## Abrir archivos con pandas

|  |  |  |
| --- | --- | --- |
| nombre | edad | ciudad |
| Juan | 28 | Madrid |
| Ana | 22 | Barcelona |

para saltar tabulaciones agrega el parametro delimiter al pd.read asi:  
 pd.read\_csv(source, delimiter='\t')

Para archivos con comentarios con el signo #,   
 se debe saltar las filas con el parametro skiprows = n

|  |  |  |
| --- | --- | --- |
| nombre | edad | ciudad |
| Juan | 28 | Madrid |
| Ana | 22 | Barcelona |
| Luis | 35 | Valencia |

Dependiendo el caso agregar el parametro: engine='python'

## Manipular archivos excel

para manipular archivos excel importar openpyxl

Si quieres acceder a todas las hojas de un documento excel, se puede hacer utilizando pd.ExcelFile

Hoja1 Hoja2 Hoja3

si quieres imprimir todo el contenido de cada hoja se utiliza un bucle

Este es el resultado:

Contenido de la hoja: Hoja1:

|  |  |  |
| --- | --- | --- |
| Nombre | Edad | Ciudad |
| Juan | 28 | Madrid |
| Ana | 22 | Barcelona |
| Luis | 35 | Valencia |

Contenido de la hoja: Hoja2:

|  |  |  |
| --- | --- | --- |
| Producto | Cantidad | Precio |
| Manzanas | 10 | 1.2 |
| Naranjas | 15 | 1.5 |
| Plátanos | 7 | 0.8 |

Contenido de la hoja: Hoja3:

|  |  |  |
| --- | --- | --- |
| Fecha | Ventas | Beneficio |
| 2023-01-01 | 100 | 20 |
| 2023-02-01 | 150 | 30 |
| 2023-03-01 | 200 | 50 |

Para leer un grupo de columnas en especifico se utiliza la propiedad usecols, donde se especifica de que columna a que columna, ej: usecols='A:B'

Para guardar un dataframe en un excel se convierte los datos en un dataframe y se utiliza la propiedad .to\_excel, donde la propiendad index se deja en false si no se quiere agregar los numeros a las filas

## Descripcion preliminar de los datos

Para tener una descripcion de los datos utlizar el metodo describe de pandas

edad  
count 14.000000  
mean 25.000000  
std 3.113247  
min 22.000000  
25% 22.000000  
50% 25.000000  
75% 28.000000  
max 28.000000

Si quiero saber los tipos utilizo la propiedad dtypes:

nombre object  
edad int64  
ciudad object

Si quero obtener los valores unicos de una columna utilizo el metodo unique, e  
 specificando la columna, ejemplo: df["nombre"].unique():

['Juan' 'Ana' 'Pablo' 'Carlos' 'Maria' 'Carla' 'Marcos' 'Jhon' 'Camila'  
 'Enzo' 'Julian' 'Tita' 'Pepe']

Si quiero contarlos utilizo el metodo value\_counts:

nombre  
Juan 2  
Ana 1  
Pablo 1  
Carlos 1  
Maria 1  
Carla 1  
Marcos 1  
Jhon 1  
Camila 1  
Enzo 1  
Julian 1  
Tita 1  
Pepe 1

## Manejo de datos faltantes

Para contar la cantidad de datos faltantes por columna   
 utilizar los metodos .isnull().sum():

ID 0  
Nombre 1  
Edad 2  
Ciudad 1  
Salario 2

Para eliminar los datos faltantes se utiliza el metodo .dropna()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario |
| 1 | Juan | 30.0 | Madrid | 3000.0 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 |
| 3 | Luis | 35.0 | Valencia | 3500.0 |
| 4 | María | 29.0 | Sevilla | 4000.0 |
| 7 | Pedro | 20.0 | Valencia | 3700.0 |

Si quiero eliminar las columnas con datos faltantes se utiliza el parametro axis=1  
 en el metodo .dropna()

|  |
| --- |
| ID |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

Se puede rellenar los datos faltantes utilizando el metodo fillna()  
 y especificar que deseas rellenar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario |
| 1 | Juan | 30.0 | Madrid | 3000.0 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 |
| 3 | Luis | 35.0 | Valencia | 3500.0 |
| 4 | María | 29.0 | Sevilla | 4000.0 |
| 5 | Carlos | nan | Madrid | 3200.0 |
| 6 | Laura | 30.0 | Madrid | nan |
| 7 | Pedro | 20.0 | Valencia | 3700.0 |
| 8 | Desconocido | 22.0 | Valencia | 3400.0 |
| 9 | Lucía | nan | nan | 3600.0 |
| 10 | Hugo | 25.0 | Sevilla | nan |

Para rellenar datos numericos como el salario se puede rellenar con la media

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario |
| 1 | Juan | 30.0 | Madrid | 3000.0 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 |
| 3 | Luis | 35.0 | Valencia | 3500.0 |
| 4 | María | 29.0 | Sevilla | 4000.0 |
| 5 | Carlos | nan | Madrid | 3200.0 |
| 6 | Laura | 30.0 | Madrid | 3450.0 |
| 7 | Pedro | 20.0 | Valencia | 3700.0 |
| 8 | Desconocido | 22.0 | Valencia | 3400.0 |
| 9 | Lucía | nan | nan | 3600.0 |
| 10 | Hugo | 25.0 | Sevilla | 3450.0 |

Para datos numericos como la edad, es mas util la mediana:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario |
| 1 | Juan | 30.0 | Madrid | 3000.0 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 |
| 3 | Luis | 35.0 | Valencia | 3500.0 |
| 4 | María | 29.0 | Sevilla | 4000.0 |
| 5 | Carlos | 27.0 | Madrid | 3200.0 |
| 6 | Laura | 30.0 | Madrid | 3450.0 |
| 7 | Pedro | 20.0 | Valencia | 3700.0 |
| 8 | Desconocido | 22.0 | Valencia | 3400.0 |
| 9 | Lucía | 27.0 | nan | 3600.0 |
| 10 | Hugo | 25.0 | Sevilla | 3450.0 |

Para rellenar datos como la cuidad, seria util el uso de la moda

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario |
| 1 | Juan | 30.0 | Madrid | 3000.0 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 |
| 3 | Luis | 35.0 | Valencia | 3500.0 |
| 4 | María | 29.0 | Sevilla | 4000.0 |
| 5 | Carlos | 27.0 | Madrid | 3200.0 |
| 6 | Laura | 30.0 | Madrid | 3450.0 |
| 7 | Pedro | 20.0 | Valencia | 3700.0 |
| 8 | Desconocido | 22.0 | Valencia | 3400.0 |
| 9 | Lucía | 27.0 | Madrid | 3600.0 |
| 10 | Hugo | 25.0 | Sevilla | 3450.0 |

Para especificar los datos en donde se deban eliminar las columnas  
 se utiliza la propiedad subset, ejemplo de subset= [Nombre]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario |
| 1 | Juan | 30.0 | Madrid | 3000.0 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 |
| 3 | Luis | 35.0 | Valencia | 3500.0 |
| 4 | María | 29.0 | Sevilla | 4000.0 |
| 5 | Carlos | nan | Madrid | 3200.0 |
| 6 | Laura | 30.0 | Madrid | nan |
| 7 | Pedro | 20.0 | Valencia | 3700.0 |
| 9 | Lucía | nan | nan | 3600.0 |
| 10 | Hugo | 25.0 | Sevilla | nan |

## Reestructurar datos

Para segmentar y clasificar datos continuos en grupos o intevalos se utiliza pd.cut():

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Ciudad | Salario | Rango\_Edad |
| 1 | Juan | 30.0 | Madrid | 3000.0 | 25-30 |
| 2 | Ana | 22.0 | Barcelona | 3200.0 | 20-25 |
| 3 | Luis | 35.0 | Valencia | 3500.0 | 30-35 |
| 4 | María | 29.0 | Sevilla | 4000.0 | 25-30 |
| 5 | Carlos | nan | Madrid | 3200.0 | nan |
| 6 | Laura | 30.0 | Madrid | nan | 25-30 |
| 7 | Pedro | 20.0 | Valencia | 3700.0 | 20-25 |
| 8 | nan | 22.0 | Valencia | 3400.0 | 20-25 |
| 9 | Lucía | nan | nan | 3600.0 | nan |
| 10 | Hugo | 25.0 | Sevilla | nan | 20-25 |

Para obtener el salario promedio por edad utilizo groupby()

Rango\_Edad  
20-25 3433.333333  
25-30 3500.000000  
30-35 3500.000000

## Manejo de duplicados

Al usar el metodo .duplicated() me indica por medio de booleans que filas son duplicadas, sin necesidad de que estos sean consecutivos

0 False  
1 False  
2 False  
3 False  
4 False  
5 True  
6 False  
7 False  
8 False  
9 False  
10 True

Se puede usar el parametro subset para especificar las columnas y saber si contiene duplicados

0 False  
1 False  
2 False  
3 False  
4 False  
5 True  
6 False  
7 False  
8 False  
9 True  
10 True

Obteniendo las filas de los duplicados con: duplicate[duplicate.duplicated()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso |
| 3 | Carlos | 30 | 58000 | 2020-03-25 |
| 2 | Luis | 35 | 60000 | 2019-06-15 |

Para eliminar los duplicados se utiliza el metodo .drop\_duplicates()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso |
| 1 | Ana | 23 | 50000 | 2020-01-10 |
| 2 | Luis | 35 | 60000 | 2019-06-15 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 |
| 4 | María | 22 | 58000 | 2018-07-30 |
| 5 | Elena | 28 | 49000 | 2018-07-30 |
| 7 | Sofía | 34 | 72000 | 2016-11-22 |
| 8 | Andrés | 45 | 61000 | 2021-04-01 |
| 9 | Laura | 29 | 52000 | 2015-05-19 |
| 10 | Ana | 41 | 67000 | 2014-08-25 |

Para crear una columna que indique los valores duplicados se hace de este metodo: duplicate['Es duplicado?'] = duplicate.duplicated()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso | Es duplicado? |
| 1 | Ana | 23 | 50000 | 2020-01-10 | False |
| 2 | Luis | 35 | 60000 | 2019-06-15 | False |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | False |
| 4 | María | 22 | 58000 | 2018-07-30 | False |
| 5 | Elena | 28 | 49000 | 2018-07-30 | False |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | True |
| 7 | Sofía | 34 | 72000 | 2016-11-22 | False |
| 8 | Andrés | 45 | 61000 | 2021-04-01 | False |
| 9 | Laura | 29 | 52000 | 2015-05-19 | False |
| 10 | Ana | 41 | 67000 | 2014-08-25 | False |
| 2 | Luis | 35 | 60000 | 2019-06-15 | True |

Usando .map puedo cambiar los valores de la columna 'Es duplicado' de un booleano a Si 0 No

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso | Es duplicado? |
| 1 | Ana | 23 | 50000 | 2020-01-10 | No |
| 2 | Luis | 35 | 60000 | 2019-06-15 | No |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | No |
| 4 | María | 22 | 58000 | 2018-07-30 | No |
| 5 | Elena | 28 | 49000 | 2018-07-30 | No |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | Si |
| 7 | Sofía | 34 | 72000 | 2016-11-22 | No |
| 8 | Andrés | 45 | 61000 | 2021-04-01 | No |
| 9 | Laura | 29 | 52000 | 2015-05-19 | No |
| 10 | Ana | 41 | 67000 | 2014-08-25 | No |
| 2 | Luis | 35 | 60000 | 2019-06-15 | Si |

Para reordenar las columnas de un dataframe se hace de esta forma:duplicate[['ID','Nombre','Edad', 'Salario','Es duplicado?','Fecha\_Ingreso']]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Es duplicado? | Fecha\_Ingreso |
| 1 | Ana | 23 | 50000 | No | 2020-01-10 |
| 2 | Luis | 35 | 60000 | No | 2019-06-15 |
| 3 | Carlos | 30 | 58000 | No | 2020-03-25 |
| 4 | María | 22 | 58000 | No | 2018-07-30 |
| 5 | Elena | 28 | 49000 | No | 2018-07-30 |
| 3 | Carlos | 30 | 58000 | Si | 2020-03-25 |
| 7 | Sofía | 34 | 72000 | No | 2016-11-22 |
| 8 | Andrés | 45 | 61000 | No | 2021-04-01 |
| 9 | Laura | 29 | 52000 | No | 2015-05-19 |
| 10 | Ana | 41 | 67000 | No | 2014-08-25 |
| 2 | Luis | 35 | 60000 | Si | 2019-06-15 |

Para elegir columnas especificas de un dataframe se puede usa la propiedad .loc de esta manera: duplicate.loc[:,['ID','Nombre']]

|  |  |
| --- | --- |
| ID | Nombre |
| 1 | Ana |
| 2 | Luis |
| 3 | Carlos |
| 4 | María |
| 5 | Elena |
| 3 | Carlos |
| 7 | Sofía |
| 8 | Andrés |
| 9 | Laura |
| 10 | Ana |
| 2 | Luis |

El primer parametro de la propiedad .loc se utiliza para especificar el rango de las filas que se selecionaran, ejemplo de filas del 1 al 9 / 1:9

|  |  |
| --- | --- |
| ID | Nombre |
| 2 | Luis |
| 3 | Carlos |
| 4 | María |
| 5 | Elena |
| 3 | Carlos |
| 7 | Sofía |
| 8 | Andrés |
| 9 | Laura |

Para eliminar una columna, ejemplo la duplicado, se utiliza el metodo ..drop(columns=['Es duplicado?'])

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso |
| 1 | Ana | 23 | 50000 | 2020-01-10 |
| 2 | Luis | 35 | 60000 | 2019-06-15 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 |
| 4 | María | 22 | 58000 | 2018-07-30 |
| 5 | Elena | 28 | 49000 | 2018-07-30 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 |
| 7 | Sofía | 34 | 72000 | 2016-11-22 |
| 8 | Andrés | 45 | 61000 | 2021-04-01 |
| 9 | Laura | 29 | 52000 | 2015-05-19 |
| 10 | Ana | 41 | 67000 | 2014-08-25 |
| 2 | Luis | 35 | 60000 | 2019-06-15 |

Si quiero filtra los salarios superiores a 50,000 se hace de la siguiente manera: duplicate.loc[duplicate['Salario' ]> 50000]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso |
| 2 | Luis | 35 | 60000 | 2019-06-15 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 |
| 4 | María | 22 | 58000 | 2018-07-30 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 |
| 7 | Sofía | 34 | 72000 | 2016-11-22 |
| 8 | Andrés | 45 | 61000 | 2021-04-01 |
| 9 | Laura | 29 | 52000 | 2015-05-19 |
| 10 | Ana | 41 | 67000 | 2014-08-25 |
| 2 | Luis | 35 | 60000 | 2019-06-15 |

Para agregar una columna nueva a un dataframe se hace de esta manera:duplicate['Posición'] agregando los valores que tendra la columna:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso | Posición |
| 1 | Ana | 23 | 50000 | 2020-01-10 | junior |
| 2 | Luis | 35 | 60000 | 2019-06-15 | junior |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | junior |
| 4 | María | 22 | 58000 | 2018-07-30 | junior |
| 5 | Elena | 28 | 49000 | 2018-07-30 | junior |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | junior |
| 7 | Sofía | 34 | 72000 | 2016-11-22 | Senior |
| 8 | Andrés | 45 | 61000 | 2021-04-01 | mid |
| 9 | Laura | 29 | 52000 | 2015-05-19 | junior |
| 10 | Ana | 41 | 67000 | 2014-08-25 | Senior |
| 2 | Luis | 35 | 60000 | 2019-06-15 | junior |

Esta nueva columna se calcula cuanto se le descuenta de afp + ars:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Nombre | Edad | Salario | Fecha\_Ingreso | Posición | AFP + ARS | Salario\_Neto |
| 1 | Ana | 23 | 50000 | 2020-01-10 | junior | 2955.0 | 47045.0 |
| 2 | Luis | 35 | 60000 | 2019-06-15 | junior | 3546.0 | 56454.0 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | junior | 3427.8 | 54572.2 |
| 4 | María | 22 | 58000 | 2018-07-30 | junior | 3427.8 | 54572.2 |
| 5 | Elena | 28 | 49000 | 2018-07-30 | junior | 2895.9 | 46104.1 |
| 3 | Carlos | 30 | 58000 | 2020-03-25 | junior | 3427.8 | 54572.2 |
| 7 | Sofía | 34 | 72000 | 2016-11-22 | Senior | 4255.2 | 67744.8 |
| 8 | Andrés | 45 | 61000 | 2021-04-01 | mid | 3605.1 | 57394.899999999994 |
| 9 | Laura | 29 | 52000 | 2015-05-19 | junior | 3073.2 | 48926.799999999996 |
| 10 | Ana | 41 | 67000 | 2014-08-25 | Senior | 3959.7 | 63040.299999999996 |
| 2 | Luis | 35 | 60000 | 2019-06-15 | junior | 3546.0 | 56454.0 |

## Concatenacion y combinacion

Para concatenar dos o mas dataframes se utiliza el metodo concat de esta manera: pd.concat([df\_lima, df\_bogota]), este es el resultado:

|  |  |  |
| --- | --- | --- |
| Producto | Ventas | Ciudad |
| A | 250 | Lima |
| B | 150 | Lima |
| A | 100 | Bogotá |
| C | 200 | Bogotá |

Al concatenar datframes es recomendable reiniciar indices usando la propiedad: reset\_index(drop=True)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Producto | Ventas | Ciudad | Producto | Inventario |
| A | 250 | Lima | A | 30.0 |
| B | 150 | Lima | B | 45.0 |
| A | 100 | Bogotá | nan | nan |
| C | 200 | Bogotá | nan | nan |

## Formatear tablas de largo a ancho

Para convertir un dataframe de largo a ancho se utiliza el metodo pivot de un dataframe, que te pide tres parametros, index, columns y values

Antes de utilizar el metodo pivot:

|  |  |  |
| --- | --- | --- |
| Producto | Mes | Ventas |
| A | Enero | 100 |
| A | Febrero | 150 |
| B | Enero | 200 |
| B | Febrero | 250 |
| C | Enero | 300 |
| C | Febrero | 350 |

Despues de utilizar el metodo pivot:

Mes Enero Febrero  
Producto   
A 100 150  
B 200 250  
C 300 350

## Formatear tablas de ancho a largo

Para formatear dataframes, para pasarlos de ancho a largo se utiliza el metodo melt, el cual te pide un dataframe y que llenes los parametros: id\_vars,value\_vars,var\_name, value\_name, el resultado es este:

|  |  |  |
| --- | --- | --- |
| Producto | Mes | Ventas |
| A | Enero | 100 |
| B | Enero | 200 |
| C | Enero | 300 |
| A | Febrero | 150 |
| B | Febrero | 250 |
| C | Febrero | 350 |

## Separar columnas

Para separar los datoscde una tabla de un dataframe en columnas se utiliza el metodo split de la propiedad str: .str.split, donde en el primer parametro se expecifica que elemento se debe separar del texto.

Antes de separar:

|  |
| --- |
| Nombre\_Completo |
| Juan Perez |
| Maria Gomez |
| Luis Martinez |

|  |
| --- |
| Fecha |
| 01-01-2024 |
| 15-02-2024 |
| 30-03-2024 |

Despues de separar:

|  |  |  |
| --- | --- | --- |
| Nombre\_Completo | Nombre | Apellido |
| Juan Perez | Juan | Perez |
| Maria Gomez | Maria | Gomez |
| Luis Martinez | Luis | Martinez |

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | dia | mes | año |
| 01-01-2024 | 01 | 01 | 2024 |
| 15-02-2024 | 15 | 02 | 2024 |
| 30-03-2024 | 30 | 03 | 2024 |

Para crear una fecha completa con las / se usa el metodo agg, y el metodo join de esta manera: .agg('/'.join,axis=1)

Resultado:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fecha | dia | mes | año | Fecha\_completa |
| 01-01-2024 | 01 | 01 | 2024 | 01/01/2024 |
| 15-02-2024 | 15 | 02 | 2024 | 15/02/2024 |
| 30-03-2024 | 30 | 03 | 2024 | 30/03/2024 |

## Conversion de datos categoricos

Para la conversion de datos categoricos se puede utilizar el metodo de codificacion ordinal que consiste en darle un valor a las variables categoricas para poder darles un orden lógico o jerárquico

Datatframe si en el orden categorico:

|  |  |  |  |
| --- | --- | --- | --- |
| producto | categoria | calidad | ventas |
| Manzana | Fruta | Alta | 50 |
| Banana | Fruta | Media | 30 |
| Cereza | Fruta | Baja | 70 |
| Durazno | Fruta | Alta | 85 |
| Pera | Fruta | Media | 40 |

Despues:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| producto | categoria | calidad | ventas | Categoria ordinal |
| Manzana | Fruta | Alta | 50 | 3 |
| Banana | Fruta | Media | 30 | 2 |
| Cereza | Fruta | Baja | 70 | 1 |
| Durazno | Fruta | Alta | 85 | 3 |
| Pera | Fruta | Media | 40 | 2 |

La codificacion one hot permite convertir cada columna categoria a binaria:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| producto | categoria | ventas | Categoria ordinal | calidad\_Alta | calidad\_Baja | calidad\_Media |
| Manzana | Fruta | 50 | 3 | True | False | False |
| Banana | Fruta | 30 | 2 | False | False | True |
| Cereza | Fruta | 70 | 1 | False | True | False |
| Durazno | Fruta | 85 | 3 | True | False | False |
| Pera | Fruta | 40 | 2 | False | False | True |

## Variables dummy

Las variables dummy son variables categoricas que toman valores binarios, usando el metodo get\_dummies se crean columnas binarias con estos valores

Antes de la conversión:

|  |  |  |  |
| --- | --- | --- | --- |
| vehiculo | color | precio | ventas |
| Auto | Rojo | 20000 | 150 |
| Camioneta | Azul | 30000 | 120 |
| Moto | Negro | 15000 | 130 |
| Camion | Blanco | 40000 | 60 |
| Auto | Rojo | 18000 | 180 |

Despues:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| precio | ventas | vehiculo\_Auto | vehiculo\_Camion | vehiculo\_Camioneta | vehiculo\_Moto | color\_Azul | color\_Blanco | color\_Negro | color\_Rojo |
| 20000 | 150 | True | False | False | False | False | False | False | True |
| 30000 | 120 | False | False | True | False | True | False | False | False |
| 15000 | 130 | False | False | False | True | False | False | True | False |
| 40000 | 60 | False | True | False | False | False | True | False | False |
| 18000 | 180 | True | False | False | False | False | False | False | True |

Usando el parametro drop\_first, se elimina la multicolinealidad que consiste en eliminar las columnas que son redundates

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| precio | ventas | vehiculo\_Camion | vehiculo\_Camioneta | vehiculo\_Moto | color\_Blanco | color\_Negro | color\_Rojo |
| 20000 | 150 | False | False | False | False | False | True |
| 30000 | 120 | False | True | False | False | False | False |
| 15000 | 130 | False | False | True | False | True | False |
| 40000 | 60 | True | False | False | True | False | False |
| 18000 | 180 | False | False | False | False | False | True |

Si quiero que los valores de las columnas dummy se muestren con texto descriptivos, se puede usar el metodo map, pero usando una expresion lambda en vez de booleanos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vehiculo\_Camion | vehiculo\_Camioneta | vehiculo\_Moto | color\_Blanco | color\_Negro | color\_Rojo |
| No | No | No | No | No | Si |
| No | Si | No | No | No | No |
| No | No | Si | No | Si | No |
| Si | No | No | Si | No | No |
| No | No | No | No | No | Si |

## Funciones de agregación

Para las funciones de agregacion usando el metodo groupby se debe especificar la columna categorica, los valores los cuales seran agrupados y el calculo que se realizara

Datos sin agrupar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| producto | categoría | ventas | precio\_unitario | fecha |
| Manzana | Fruta | 50 | 0.5 | 2024-01-01 00:00:00 |
| Manzana | Fruta | 30 | 0.5 | 2024-01-02 00:00:00 |
| Banana | Fruta | 20 | 0.3 | 2024-01-03 00:00:00 |
| Banana | Legumbre | 15 | 0.3 | 2024-01-04 00:00:00 |
| Cereza | Fruta | 10 | 0.8 | 2024-01-05 00:00:00 |
| Cereza | Fruta | 25 | 0.8 | 2024-01-06 00:00:00 |
| Manzana | Fruta | 45 | 0.5 | 2024-01-07 00:00:00 |
| Banana | Legumbre | 55 | 0.3 | 2024-01-08 00:00:00 |
| Cereza | Fruta | 40 | 0.8 | 2024-01-09 00:00:00 |

Datos de suma de ventas agrupados por producto, df.groupby('producto')['ventas'].sum():

producto  
Banana 90  
Cereza 75  
Manzana 125

Datos del promedio de precio por unidad agrupados por producto, df.groupby('producto')['precio\_unitario'].mean()

producto  
Banana 0.3  
Cereza 0.8  
Manzana 0.5

Con el metodo agg se puede agregar multiples funciones, ejemplo:

|  |  |  |
| --- | --- | --- |
| sum | min | max |
| 90 | 15 | 55 |
| 75 | 10 | 40 |
| 125 | 30 | 50 |

Si quiero cambiar los datos agrupados convertidos en una serie a un dataframe se utiliza el metodo .reset\_index()

|  |  |  |  |
| --- | --- | --- | --- |
| producto | sum | min | max |
| Banana | 90 | 15 | 55 |
| Cereza | 75 | 10 | 40 |
| Manzana | 125 | 30 | 50 |

Se puede agrupar varias columnas con groupby y crear varias funciones con el metodo agg:

|  |  |
| --- | --- |
| total\_sales | average\_price |
| 20.0 | 0.3 |
| 75.0 | 0.8000000000000002 |
| 125.0 | 0.5 |
| 70.0 | 0.3 |

## Funciones personalizadas

Usan el metodo apply, se puede integrar funciones personalizadas

Dataframe original:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| producto | categoría | ventas | precio\_unitario | fecha |
| Manzana | Fruta | 50 | 0.5 | 2024-01-01 00:00:00 |
| Manzana | Fruta | 30 | 0.5 | 2024-01-02 00:00:00 |
| Banana | Fruta | 20 | 0.3 | 2024-01-03 00:00:00 |
| Banana | Fruta | 15 | 0.3 | 2024-01-04 00:00:00 |
| Cereza | Fruta | 10 | 0.8 | 2024-01-05 00:00:00 |
| Cereza | Fruta | 25 | 0.8 | 2024-01-06 00:00:00 |
| Manzana | Fruta | 45 | 0.5 | 2024-01-07 00:00:00 |
| Banana | Fruta | 55 | 0.3 | 2024-01-08 00:00:00 |
| Cereza | Fruta | 40 | 0.8 | 2024-01-09 00:00:00 |

dataframe con los ingresos totales:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| producto | categoría | ventas | precio\_unitario | fecha | Ingresos totales |
| Manzana | Fruta | 50 | 0.5 | 2024-01-01 00:00:00 | 25.0 |
| Manzana | Fruta | 30 | 0.5 | 2024-01-02 00:00:00 | 15.0 |
| Banana | Fruta | 20 | 0.3 | 2024-01-03 00:00:00 | 6.0 |
| Banana | Fruta | 15 | 0.3 | 2024-01-04 00:00:00 | 4.5 |
| Cereza | Fruta | 10 | 0.8 | 2024-01-05 00:00:00 | 8.0 |
| Cereza | Fruta | 25 | 0.8 | 2024-01-06 00:00:00 | 20.0 |
| Manzana | Fruta | 45 | 0.5 | 2024-01-07 00:00:00 | 22.5 |
| Banana | Fruta | 55 | 0.3 | 2024-01-08 00:00:00 | 16.5 |
| Cereza | Fruta | 40 | 0.8 | 2024-01-09 00:00:00 | 32.0 |

dataframe con la clasificacion de las ventas:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| producto | categoría | ventas | precio\_unitario | fecha | Ingresos totales | clasificaciones |
| Manzana | Fruta | 50 | 0.5 | 2024-01-01 00:00:00 | 25.0 | Alto |
| Manzana | Fruta | 30 | 0.5 | 2024-01-02 00:00:00 | 15.0 | Medio |
| Banana | Fruta | 20 | 0.3 | 2024-01-03 00:00:00 | 6.0 | Medio |
| Banana | Fruta | 15 | 0.3 | 2024-01-04 00:00:00 | 4.5 | Bajo |
| Cereza | Fruta | 10 | 0.8 | 2024-01-05 00:00:00 | 8.0 | Bajo |
| Cereza | Fruta | 25 | 0.8 | 2024-01-06 00:00:00 | 20.0 | Medio |
| Manzana | Fruta | 45 | 0.5 | 2024-01-07 00:00:00 | 22.5 | Alto |
| Banana | Fruta | 55 | 0.3 | 2024-01-08 00:00:00 | 16.5 | Alto |
| Cereza | Fruta | 40 | 0.8 | 2024-01-09 00:00:00 | 32.0 | Medio |

dataframe con el porcentaje de las ventas:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| producto | categoría | ventas | precio\_unitario | fecha | Ingresos totales | clasificaciones | Porcentaje |
| Manzana | Fruta | 50 | 0.5 | 2024-01-01 00:00:00 | 25.0 | Alto | 40 |
| Manzana | Fruta | 30 | 0.5 | 2024-01-02 00:00:00 | 15.0 | Medio | 24 |
| Banana | Fruta | 20 | 0.3 | 2024-01-03 00:00:00 | 6.0 | Medio | 22 |
| Banana | Fruta | 15 | 0.3 | 2024-01-04 00:00:00 | 4.5 | Bajo | 16 |
| Cereza | Fruta | 10 | 0.8 | 2024-01-05 00:00:00 | 8.0 | Bajo | 13 |
| Cereza | Fruta | 25 | 0.8 | 2024-01-06 00:00:00 | 20.0 | Medio | 33 |
| Manzana | Fruta | 45 | 0.5 | 2024-01-07 00:00:00 | 22.5 | Alto | 36 |
| Banana | Fruta | 55 | 0.3 | 2024-01-08 00:00:00 | 16.5 | Alto | 61 |
| Cereza | Fruta | 40 | 0.8 | 2024-01-09 00:00:00 | 32.0 | Medio | 53 |

## Funciones de grupo

Estas funciones por grupo esta hecho con los metodos groupby y agg

Dataframe original:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| producto | categoría | ventas | precio\_unitario | fecha |
| Manzana | Fruta | 50 | 0.5 | 2024-01-01 00:00:00 |
| Manzana | Fruta | 30 | 0.5 | 2024-01-02 00:00:00 |
| Banana | Fruta | 20 | 0.3 | 2024-01-03 00:00:00 |
| Banana | Fruta | 15 | 0.3 | 2024-01-04 00:00:00 |
| Cereza | Fruta | 10 | 0.8 | 2024-01-05 00:00:00 |
| Cereza | Fruta | 25 | 0.8 | 2024-01-06 00:00:00 |
| Manzana | Fruta | 45 | 0.5 | 2024-01-07 00:00:00 |
| Banana | Fruta | 55 | 0.3 | 2024-01-08 00:00:00 |
| Cereza | Fruta | 40 | 0.8 | 2024-01-09 00:00:00 |

Agrupación de productos por ventas:

|  |  |  |
| --- | --- | --- |
| producto | sum | mean |
| Banana | 90 | 30.0 |
| Cereza | 75 | 25.0 |
| Manzana | 125 | 41.666666666666664 |

Rango de ventas, (hecho con una función personalizada):

|  |  |
| --- | --- |
| producto | ventas |
| Banana | 40 |
| Cereza | 30 |
| Manzana | 20 |

Agrupación de ingreso total por producto:

|  |  |  |  |
| --- | --- | --- | --- |
| producto | categoría | ventas | ingreso\_total |
| Banana | Fruta | 90 | 27.0 |
| Cereza | Fruta | 75 | 60.0 |
| Manzana | Fruta | 125 | 62.5 |

## Exportación de datos

Para exportar datos se debe primero convertir el conjunto de datos en un dataframe, despues se elige el formato para exportar, de los cuales estan:  
 to\_csv  
 to\_excel  
 to\_parquet  
 to\_html  
 etc.

Al elegir cualquiera de los formatos para exportar se debe ingresar el nombre que tendra el archivo y su extensión de esta manera: to\_excel('Output/documento excel.xlsx',index=True) donde la propiedad index indica si deberia estar numerada cada fila del dataframe.

Recomendaciones para el guardado de archivos:  
 En archivos csv, el parametro sep es para separar los delimitadores.  
 float\_format='%.2f' Para expecificar que solo haya 2 decimales  
 El parametro compression es para especificar el formato en el que se comprimirá el archivo  
 el parametro compression permite los siguientes procesos compression={'method': 'zip', 'archive\_name': 'datos\_comprimido.xlsx'}  
 el parametro encoding, permite codificar el archivo a multiples formatos: encoding='utf-8'  
 Con el parametro na\_rep puedes especificar con que rellenar los datos nulos o vacios, ej: na\_rep='N/A'

## Manejo de json

Para pasar cadenas de json al metodo pd.read\_json, es necesario convertir esas cadenas en un formato StringIO de la siguiente forma:

json\_data = '''  
[  
 {"producto": "A", "ventas": 100},  
 {"producto": "B", "ventas": 200},  
 {"producto": "C", "ventas": 300}  
]  
'''  
  
# Envolver la cadena JSON en un objeto StringIO  
json\_stream = StringIO(json\_data)  
  
Es necesario hacer estas importaciones: from io import StringIO

Diferentes orientaciones que se pueden utilizar a exportar en json:

orient='split':

{"columns":["Nombre","Edad","Ciudad"],"index":[0,1,2],"data":[["Ana",23,"Madrid"],["Juan",45,"Barcelona"],["Pedro",34,"Valencia"]]}

orient='records' uso de la propiedad lines = True (unicamente utilizable con esta orientación):

{"Nombre":"Ana","Edad":23,"Ciudad":"Madrid"}  
{"Nombre":"Juan","Edad":45,"Ciudad":"Barcelona"}  
{"Nombre":"Pedro","Edad":34,"Ciudad":"Valencia"}

orient='index'

{"0":{"Nombre":"Ana","Edad":23,"Ciudad":"Madrid"},"1":{"Nombre":"Juan","Edad":45,"Ciudad":"Barcelona"},"2":{"Nombre":"Pedro","Edad":34,"Ciudad":"Valencia"}}

orient='columns:'

{"Nombre":{"0":"Ana","1":"Juan","2":"Pedro"},"Edad":{"0":23,"1":45,"2":34},"Ciudad":{"0":"Madrid","1":"Barcelona","2":"Valencia"}}

orient='values':

[["Ana",23,"Madrid"],["Juan",45,"Barcelona"],["Pedro",34,"Valencia"]]

## Manejo de xml y html

Para la lectura de archivos xml se usa el metodo read\_xml, si no es un archivo, se usa StringIO de la libreria io al igual que los json

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | author | title | genre | price | publish\_date |
| bk101 | Gambardella, Matthew | XML Developer's Guide | Computer | 44.95 | 2000-10-01 |
| bk102 | Ralls, Kim | Midnight Rain | Fantasy | 5.95 | 2000-12-16 |
| bk103 | Corets, Eva | Maeve Ascendant | Fantasy | 5.95 | 2000-11-17 |

Si un documento html posee varias tablas usando tables[n] donde n es el numero de tabla que se busca acceder:

|  |  |  |
| --- | --- | --- |
| Producto | CategorÃ­a | Precio |
| Ordenador PortÃ¡til | ElectrÃ³nica | 800.0 |
| Mesa de Comedor | Muebles | 300.0 |
| Silla de Oficina | Muebles | 120.0 |

## Uso de sql

Para crear un base de datos de sqlite se usa el metodo create\_engine de sqlalchemy, el parametro que exige es este: sqlite:///Output/nombre\_de\_la\_basededatos.db, (Las primeras 3 diagonales son obligatorias cuando no hay un nombre de usuario, contraseña o demas parametros)

Con el metodo to\_sql de un dataframe puedes hacer que se guarden los datos en una tabla de una base de datos.

Parametros:  
 name: (obligatorio), Especifica el nombre de la tabla en la base de datos donde se guardará el DataFrame.  
 con: (obligatorio), La conexión a la base de datos, creada previamente con SQLAlchemy (usando create\_engine).  
 if\_exists: Especifica que hacer si la tabla ya existe.  
 index: Pametro booleano para indicar si se debe guardar el indice en la base de datos

Con el metodo read\_sql se puede pasar una query para realizar una operación en db, toma como parametro la conexión 'con'

## Uso de orm

La libreria de sqlalchemy posee multiples clases para mapear clases de python en tablas para una base de datos, entre ellas estan Column, que representaran las columnas de las tablas

De sqlalchemy.orm estan:  
 declarative\_base: Clase base para definir modelos (tablas).  
 sessionmaker: Crea sesiones para interactuar con la base de datos.

Al instanciar Base = declarative\_base() Base se convierte en una clase la cual servira para definir las clases que seran mapeadas en tablas para las base de datos.

Con Session se realiza lo siguiente:   
 Session = sessionmaker(bind=engine)  
 session = Session()  
   
 sessionmaker: Genera una fábrica de sesiones ligadas al motor (base de datos).  
 session: Es una instancia de sesión que se usa para interactuar con la base de datos.

Con: Base.metadata.create\_all(engine) Crea las tablas en la base de datos basadas en los modelos definidos (si no existen).

Con: session.add(empleado) y session.commit() ocurre lo siguiente:  
 session.add(empleado): Añade la instancia al contexto de la sesión.  
 session.commit(): Confirma los cambios y guarda el registro en la base de datos.

## Manejo de fechas y tiempo

El metodo date\_range de pandas permite generar una serie de fechas apartir de una fecha inicial, sus parametros son:  
 start: Recibe un valor de fecha o de tipo string que sea una fecha, sirve como fecha inicial.  
   
 periods: Es un valor de tipo integer que especifica la cantidad de fechas o puntos de tiempo se deben generar en el rango  
 freq: Especifica la frecuencia de los intervalos entre las fechas generadas. Por defecto es 'D' (diaria).

Una serie es una estructura de datos unidimensional. se pueden crear series con el metodo pd.Series de pandas, posee como parametros data que recibe la información de la serie e index, que pueden contener los idices de los datos.

2023-01-01 150  
2023-01-02 152  
2023-01-03 149  
2023-01-04 153  
2023-01-05 155  
2023-01-06 158  
2023-01-07 157  
2023-01-08 160  
2023-01-09 162  
2023-01-10 161  
Freq: D

Con la propiedad .iloc[n] se puede obtener el elemento de una serie basado en su posición numerica, si se usa un slicing solo traera los datos de la primera posición especificada hasta la anterior a la ultima espcificada, ej: 1:3 donde se seleccionará los elementos en las posiciones 1 y 2 de la Serie.

152

2023-01-02 152  
2023-01-03 149  
Freq: D

Se puede obtener un subconjunto de datos de las series usando slicing ya sea espicificando el indice o la posición.

Por posicion [0:3]

2023-01-01 150  
2023-01-02 152  
2023-01-03 149  
Freq: D

Por indice '2023-01-01':'2023-01-05'

2023-01-01 150  
2023-01-02 152  
2023-01-03 149  
2023-01-04 153  
2023-01-05 155  
Freq: D

Con el metodo diff puedes obtener la diferencia de valores de los datos con respecto a la fecha anterior.

2023-01-01 NaN  
2023-01-02 2.0  
2023-01-03 -3.0  
2023-01-04 4.0  
2023-01-05 2.0  
2023-01-06 3.0  
2023-01-07 -1.0  
2023-01-08 3.0  
2023-01-09 2.0  
2023-01-10 -1.0  
Freq: D

El metodo resample se utiliza para reasignar o agrupar datos de series temporales en diferentes intervalos de tiempo, acepta como parametro el tipo de frecuencia al cual se quiere cambiar ej: (dias,semanas, meses etc)

Suma de series agrupadas en semana:

2023-01-01 150  
2023-01-08 1084  
2023-01-15 323  
Freq: W-SUN

Para el formateo de fechas se usa el metodo pd.to\_datetime función que convierte una columna de tipo texto, enteros u otros formatos a un objeto de fecha y hora (datetime64)., con el parametro format se puede asignar un formato especifico a las fechas.

Fechas del dataframe sin formatear

|  |
| --- |
| fecha |
| 2024-08-01 |
| 01/08/2024 |
| August 1, 2024 |
| 20240801 |

Fechas convertidas con to\_datetime y formateadas con format='mixed' (Formato mixto)

|  |
| --- |
| fecha |
| 2024-08-01 00:00:00 |
| 2024-01-08 00:00:00 |
| 2024-08-01 00:00:00 |
| 2024-08-01 00:00:00 |

con el metodo .dt.strftime('%d-%m-%Y') puedes formatear columnas en formato datetime en columnas string formateadas en una patron especifico

Fecha formateada en '%d-%m-%Y' dia-mes-año:

|  |  |
| --- | --- |
| fecha | fecha\_formateada |
| 2024-08-01 00:00:00 | 01-08-2024 |
| 2024-01-08 00:00:00 | 08-01-2024 |
| 2024-08-01 00:00:00 | 01-08-2024 |
| 2024-08-01 00:00:00 | 01-08-2024 |

El argumento errors='coerce' en el método pd.to\_datetime() se utiliza para manejar errores durante la conversión de datos a formato de fecha y hora (datetime). Si pandas encuentra un valor que no puede ser convertido a una fecha válida, en lugar de generar un error, asignará un valor especial llamado NaT ('Not a Time')

Para la extracción de partes de una fecha (como dia mes o año) se utiliza dt.day|dt.month|dt.year

|  |  |  |  |
| --- | --- | --- | --- |
| fecha | dia | mes | año |
| 2024-08-01 00:00:00 | 1 | 8 | 2024 |
| 2024-01-08 00:00:00 | 8 | 1 | 2024 |
| 2024-08-01 00:00:00 | 1 | 8 | 2024 |
| 2024-08-01 00:00:00 | 1 | 8 | 2024 |

Para la extraccion del tiempo (hora, minuto, segundo) se utiliza dt.hour|dt.minute|dt.second

Fechas con horas:

|  |
| --- |
| fecha\_hora |
| 2024-08-01 14:30:00 |
| 2023-12-15 09:45:30 |
| 2025-07-20 22:10:15 |

|  |  |  |  |
| --- | --- | --- | --- |
| fecha\_hora | hora | minuto | segundo |
| 2024-08-01 14:30:00 | 14 | 30 | 0 |
| 2023-12-15 09:45:30 | 9 | 45 | 30 |
| 2025-07-20 22:10:15 | 22 | 10 | 15 |

El método dt.isocalendar() se utiliza para obtener información relacionada con el calendario ISO para una columna de fechas. El método devuelve un DataFrame con tres columnas: year/week/day:

|  |  |  |
| --- | --- | --- |
| year | week | day |
| 2024 | 31 | 4 |
| 2023 | 50 | 5 |
| 2025 | 29 | 7 |

Si una columna con valores datetime posee datos faltantes se pueden usar los metodos ffill() y bffill(). ffill rellena la información faltante con la fecha de la fila anterior mientras que bffill con la posterior

Con pd.Timestamp puedes convertir una cadena en un objeto de tipo Timestamp (fecha y hora en pandas). Combinándolo con .fillna() puedes remplazar los valores faltantes con la fecha proporcionada

Fecha sin limpiar:

|  |
| --- |
| fecha |
| 2024-08-01 00:00:00 |
| NaT |
| NaT |
| 2023-12-15 00:00:00 |

uso de drona():

fecha  
0 2024-08-01  
3 2023-12-15

Uso de notna() devuelve un booleano que indica si hay valores vacios o no

0 True  
1 False  
2 False  
3 True

Se puede hacer calculos con las fecha como ejemplo restar:

|  |  |  |
| --- | --- | --- |
| fecha\_inicio | fecha\_fin | diferencia\_dias |
| 2024-08-01 00:00:00 | 2024-08-10 00:00:00 | 9 days 00:00:00 |
| 2023-12-15 00:00:00 | 2021-12-25 00:00:00 | -720 days +00:00:00 |
| 2025-07-20 00:00:00 | 2025-08-01 00:00:00 | 12 days 00:00:00 |

Con el pd.DateOffset() se puede sumar o restar períodos de tiempo a fechas en un DataFrame o Series de tipo datetime:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| fecha\_inicio | fecha\_fin | diferencia\_dias | fecha\_inicio\_mas\_6d | fecha\_menos\_1\_mes | fecha\_mas\_2\_años |
| 2024-08-01 00:00:00 | 2024-08-10 00:00:00 | 9 days 00:00:00 | 2024-08-07 00:00:00 | 2024-07-10 00:00:00 | 2026-08-01 00:00:00 |
| 2023-12-15 00:00:00 | 2021-12-25 00:00:00 | -720 days +00:00:00 | 2023-12-21 00:00:00 | 2021-11-25 00:00:00 | 2025-12-15 00:00:00 |
| 2025-07-20 00:00:00 | 2025-08-01 00:00:00 | 12 days 00:00:00 | 2025-07-26 00:00:00 | 2025-07-01 00:00:00 | 2027-07-20 00:00:00 |

Con el metodo resample puedes reagrupar las fecha e integrarle una funcion de agregacion, ejemplo reagrupacion a final de mes resample('M') con promedio de dias

2024-01-31 15.0  
2024-02-29 45.0  
2024-03-31 75.0  
2024-04-30 105.5  
2024-05-31 136.0  
2024-06-30 166.5  
2024-07-31 197.0  
2024-08-31 228.0  
2024-09-30 258.5  
2024-10-31 289.0  
2024-11-30 319.5  
2024-12-31 350.0  
Freq: M

Para realizar promedios moviles se utiliza el metodo rolling(window=7).mean() donde el parametro window especifica de cuanto sera la ventana deslizante de 7 días sobre la serie de datos.

2024-01-01 NaN  
2024-01-02 NaN  
2024-01-03 NaN  
2024-01-04 NaN  
2024-01-05 NaN  
2024-01-06 NaN  
2024-01-07 3.0  
2024-01-08 4.0  
2024-01-09 5.0  
2024-01-10 6.0  
2024-01-11 7.0  
2024-01-12 8.0  
2024-01-13 9.0  
2024-01-14 10.0  
Freq: D