>** print(a.dot(b))
* 4

### Módulo de un vector

Para calcular el módulo de un vector se utiliza el siguiente método:

* norm(v): Devuelve el módulo del vector v.
* **>>> import** numpy **as** np
* **>>>** a = np.array([3, 4])
* **>>>** print(np.linalg.norm(a))
* 5.0

### Producto de dos matrices

Para realizar el producto matricial se utiliza el mismo operador @ y método que para el producto escalar de vectores:

* a.dot(b) : Devuelve el producto matricial de las matrices a y b siempre y cuando sus dimensiones sean compatibles.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 2, 3], [4, 5, 6]])
* **>>>** b = np.array([[1, 1], [2, 2], [3, 3]])
* **>>>** print(a @ b)
* [[14 14]
* [32 32]]
* **>>>** print(a.dot(b))
* [[14 14]
* [32 32]]

### Matriz traspuesta

Para trasponer una matriz se utiliza el método

* a.T : Devuelve la matriz traspuesta de la matriz a.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 2, 3], [4, 5, 6]])
* **>>>** print(a.T)
* [[1 4]
* [2 5]
* [3 6]]

### Traza de una matriz

La traza de una matriz cuadrada se calcula con el siguiente método:

* a.trace() : Devuelve la traza (suma de la diagonal principal) de la matriz cuadrada a.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
* **>>>** print(a.trace())
* 15

### Determinante de una matriz

El determinante de una matriz cuadrada se calcula con la siguiente función:

* det(a) : Devuelve el determinante de la matriz cuadrada a.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 2], [3, 4]])
* **>>>** print(np.linalg.det(a))
* -2.0

### Matriz inversa

La inversa de una matriz se calcula con la siguiente función:

* inv(a) : Devuelve la matriz inversa de la matriz cuadrada a.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 2], [3, 4]])
* **>>>** print(np.linalg.inv(a))
* [[-2. 1. ]
* [ 1.5 -0.5]]

### Autovalores de una matriz

Los autovalores de una matriz cuadrada se calculan con la siguiente función:

* eigvals(a) : Devuelve los autovalores de la matriz cuadrada a.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 1, 0], [1, 2, 1], [0, 1, 1]])
* **>>>** print(np.linalg.eigvals(a))
* [ 3.00000000e+00 1.00000000e+00 -3.36770206e-17]

### Autovectores de una matriz

Los auto vectores de una matriz cuadrada se calculan con la siguiente función:

* eig(a) : Devuelve los autovalores y los auto vectores asociados de la matriz cuadrada a.
* **>>> import** numpy **as** np
* **>>>** a = np.array([[1, 1, 0], [1, 2, 1], [0, 1, 1]])
* **>>>** print(np.linalg.eig(a))
* (array([ 3.00000000e+00, 1.00000000e+00, -3.36770206e-17]), array([[-4.08248290e-01, 7.07106781e-01, 5.77350269e-01],
* [-8.16496581e-01, 2.61239546e-16, -5.77350269e-01],
* [-4.08248290e-01, -7.07106781e-01, 5.77350269e-01]]))

### Solución de un sistema de ecuaciones

Para resolver un sistema de ecuaciones lineales se utiliza la función siguiente:

* solve(a, b) : Devuelve la solución del sistema de ecuaciones lineales con los coeficientes de la matriz a y los términos independientes de la matriz b.
* **>>> import** numpy **as** np
* *# Sistema de dos ecuaciones y dos incógnitas*
* *# x + 2y = 1*
* *# 3x + 5y = 2*
* **>>>** a = np.array([[1, 2], [3, 5]])
* **>>>** b = np.array([1, 2])
* **>>>** print(np.linalg.solve(a, b))

[-1. 1.]

## Matplotlib.

Es una biblioteca de Python para crear gráficos en 2D y 3D. Fue desarrollada en 2002 por John Hunter con el objetivo de replicar las capacidades gráficas de MATLAB en Python.

### 🎯 ¿Para qué sirve?\*\*

Sirve para:

Visualizar datos de forma clara y atractiva.

Crear gráficos estáticos, animados e interactivos.

Personalizar cada aspecto del gráfico: colores, estilos, etiquetas, leyendas, etc.

Integrarse fácilmente con Numpy, pandas y otras bibliotecas de análisis de datos.

### ⚙️ ¿Cómo funciona?\*\*

Matplotlib se basa en dos conceptos clave:

Figure: el lienzo general donde se dibujan los gráficos.

Axes: los sistemas de coordenadas dentro de la figura.

El módulo más usado es pyplot, que simplifica la creación de gráficos con funciones como plot(), bar(), scatter(), etc.

Ejemplo.

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import os

#ver la ruta actual

#print(os.getcwd())

#Crear un gráfico simple de población por estado

eje = ['Tabasco', 'Veracruz', 'Puebla', 'Oaxaca', 'Jalisco', 'Yucatán']

eje\_Y = [10000, 20000, 15000, 50000, 80000, 100000]

plt.bar(eje\_x, eje\_Y, *color* = 'blue', *width*=0.5)

plt.title('Poblacion por estado')#Mostrar titulo

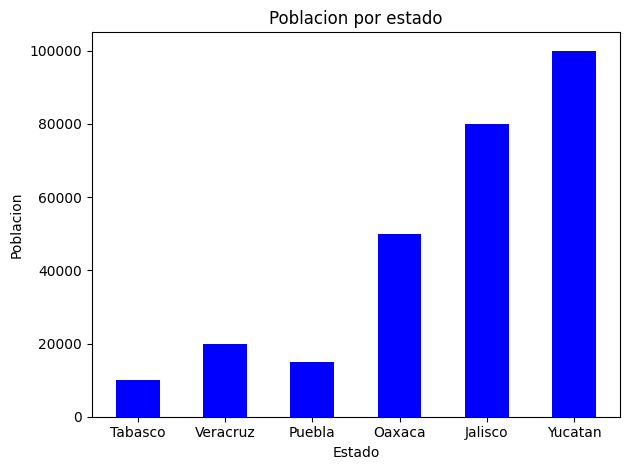
plt.xlabel('Estado')#Etiquta eje x

plt.ylabel('Poblacion')#Etiquta eje y

plt.grid(False)#Mostrar cuadriculas

plt.tight\_layout()

plt.show()



#Generar otra grafica simple de línea.

#Cambiar el tamaño de la figura

plt.Figure(*física*=(12, 20))

plt.plot(eje\_x, eje\_Y, *marker*='\*', *linestyle*= '--', *color*='red')

plt.title('Población por estado')

#Editar los números que aparecen en los ejes y

plt.yticks(np.arange(0, 110000, 10000))

plt.show()

### Gráfico de línea.

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

#Crear un gráfico de pastel simple de la población por estado

plt.Figure(*figsize*=(12, 10))

plt.pie(eje\_Y, *labels*=eje\_x, *autopct*='%1.1f%%', *startangle*=140)

plt.title('Población por estado')

#Activar la leyenda

plt.legend(eje\_x, *loc*='upper right')

plt.axis('equal')

plt.savefig('Grafico\_Pastel.png')

plt.show()

### Grafica de pastel.

Gráfico, Gráfico circular

El contenido generado por IA puede ser incorrecto.

#Grafica básica

EjeX = ['a', 'b', 'c', 'd']

dat1 = [10, 20, 30, 40]

dat2 = [40, 30, 50, 60]

plt.plot(EjeX, dat1)

plt.plot(EjeX, dat2)

plt.xlabel('EJE X')

plt.legend(eje\_x)

plt.title('Grafica sencilla')

plt.show()

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

### Gráficos de líneas usando numpy.

#Grafico usando random

años = [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019]

N1 = np.random.rand(10)\*100

N2 = np.random.rand(10)\*200+100

N3 = np.random.rand(10)\*300+200

N4 = np.random.rand(10)\*400+300

N5 = np.random.rand(10)\*500+400

plt.plot(años, N1, *marker*='\*', *linestyle*='-')#marker='\*' Agrega un marcador

plt.plot(años, N2, *marker*='\*', *linestyle*=':')

plt.plot(años, N3, *marker*='\*', *linestyle*='--')

plt.plot(años, N4, *marker*='\*', *linestyle*='-.')

plt.plot(años, N5, *marker*='\*', *linestyle*=' ')

#Activar la leyenda

plt.legend(años, *loc*='upper right')#Mostrar leyendas

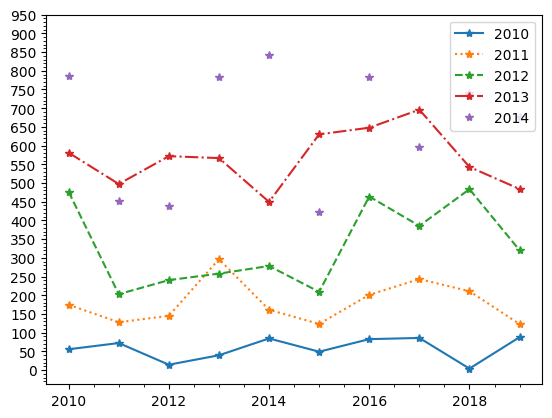
#Personalizar el eje de puntaje de 0 a 1000 en incremento de 50

plt.yticks(np.arange(0, 1000, 50))

#Activar marcas menores

plt.minorticks\_on()

plt.show()



### Graficas de barras verticales.

#Grafica verticales

plt.bar(eje\_x, eje\_Y)

plt.title('Grafico vertical')

plt.xlabel('Estado')#Etiquta eje x

plt.ylabel('Población')#Etiquta eje y

#plt.legend(eje\_x, loc='upper right')

plt.show()

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

### Gráficas horizontales de barras.

#Graficas horizontales

plt.barh(eje\_x, eje\_Y)

plt.title('Grafico vertical')

plt.xlabel('Estado')#Etiquta eje x

plt.ylabel('Población')#Etiquta eje y

#plt.legend(eje\_x, loc='upper right')

plt.show()

Gráfico, Gráfico de barras, Gráfico de embudo

El contenido generado por IA puede ser incorrecto.

### Graficas de dispersion.

#Grafica de dispersión

plt.scatter(eje\_x, eje\_Y, *marker*='\*')

#plt.legend(['Tabasco', 'Veracruz', 'Puebla', 'Oaxaca', 'Jalisco', 'Yucatán'], loc= 'upper right')

plt.title('grafico de dispersión')

plt.xlabel('Estado')

plt.show()

Gráfico, Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

#Grafico vertical

Ex = np.arange(0, 10)

plt.bar(Ex, N5, *width*=1/5)

plt.bar(Ex+0.2, N4, *width*=1/5)

plt.bar(Ex+0.4, N3, *width*=1/5)

plt.bar(Ex+0.6, N2, *width*=1/5)

plt.bar(Ex+0.8, N5, *width*=1/5)

plt.yticks(np.arange(0, 1000, 50))

plt.show()

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

#Crear otro ejemplo básico de grafico de dispersion

plt.scatter(años, N1, *color*='red')

plt.scatter(años, N2, *color*='yellow')

plt.scatter(años, N3, *color*='blue')

plt.scatter(años, N4, *color*='green')

plt.scatter(años, N4, *linestyle*='--', *color*='orange')

plt.legend(años, *loc*='upper left')

plt.colorbar()#Muestra la barra de colores

plt.grid()

plt.show()

Gráfico, Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

### Histogramas

#Crear un histograma desde cero

edades = [15, 16, 17,18,19,20,21,22,23,24,25,26,27,28,29,30]

bins = [15, 20, 22, 25, 35]

plt.hist(edades, *bins*=bins, *edgecolor*='red', *color*='green', *alpha*=0.5)

plt.show()

Gráfico, Histograma

El contenido generado por IA puede ser incorrecto.

#Grafica de cajas Boxplots

plt.boxplot(edades)

plt.title("Gráfico de caja")

plt.show()

### gráficos de cajas.

Gráfico, Gráfico de cajas y bigotes

El contenido generado por IA puede ser incorrecto.

x = [1, 2, 3, 4]

plt.subplot(1, 2, 1)

plt.plot(x, [i\*\*2 for i in x])

plt.title("Cuadrados")

plt.subplot(1, 2, 2)

plt.plot(x, [i\*\*3 for i in x])

plt.title("Cubos")

plt.tight\_layout()

plt.show()

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

#Ejemplo práctico de graficos.

Xnumpy = np.array([3, 5, 6, 10, 12])

Yarray = np.array([1, 3, 5, 6, 8])

plt.plot(Xnumpy, Yarray, *marker*='o', *color*='red', *linestyle*='--', *linewidth*=2)

plt.show()

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

### Gráficos a partir de una tabla dinámica

Texto

El contenido generado por IA puede ser incorrecto.

import matplotlib.pyplot as plt

tabla1.plot(*kind*='bar', *legend*=False, *color*='skyblue')

plt.title('Precio Promedio por Estado')

plt.xlabel('Estado')

plt.ylabel('Precio Promedio')

plt.xticks(*rotation*=45)  # Rotar etiquetas si es necesario

plt.tight\_layout()       # Ajusta el gráfico para que no se corten las etiquetas

plt.show()

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

Ejemplo 2 grafica de pastel.

plt.figure(*figsize*=(10, 6))

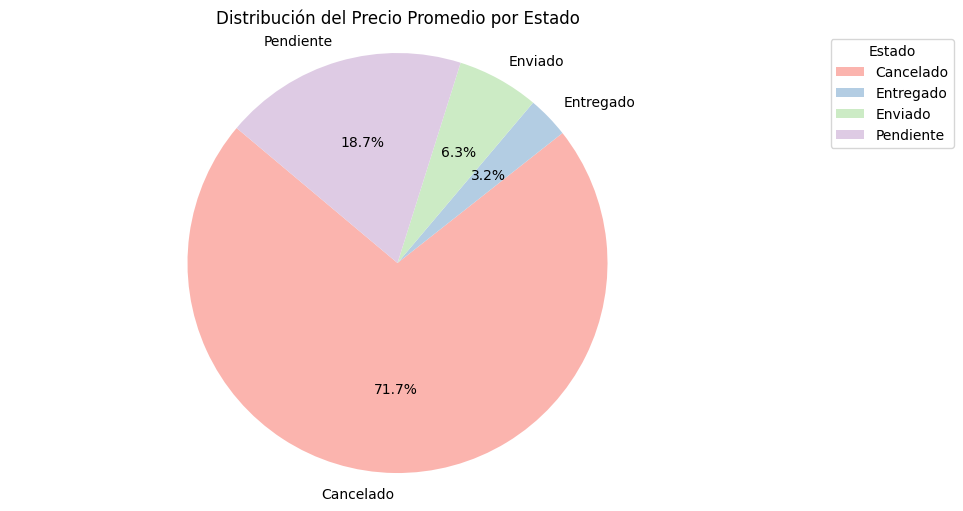
plt.pie(tabla1['Precio'], *labels*=tabla1.index, *autopct*='%1.1f%%', *startangle*=140, *colors*=plt.cm.Pastel1.colors)

plt.title('Distribución del Precio Promedio por Estado')

plt.legend(*title*='Estado', *bbox\_to\_anchor*=(1.05, 1) , *loc*='upper left')

plt.axis('equal')  # Hace que el gráfico sea un círculo perfecto

plt.show()



### Pandas y SQL.

Requisitos previos

Antes de empezar, asegúrate de tener instalados los paquetes necesarios:

*pip install pandas pyodbc sqlalchemy*

* **pandas** → para manejar los datos en DataFrames.
* **pyodbc** → controlador ODBC para conectarse a SQL Server.
* **sqlalchemy** → facilita la conexión y consultas.

También necesitas el **driver ODBC de SQL Server** instalado en tu sistema:  
👉 Microsoft ODBC Driver for SQL Server.

**Conexión a SQL Server con pandas**

Existen dos formas principales: pyodbc directo o usando sqlalchemy.

📌 Opción 1: Usando pyodbc

*import pyodbc*

*import pandas as pd*

*# Parámetros de conexión*

*server = "UNITY12\RAMSES"          # o la IP de tu servidor*

*database = "TiendaInformatica"  # tu base de datos*

*username = "sa"               # tu usuario de SQL Server*

*password = "12345"            # tu contraseña*

*# Conexión*

*conn = pyodbc.connect(*

*f"DRIVER={{ODBC Driver 17 for SQL Server}};"*

*f"SERVER={server};"*

*f"DATABASE={database};"*

*f"UID={username};"*

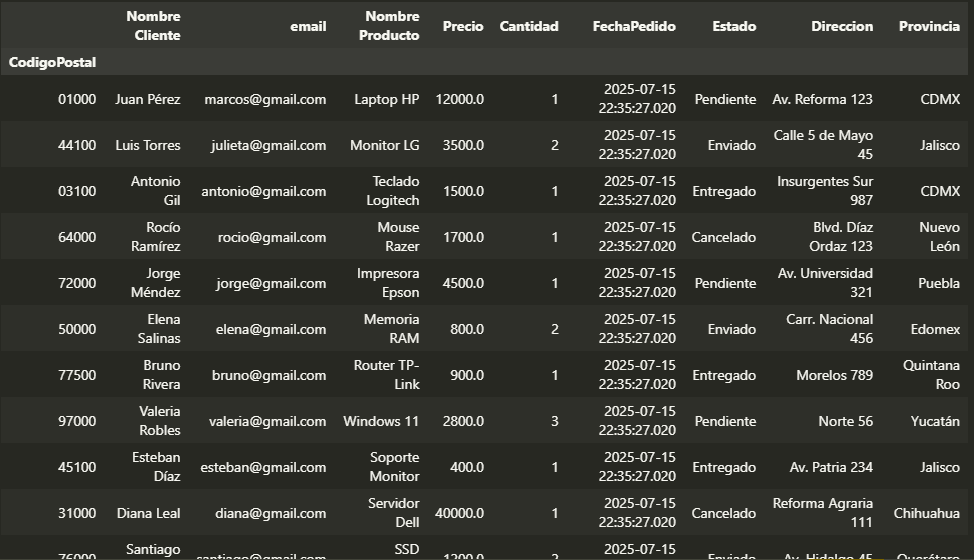
*f"PWD={password}"*

*)*

*# Leer una tabla con pandas*

*df = pd.read\_sql(" SELECT Cli.NOMBRE AS [Nombre Cliente], Cli.email, P.Nombre as [Nombre Producto],P.Precio,DP.Cantidad,PD.FechaPedido,PD.Estado,D.Direccion,D.CodigoPostal,D.Provincia FROM DetallesPedido DP INNER join Pedidos PD ON DP.IdPedido = PD.IdPedido INNER join Productos P ON DP.IdProducto = P.IdProducto INNER join Clientes Cli on PD.IdCliente = Cli.IdCliente INNER join Direcciones D on PD.IdDireccion = D.IdDireccion", conn, index\_col='CodigoPostal')*

*df*

**

**Forma 2**

**Usando sqlalchemy**

*from sqlalchemy import create\_engine*

*import pandas as pd*

*# Crear conexión con sqlalchemy*

*engine = create\_engine(*

*"mssql+pyodbc://sa:12345@localhost/TiendaInformatica?driver=ODBC+Driver+17+for+SQL+Server"*

*)*

*# Leer con pandas*

*df = pd.read\_sql("SELECT TOP 10 \* FROM Clientes", engine)*

*print(df.head())*

## Polars.

**POLARS**

Polars es una biblioteca de código abierto para la manipulación de datos , conocida por ser una de las soluciones de procesamiento de datos más rápidas en una sola máquina. Cuenta con una API bien estructurada y tipificada, expresiva y fácil de usar.

**Rápido**

Polars se desarrolló desde cero pensando en el rendimiento. Su motor de consultas multihilo está escrito en Rust y diseñado para un paralelismo efectivo. Su procesamiento vectorizado y columnar permite algoritmos coherentes con la caché y un alto rendimiento en procesadores modernos.

### ¿Por qué usar polares?

Polars se evaluó en una versión derivada del benchmark independiente TPC-H frente a otras soluciones. Este benchmark busca replicar las operaciones de manipulación de datos utilizadas en la práctica. Polars supera ampliamente a otras soluciones gracias a su motor de ejecución en paralelo, algoritmos eficientes y el uso de vectorización con SIMD (Instrucción Única, Datos Múltiples). En comparación con Pandas, puede alcanzar mejoras de rendimiento de más de 30 veces.

### ¿Qué es Polars?

Es una librería open source para manipulación de datos (similar a pandas).

Fue creada en Rust (lenguaje ultrarrápido y seguro en memoria).

Tiene bindings para Python, Rust y otros lenguajes.

Está diseñada para manejar datasets grandes, incluso mayores a la memoria RAM disponible.

### ¿Para qué sirve?

Análisis de datos tabulares (igual que pandas).

Procesamiento rápido de grandes volúmenes de datos (Big Data en una sola máquina).

Limpieza, transformación y agregación de datos.

Lectura y escritura eficiente en formatos como CSV, Parquet, JSON, IPC/Arrow.

Pipelines optimizados para ciencia de datos, machine learning y ETL.

**¿Cómo funciona?**

Está escrito en Rust y usa Apache Arrow como formato interno de memoria (columnar, muy eficiente).

Tiene dos modos:

Eager API → como pandas, ejecuta todo inmediatamente.

Lazy API → construcción de un plan de ejecución que optimiza todas las operaciones antes de ejecutarlas (similar a Spark o SQL).

Esto significa que Polars optimiza automáticamente el orden de operaciones y reduce el uso de memoria.

### Optimización De Consultas

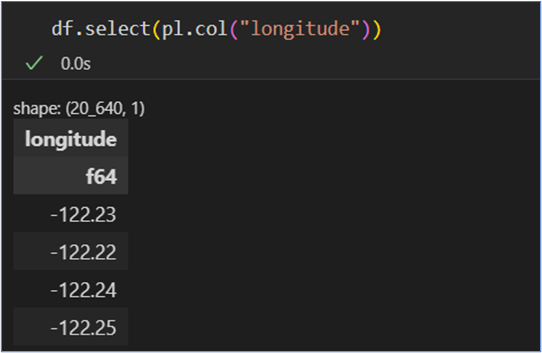
Otro pilar fundamental del rendimiento de Polars es cómo evalúa el código. Por defecto, Pandas utiliza una ejecución eager, realizando las operaciones en el orden en que las has escrito. En cambio, Polars tiene la capacidad de realizar tanto una ejecución eager como una ejecución lazy, donde un optimizador de consultas evalúa todas las operaciones requeridas y traza la forma más eficiente de ejecutar el código. Esto puede incluir, entre otras cosas, reescribir el orden de ejecución de las operaciones o eliminar cálculos redundantes.

### Expresión Select

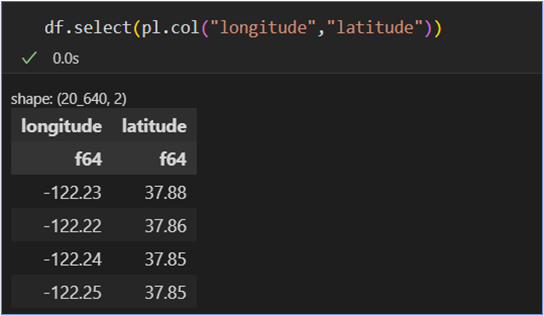
La expresión select nos permite seleccionar las columnas que queramos. Para trabajar con esta expresión simplemente necesitamos dos cosas:

* El DataFrame sobre el que queramos trabajar.
* Los nombres de las columnas a seleccionar.

Por ejemplo, sobre el dataset anterior, si quisiéramos obtener la columna de longitud, haríamos la siguiente consulta:



Cabe destacar que los nombres de las columnas siempre deben pasarse para cualquier consulta en un objeto del tipo Polars.col. Es decir, si quisiéramos sacar tanto la longitud como la latitud, la consulta debería ser la siguiente:



### Expresión Filter

La expresión filter nos permite obtener un subconjunto del DataFrame original, a través de una condición. Para utilizarla, necesitamos 2 cosas:

    1. El DataFrame sobre el que queramos trabajar.

    2. Los filtros que queremos aplicar.

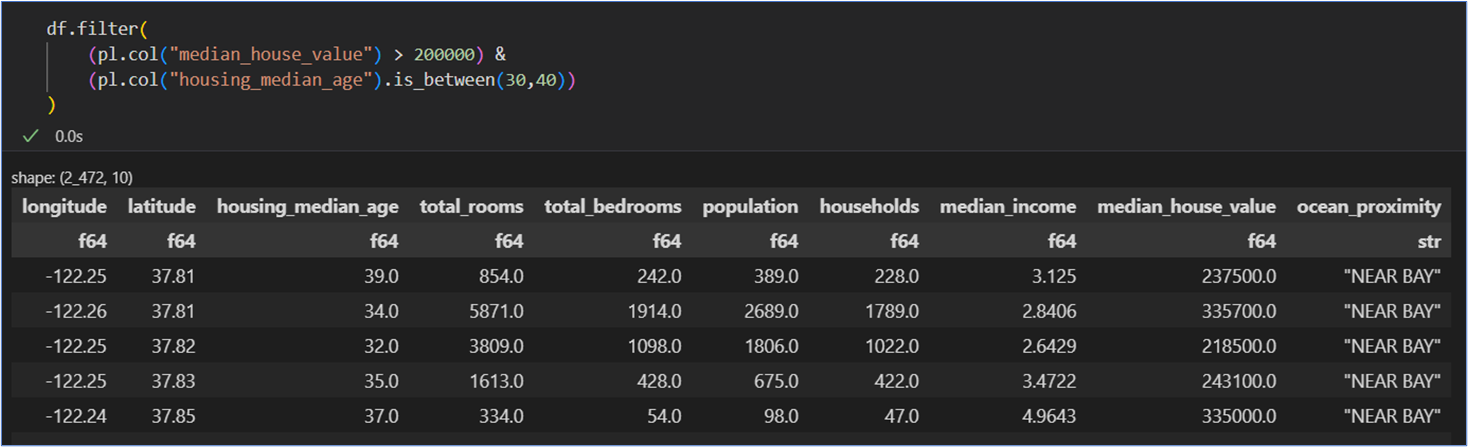
Por ejemplo, si sobre el dataset quisiéramos seleccionar aquellas filas cuya casa tenga un valor superior a los 200.000 dólares, la consulta debería ser la siguiente:



Polars cuenta también con operadores de filtrado básicos que nos permiten simplificar las consultas. Por ejemplo, el operador is\_between nos permite consultar las filas de las casas cuya edad mediana de las personas que viven en ella esté entre los 30 y 40 años.



Como se puede suponer, los filtros se pueden combinar para crear consultas más complejas.



### Expresión with columns

La expresión with\_columns nos permite añadir columnas a nuestro dataset en base a operaciones. Por ejemplo, si quisiéramos calcular cuantas habitaciones hay por persona en una determinada zona, nuestra expresión tendría la siguiente forma:

Como se puede ver, para añadir una columna el parámetro que le pasamos a la expresión with\_columns tiene dos partes:

(Operación).alias(Nombre de la columna resultado)

Al igual que ocurre con las anteriores expresiones, con la expresión with\_columns podemos calcular también varias columnas a la misma vez. Si, por ejemplo, quisiéramos hacer el cálculo anterior y además calcular cuantas habitaciones para dormir hay por persona, la expresión tendría la siguiente forma:

### Expresión group by

La expresión group\_by nos permite crear nuevos DataFrames mediante cálculos por grupos. Por ejemplo, en nuestro conjunto de datos contamos con una variable llamada *ocean\_proximity*, la cual es una tira de caracteres que nos indica la proximidad al mar. Esta variable tiene sólo 5 valores posibles, es decir, se trata de una variable categórica. Si quisiéramos calcular por ejemplo los valores medios para cada variable, agrupando por zona, nuestro código sería el siguiente:

Como se observa, si combinamos la operación de group\_by con el uso de una métrica. Obtenemos el valor para todas las variables del conjunto de datos.

Sin embargo, si lo que buscamos es obtener diversas métricas, como por ejemplo el mínimo, máximo y media, debemos hacerlo a nivel de variable y mediante el uso del operador agg.

### Combinación De Expresiones

Ahora que ya conocemos las expresiones básicas de *Polars*, cabe destacar que estas expresiones pueden combinarse entre ellas, mediante concatenación de puntos, para realizar consultas más complejas. Por ejemplo, podría darse el caso de que creemos nuevas columnas utilizando la expresión with\_columns, pero nuestro dataset tenga demasiadas columnas y no podamos comprobar los cálculos a simple vista. En este caso, una solución sería combinar la expresión with\_columns con una expresión select como se puede ver en la siguiente captura:

Otra posible consulta, sería agrupar las muestras por su etiqueta de proximidad al mar y obtener aquellas zonas cuya edad media sea superior a 30 años. En este caso, nuestra consulta tendría la siguiente forma:

### Traducción De Expresiones Pandas A Polars

Ahora que ya conocemos las expresiones de *Polars* y cómo emplearlas, desarrollaremos un pequeño diccionario de cómo realizar algunas de las consultas más tradicionales de *Pandas* en *Polars.*

### Selección De Columnas

### Selección De Filas Por Índice

Para seleccionar filas por índice tenemos el inconveniente de que en Polars los DataFrames no poseen índice. Sin embargo, crearse una columna auxiliar que realice la función del índice para después trabajar con ella es muy sencillo, el código tendría la siguiente forma:

df.insert\_column(pos,pl.Series(nombre,valores))

pos: posición en la que queremos insertar la columna (0 si queremos que esté al principio)  
nombre: nombre que le queremos dar a la columna  
valores: valores que toman las observaciones  
De esta forma, crear nuestro índice sería tan sencillo como:

df.insert\_column(0,pl.Series(“índice”,range(df.shape[0])))

1. Selección De Filas Por Filtro

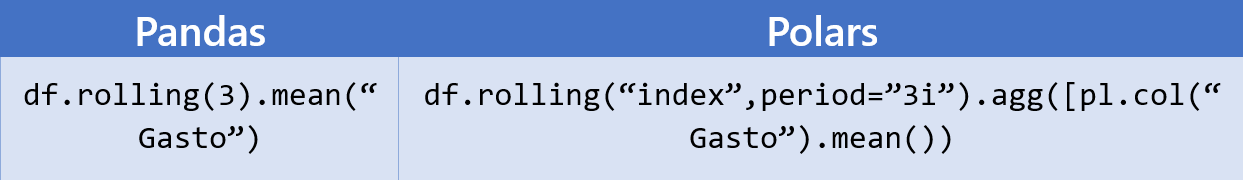
En este caso, utilizamos la expresión filter que hemos visto en Polars.

1. Creación De Columnas
2. Unión De DataFrames
3. Ordenar DataFrames
4. Más Funciones Interesantes

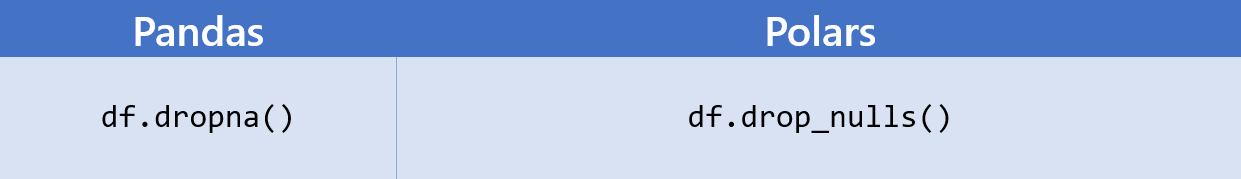
Para escoger las n filas con menor valor en una variable (en este caso 5):

Para rellenar los valores faltantes de una tabla con un determinado valor (en este caso 0):

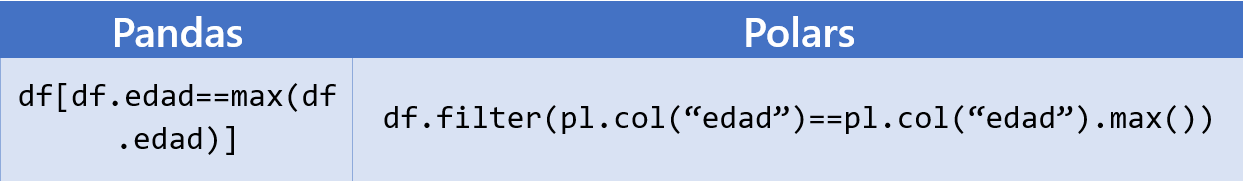
Para calcular la media móvil de una serie temporal usando n periodos (en este caso 3):



Para eliminar las filas con valores faltantes:



Para quedarnos con la/s fila/s que contengan el valor máximo para una determinada variable:



### Ventajas de Polars.

* ✅ Velocidad → mucho más rápido que pandas, especialmente con grandes datasets.
* ✅ Soporte a lazy evaluation → permite optimización automática de consultas.
* ✅ Paralelización → aprovecha todos los núcleos del CPU de forma nativa.
* ✅ Bajo consumo de memoria → gracias al uso de Apache Arrow (columnar).
* ✅ Escalabilidad → puede manejar datasets más grandes que la memoria disponible.
* ✅ Tipos de datos más seguros → menor riesgo de errores silenciosos en conversiones.

### Desventajas de Polars

* ⚠️ Comunidad más pequeña → pandas tiene muchos más usuarios y soporte.
* ⚠️ Menos tutoriales y recursos → aunque está creciendo rápido.
* ⚠️ Compatibilidad limitada → algunas librerías de Python esperan pandas (ej. scikit-learn, statsmodels).
* ⚠️ Curva de aprendizaje → si vienes de pandas, hay que acostumbrarse a la API de Polars.
* ⚠️ Funciones estadísticas avanzadas → pandas todavía tiene más soporte nativo (aunque Polars mejora día a día).

**Cuadro Comparativo**

| **Característica** | **Polars 🦀** | **Pandas 🐼** |
| --- | --- | --- |
| **Lenguaje base** | *Rust + Python* | *Python (Cython en partes)* |
| **Velocidad** | *Muy alta 🚀 (multihilo, optimizado)* | *Media (monohilo)* |
| **Lazy evaluation** | *✅ Sí* | *❌ No* |
| **Paralelización** | *✅ Nativa (usa todos los núcleos)* | *❌ No* |
| **Consumo de memoria** | *Bajo (columnares con Apache Arrow)* | *Alto (basado en arrays de NumPy)* |
| **Facilidad de uso** | *Media (API parecida a pandas, pero diferente)* | *Alta (muy popular y documentada)* |
| **Comunidad y ecosistema** | *Creciente, más pequeño* | *Muy grande, estándar en la industria* |
| **Compatibilidad** | *Limitada con otras librerías* | *Excelente, ampliamente soportado* |
| **Escalabilidad** | *Muy buena (datasets grandes, out-of-core)* | *Limitada (datasets deben caber en RAM)* |
| **Casos de uso** | *Big Data en una sola máquina, ETL, análisis rápido* | *Análisis estándar, prototipado, ML con scikit-learn* |
| **Curva de aprendizaje** | *Media (nueva API, pero intuitiva)* | *Baja (muy difundido y con muchos tutoriales)* |

**Conclusión:**

Si trabajas con datasets pequeños/medianos y quieres facilidad → pandas 🐼.

Si necesitas velocidad, datasets grandes o paralelización → Polars 🦀.

## Searborn.

## Machine Learning