## Abrir archivos con pandas

nombre edad ciudad  
0 Juan 28 Madrid  
1 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  
0 Juan 28 Madrid  
1 Ana 22 Barcelona  
2 Luis 35 Valencia

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

## 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  
0 1 Juan 30.0 Madrid 3000.0  
1 2 Ana 22.0 Barcelona 3200.0  
2 3 Luis 35.0 Valencia 3500.0  
3 4 María 29.0 Sevilla 4000.0  
6 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  
0 1  
1 2  
2 3  
3 4  
4 5  
5 6  
6 7  
7 8  
8 9  
9 10

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

ID Nombre Edad Ciudad Salario  
0 1 Juan 30.0 Madrid 3000.0  
1 2 Ana 22.0 Barcelona 3200.0  
2 3 Luis 35.0 Valencia 3500.0  
3 4 María 29.0 Sevilla 4000.0  
4 5 Carlos NaN Madrid 3200.0  
5 6 Laura 30.0 Madrid NaN  
6 7 Pedro 20.0 Valencia 3700.0  
7 8 Desconocido 22.0 Valencia 3400.0  
8 9 Lucía NaN NaN 3600.0  
9 10 Hugo 25.0 Sevilla NaN

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

ID Nombre Edad Ciudad Salario  
0 1 Juan 30.0 Madrid 3000.0  
1 2 Ana 22.0 Barcelona 3200.0  
2 3 Luis 35.0 Valencia 3500.0  
3 4 María 29.0 Sevilla 4000.0  
4 5 Carlos NaN Madrid 3200.0  
5 6 Laura 30.0 Madrid 3450.0  
6 7 Pedro 20.0 Valencia 3700.0  
7 8 Desconocido 22.0 Valencia 3400.0  
8 9 Lucía NaN NaN 3600.0  
9 10 Hugo 25.0 Sevilla 3450.0

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

ID Nombre Edad Ciudad Salario  
0 1 Juan 30.0 Madrid 3000.0  
1 2 Ana 22.0 Barcelona 3200.0  
2 3 Luis 35.0 Valencia 3500.0  
3 4 María 29.0 Sevilla 4000.0  
4 5 Carlos 27.0 Madrid 3200.0  
5 6 Laura 30.0 Madrid 3450.0  
6 7 Pedro 20.0 Valencia 3700.0  
7 8 Desconocido 22.0 Valencia 3400.0  
8 9 Lucía 27.0 NaN 3600.0  
9 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  
0 1 Juan 30.0 Madrid 3000.0  
1 2 Ana 22.0 Barcelona 3200.0  
2 3 Luis 35.0 Valencia 3500.0  
3 4 María 29.0 Sevilla 4000.0  
4 5 Carlos 27.0 Madrid 3200.0  
5 6 Laura 30.0 Madrid 3450.0  
6 7 Pedro 20.0 Valencia 3700.0  
7 8 Desconocido 22.0 Valencia 3400.0  
8 9 Lucía 27.0 Madrid 3600.0  
9 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  
0 1 Juan 30.0 Madrid 3000.0  
1 2 Ana 22.0 Barcelona 3200.0  
2 3 Luis 35.0 Valencia 3500.0  
3 4 María 29.0 Sevilla 4000.0  
4 5 Carlos NaN Madrid 3200.0  
5 6 Laura 30.0 Madrid NaN  
6 7 Pedro 20.0 Valencia 3700.0  
8 9 Lucía NaN NaN 3600.0  
9 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  
0 1 Juan 30.0 Madrid 3000.0 25-30  
1 2 Ana 22.0 Barcelona 3200.0 20-25  
2 3 Luis 35.0 Valencia 3500.0 30-35  
3 4 María 29.0 Sevilla 4000.0 25-30  
4 5 Carlos NaN Madrid 3200.0 NaN  
5 6 Laura 30.0 Madrid NaN 25-30  
6 7 Pedro 20.0 Valencia 3700.0 20-25  
7 8 NaN 22.0 Valencia 3400.0 20-25  
8 9 Lucía NaN NaN 3600.0 NaN  
9 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  
5 3 Carlos 30 58000 2020-03-25  
10 2 Luis 35 60000 2019-06-15

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

ID Nombre Edad Salario Fecha\_Ingreso  
0 1 Ana 23 50000 2020-01-10  
1 2 Luis 35 60000 2019-06-15  
2 3 Carlos 30 58000 2020-03-25  
3 4 María 22 58000 2018-07-30  
4 5 Elena 28 49000 2018-07-30  
6 7 Sofía 34 72000 2016-11-22  
7 8 Andrés 45 61000 2021-04-01  
8 9 Laura 29 52000 2015-05-19  
9 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?  
0 1 Ana 23 50000 2020-01-10 False  
1 2 Luis 35 60000 2019-06-15 False  
2 3 Carlos 30 58000 2020-03-25 False  
3 4 María 22 58000 2018-07-30 False  
4 5 Elena 28 49000 2018-07-30 False  
5 3 Carlos 30 58000 2020-03-25 True  
6 7 Sofía 34 72000 2016-11-22 False  
7 8 Andrés 45 61000 2021-04-01 False  
8 9 Laura 29 52000 2015-05-19 False  
9 10 Ana 41 67000 2014-08-25 False  
10 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?  
0 1 Ana 23 50000 2020-01-10 No  
1 2 Luis 35 60000 2019-06-15 No  
2 3 Carlos 30 58000 2020-03-25 No  
3 4 María 22 58000 2018-07-30 No  
4 5 Elena 28 49000 2018-07-30 No  
5 3 Carlos 30 58000 2020-03-25 Si  
6 7 Sofía 34 72000 2016-11-22 No  
7 8 Andrés 45 61000 2021-04-01 No  
8 9 Laura 29 52000 2015-05-19 No  
9 10 Ana 41 67000 2014-08-25 No  
10 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  
0 1 Ana 23 50000 No 2020-01-10  
1 2 Luis 35 60000 No 2019-06-15  
2 3 Carlos 30 58000 No 2020-03-25  
3 4 María 22 58000 No 2018-07-30  
4 5 Elena 28 49000 No 2018-07-30  
5 3 Carlos 30 58000 Si 2020-03-25  
6 7 Sofía 34 72000 No 2016-11-22  
7 8 Andrés 45 61000 No 2021-04-01  
8 9 Laura 29 52000 No 2015-05-19  
9 10 Ana 41 67000 No 2014-08-25  
10 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  
0 1 Ana  
1 2 Luis  
2 3 Carlos  
3 4 María  
4 5 Elena  
5 3 Carlos  
6 7 Sofía  
7 8 Andrés  
8 9 Laura  
9 10 Ana  
10 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  
1 2 Luis  
2 3 Carlos  
3 4 María  
4 5 Elena  
5 3 Carlos  
6 7 Sofía  
7 8 Andrés  
8 9 Laura

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

ID Nombre Edad Salario Fecha\_Ingreso  
0 1 Ana 23 50000 2020-01-10  
1 2 Luis 35 60000 2019-06-15  
2 3 Carlos 30 58000 2020-03-25  
3 4 María 22 58000 2018-07-30  
4 5 Elena 28 49000 2018-07-30  
5 3 Carlos 30 58000 2020-03-25  
6 7 Sofía 34 72000 2016-11-22  
7 8 Andrés 45 61000 2021-04-01  
8 9 Laura 29 52000 2015-05-19  
9 10 Ana 41 67000 2014-08-25  
10 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  
1 2 Luis 35 60000 2019-06-15  
2 3 Carlos 30 58000 2020-03-25  
3 4 María 22 58000 2018-07-30  
5 3 Carlos 30 58000 2020-03-25  
6 7 Sofía 34 72000 2016-11-22  
7 8 Andrés 45 61000 2021-04-01  
8 9 Laura 29 52000 2015-05-19  
9 10 Ana 41 67000 2014-08-25  
10 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  
0 1 Ana 23 50000 2020-01-10 junior  
1 2 Luis 35 60000 2019-06-15 junior  
2 3 Carlos 30 58000 2020-03-25 junior  
3 4 María 22 58000 2018-07-30 junior  
4 5 Elena 28 49000 2018-07-30 junior  
5 3 Carlos 30 58000 2020-03-25 junior  
6 7 Sofía 34 72000 2016-11-22 Senior  
7 8 Andrés 45 61000 2021-04-01 mid  
8 9 Laura 29 52000 2015-05-19 junior  
9 10 Ana 41 67000 2014-08-25 Senior  
10 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  
0 1 Ana 23 50000 2020-01-10 junior 2955.0 47045.0  
1 2 Luis 35 60000 2019-06-15 junior 3546.0 56454.0  
2 3 Carlos 30 58000 2020-03-25 junior 3427.8 54572.2  
3 4 María 22 58000 2018-07-30 junior 3427.8 54572.2  
4 5 Elena 28 49000 2018-07-30 junior 2895.9 46104.1  
5 3 Carlos 30 58000 2020-03-25 junior 3427.8 54572.2  
6 7 Sofía 34 72000 2016-11-22 Senior 4255.2 67744.8  
7 8 Andrés 45 61000 2021-04-01 mid 3605.1 57394.9  
8 9 Laura 29 52000 2015-05-19 junior 3073.2 48926.8  
9 10 Ana 41 67000 2014-08-25 Senior 3959.7 63040.3  
10 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  
0 A 250 Lima  
1 B 150 Lima  
0 A 100 Bogotá  
1 C 200 Bogotá

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

Producto Ventas Ciudad Producto Inventario  
0 A 250 Lima A 30.0  
1 B 150 Lima B 45.0  
2 A 100 Bogotá NaN NaN  
3 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  
0 A Enero 100  
1 A Febrero 150  
2 B Enero 200  
3 B Febrero 250  
4 C Enero 300  
5 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  
0 A Enero 100  
1 B Enero 200  
2 C Enero 300  
3 A Febrero 150  
4 B Febrero 250  
5 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  
0 Juan Perez  
1 Maria Gomez  
2 Luis Martinez

Fecha  
0 01-01-2024  
1 15-02-2024  
2 30-03-2024

Despues de separar:

Nombre\_Completo Nombre Apellido  
0 Juan Perez Juan Perez  
1 Maria Gomez Maria Gomez  
2 Luis Martinez Luis Martinez

Fecha dia mes año  
0 01-01-2024 01 01 2024  
1 15-02-2024 15 02 2024  
2 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  
0 01-01-2024 01 01 2024 01/01/2024  
1 15-02-2024 15 02 2024 15/02/2024  
2 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  
0 Manzana Fruta Alta 50  
1 Banana Fruta Media 30  
2 Cereza Fruta Baja 70  
3 Durazno Fruta Alta 85  
4 Pera Fruta Media 40

Despues:

producto categoria calidad ventas Categoria ordinal  
0 Manzana Fruta Alta 50 3  
1 Banana Fruta Media 30 2  
2 Cereza Fruta Baja 70 1  
3 Durazno Fruta Alta 85 3  
4 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  
0 Manzana Fruta 50 3 True False False  
1 Banana Fruta 30 2 False False True  
2 Cereza Fruta 70 1 False True False  
3 Durazno Fruta 85 3 True False False  
4 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  
0 Auto Rojo 20000 150  
1 Camioneta Azul 30000 120  
2 Moto Negro 15000 130  
3 Camion Blanco 40000 60  
4 Auto Rojo 18000 180

Despues:

precio ventas vehiculo\_Auto vehiculo\_Camion vehiculo\_Camioneta vehiculo\_Moto color\_Azul color\_Blanco color\_Negro color\_Rojo  
0 20000 150 True False False False False False False True  
1 30000 120 False False True False True False False False  
2 15000 130 False False False True False False True False  
3 40000 60 False True False False False True False False  
4 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  
0 20000 150 False False False False False True  
1 30000 120 False True False False False False  
2 15000 130 False False True False True False  
3 40000 60 True False False True False False  
4 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  
0 No No No No No Si  
1 No Si No No No No  
2 No No Si No Si No  
3 Si No No Si No No  
4 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  
0 Manzana Fruta 50 0.5 2024-01-01  
1 Manzana Fruta 30 0.5 2024-01-02  
2 Banana Fruta 20 0.3 2024-01-03  
3 Banana Legumbre 15 0.3 2024-01-04  
4 Cereza Fruta 10 0.8 2024-01-05  
5 Cereza Fruta 25 0.8 2024-01-06  
6 Manzana Fruta 45 0.5 2024-01-07  
7 Banana Legumbre 55 0.3 2024-01-08  
8 Cereza Fruta 40 0.8 2024-01-09

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  
producto   
Banana 90 15 55  
Cereza 75 10 40  
Manzana 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  
0 Banana 90 15 55  
1 Cereza 75 10 40  
2 Manzana 125 30 50

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

total\_sales average\_price  
categoría producto   
Fruta Banana 20 0.3  
 Cereza 75 0.8  
 Manzana 125 0.5  
Legumbre Banana 70 0.3

## Funciones personalizadas

Usan el metodo apply, se puede integrar funciones personalizadas

Dataframe original:

producto categoría ventas precio\_unitario fecha  
0 Manzana Fruta 50 0.5 2024-01-01  
1 Manzana Fruta 30 0.5 2024-01-02  
2 Banana Fruta 20 0.3 2024-01-03  
3 Banana Fruta 15 0.3 2024-01-04  
4 Cereza Fruta 10 0.8 2024-01-05  
5 Cereza Fruta 25 0.8 2024-01-06  
6 Manzana Fruta 45 0.5 2024-01-07  
7 Banana Fruta 55 0.3 2024-01-08  
8 Cereza Fruta 40 0.8 2024-01-09

dataframe con los ingresos totales:

producto categoría ventas precio\_unitario fecha Ingresos totales  
0 Manzana Fruta 50 0.5 2024-01-01 25.0  
1 Manzana Fruta 30 0.5 2024-01-02 15.0  
2 Banana Fruta 20 0.3 2024-01-03 6.0  
3 Banana Fruta 15 0.3 2024-01-04 4.5  
4 Cereza Fruta 10 0.8 2024-01-05 8.0  
5 Cereza Fruta 25 0.8 2024-01-06 20.0  
6 Manzana Fruta 45 0.5 2024-01-07 22.5  
7 Banana Fruta 55 0.3 2024-01-08 16.5  
8 Cereza Fruta 40 0.8 2024-01-09 32.0

dataframe con la clasificacion de las ventas:

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

dataframe con el porcentaje de las ventas:

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

Agrupación de productos por ventas:

producto sum mean  
0 Banana 90 30.000000  
1 Cereza 75 25.000000  
2 Manzana 125 41.666667

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

producto ventas  
0 Banana 40  
1 Cereza 30  
2 Manzana 20

Agrupación de ingreso total por producto:

producto categoría ventas ingreso\_total  
0 Banana Fruta 90 27.0  
1 Cereza Fruta 75 60.0  
2 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  
0 bk101 Gambardella, Matthew XML Developer's Guide Computer 44.95 2000-10-01  
1 bk102 Ralls, Kim Midnight Rain Fantasy 5.95 2000-12-16  
2 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  
0 Ordenador PortÃ¡til ElectrÃ³nica 800.0  
1 Mesa de Comedor Muebles 300.0  
2 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'