# 🐍 Introducción a Python para el Análisis de Datos


## 👨‍💻 Jorge Gómez Galván

- LinkedIn: linkedin.com/in/jorgeggalvan/
- E-mail: gomezgalvanjorge@gmail.com


# Capítulo 3: Manipulación de Datos

---

Este notebook aborda las técnicas fundamentales para la manipulación de datos, incluyendo la agregación de datos para resumir información, las operaciones con fechas mediante la librería Datetime, y la unión de tablas en Pandas para combinar datasets. También se explora el pivoteo de tablas para reorganizar datos.

*El notebook ha sido adaptado a partir del trabajo de Juan Martín Bellido, cuyo contenido original se encuentra en este enlace.*

## Índice

---



## 3.1 - Agregación de datos

---

En el contexto del análisis de datos, agregar datos es fundamental para realizar cálculos que permitan resumir información y extraer insights de valor. Este proceso es similar al que se hace al crear tablas dinámicas (*pivot tables*) en Excel, donde los datos se resumen y ordenan para facilitar su interpretación y análisis.

Al realizar una agregación, se debe definir el tipo de cálculo que se desee aplicar. Para esto, se utilizan las funciones de agregación específicas diseñadas para realizar cálculos resumidos.

A continuación, se presentan las funciones de agregación más comunes:

| Función de agregación | Operación |
|---|---|
| `count()` | Número de filas no nulas |
| `size()` | Número total de filas (incluyendo nulos) |
| `nunique()` | Número de valores únicos |
| `sum()` | Suma |
| `mean()` | Media aritmética |
| `median()` | Mediana |
| `mode()` | Moda |
| `max()` | Valor máximo |
| `min()` | Valor mínimo |

### Agregaciones básicas

Para realizar una agregación básica sobre un DataFrame, puedes utilizar directamente las funciones de agregación.

```
df.agg_function()
```

In [1]:
```
# Importamos Pandas
import pandas as pd
```

In [2]:
```
# Importamos un DataFrame
df_jamesbond = pd.read_csv('./data/james_bond.csv')
```

In [3]:
```
# Aplicamos la función de agregación 'max()' para cada una de las variables
df_jamesbond.max()
```

Out[3]:
```
Film                    You Only Live Twice
Year                                   2021
Actor                       Timothy Dalton
Director                     Terence Young
Shooting Locations    Mexico, United States
Box Office                            943.5
Budget                                226.4
Bond Actor Salary                      17.9
IMDb Score                              8.0
dtype: object
```

In [4]:
```
# Seleccionamos una variable y obtenemos su media
df_jamesbond['Bond Actor Salary'].mean()
```

Out[4]:
```
6.85
```

In [5]:
```
# Seleccionamos dos variables y obtenemos su suma, redondeando el resultado
df_jamesbond[['Box Office', 'Budget']].sum().round()
```

Out[5]:
```
Box Office    13064.0
Budget         2310.0
dtype: float64
```

In [6]:
```
# Seleccionamos una variable y obtenemos sus valores únicos
df_jamesbond['Actor'].nunique()
```

Out[6]:
```
7
```

In [7]:
```
# Para seleccionar una única variable sin espacios en su nombre, podemos usar la notación de punto
df_jamesbond.Actor.nunique()
```

Out[7]:
```
7
```

## Método `.agg()`

Pandas incluye el método `.agg()`, dedicado expresamente a agregar datos que permite aplicar varias funciones de agregación a diferentes columnas de un DataFrame.

La sintaxis que se utiliza es la siguiente:

```
df.agg({'column_name':'agg_func'})
```

👉 Se utiliza un diccionario para especificar las variables sobre las cuales se realizan las operaciones de agregación.

In [8]:
```
# Obtenemos los actores únicos con '.agg()'
df_jamesbond.agg({'Actor':'nunique'})
```

Out[8]:
```
Actor    7
dtype: int64
```

In [9]:
```
# Calculamos el mínimo, el máximo y la media de 'Box Office'
df_jamesbond.agg({'Box Office':['min', 'max', 'mean']})
```

Out[9]:

| | Box Office |
|---|---|
| **min** | 250.900000 |
| **max** | 943.500000 |
| **mean** | 483.866667 |

In [10]:
```
# Agregamos dos variables, y calculamos el mínimo, el máximo y la media
df_jamesbond.agg({'Budget':['min', 'max', 'mean'],
                  'Bond Actor Salary':['min', 'max', 'mean']})
```

Out[10]:

| | Budget | Bond Actor Salary |
|---|---|---|
| **min** | 7.000000 | 0.60 |
| **max** | 226.400000 | 17.90 |
| **mean** | 85.559259 | 6.85 |

In [11]:
```
# Agregamos dos variables, y calculamos el mínimo, el máximo y la media
df_jamesbond.agg({'Budget':['min', 'max', 'mean'],
```

```
                   'Bond Actor Salary':['min', 'max', 'mean']})
```

Out[11]:

| | Budget | Bond Actor Salary |
|---|---|---|
| **min** | 7.000000 | 0.60 |
| **max** | 226.400000 | 17.90 |
| **mean** | 85.559259 | 6.85 |

In [12]:
```
# Podemos realizar calculos diferentes al agregar más de una variable
df_jamesbond.agg({'Budget':['min', 'max', 'mean'],
                  'Bond Actor Salary':['mean', 'sum']})
```

Out[12]:

| | Budget | Bond Actor Salary |
|---|---|---|
| **min** | 7.000000 | NaN |
| **max** | 226.400000 | NaN |
| **mean** | 85.559259 | 6.85 |
| **sum** | NaN | 123.30 |

## Agregaciones agrupadas

El método `.groupby()` de Pandas es muy importante para analizar datos, ya que se utiliza con mucha frecuencia para realizar agregaciones. Este método permite agrupar datos en función de una o más sobre variables categóricas y luego aplicar funciones de agregación sobre cada grupo.

La sintaxis para agrupar por más de una columna categórica es:

> `df.groupby('column_name_1').agg({'column_name_2':'agg_func'})`

> `df.groupby('column_name_1')['column_name_2'].agg_func`

- `column_name_1` es la columna por las que se desea agrupar los datos.
- `column_name_2` es la columna a la que aplica la función de agregación.

In [13]:
```
# Agrupamos por actor y calculamos la mediana de dos variables para cada uno
df_jamesbond.groupby('Actor')[['Bond Actor Salary','Box Office']].median()
```

Out[13]:

| Actor | Bond Actor Salary | Box Office |
|---|---|---|
| **Daniel Craig** | 8.10 | 589.4 |
| **David Niven** | NaN | 260.0 |
| **George Lazenby** | 0.60 | 291.5 |
| **Pierce Brosnan** | 11.75 | 464.3 |
| **Roger Moore** | 8.45 | 449.4 |
| **Sean Connery** | 3.80 | 514.2 |
| **Timothy Dalton** | 6.55 | 282.2 |

In [14]:
```
# Repetimos la agrupación anterior utilizando el método '.agg()'
df_jamesbond.groupby('Actor').agg({'Bond Actor Salary':'median',
                                   'Box Office':'median'})
```

Out[14]:

| Actor | Bond Actor Salary | Box Office |
|---|---|---|
| **Daniel Craig** | 8.10 | 589.4 |
| **David Niven** | NaN | 260.0 |
| **George Lazenby** | 0.60 | 291.5 |
| **Pierce Brosnan** | 11.75 | 464.3 |
| **Roger Moore** | 8.45 | 449.4 |
| **Sean Connery** | 3.80 | 514.2 |
| **Timothy Dalton** | 6.55 | 282.2 |

👉 Al realizar agregaciones en DataFrames, puede ser práctica utilizar otros métodos para formatear los resultados. Por ejemplo, se puede renombrar los nombre de las nuevas columnas, ordenar los datos o resetear los índices para mejorar la claridad y la interpretabilidad.

In [15]:
```
# Repetimos otra vez la agrupación anterior y la almacenamos en un nuevo DataFrame
df_bond_actor = df_jamesbond.groupby('Actor').agg({'Bond Actor Salary':'median',
                                                   'Box Office':'median'})

# Ordenamos la agregación agrupada y renombramos las columnas por defecto
df_bond_actor.sort_values(by='Bond Actor Salary', ascending=False).rename(columns={'Bond Actor Salary':'Median Actor Sala
```

Out[15]:

| Actor | Median Actor Salary | Median Box Office |
|---|---|---|
| **Pierce Brosnan** | 11.75 | 464.3 |
| **Roger Moore** | 8.45 | 449.4 |
| **Daniel Craig** | 8.10 | 589.4 |
| **Timothy Dalton** | 6.55 | 282.2 |
| **Sean Connery** | 3.80 | 514.2 |
| **George Lazenby** | 0.60 | 291.5 |
| **David Niven** | NaN | 260.0 |

👉 Se puede agrupar los datos por varias columnas con una lista de nombres de columnas al método `.groupby()`, y aplicar diferentes funciones de agregación a una o más columnas especificadas.

```
df.groupby(['column_1', 'column_2']).agg({'column_3':['agg_func_1','agg_func_2',...],
                                          'column_4':['agg_func_1',...]})
```

In [16]:
```
# Agrupamos por actor y calculamos varias estadísticas para 'Bond Actor Salary' y 'Budget'
df_jamesbond.groupby('Actor').agg({'Bond Actor Salary':['size','sum','max','mean'],
                                   'Budget':['max','mean']})
```

Out[16]:

| | Bond Actor Salary | | | | Budget | |
|---|---|---|---|---|---|---|
| | size | sum | max | mean | max | mean |
| **Actor** | | | | | | |
| **Daniel Craig** | 5 | 25.9 | 14.5 | 8.633333 | 226.4 | 185.920000 |
| **David Niven** | 1 | 0.0 | NaN | NaN | 70.0 | 70.000000 |
| **George Lazenby** | 1 | 0.6 | 0.6 | 0.600000 | 37.3 | 37.300000 |
| **Pierce Brosnan** | 4 | 46.5 | 17.9 | 11.625000 | 158.3 | 130.825000 |
| **Roger Moore** | 7 | 16.9 | 9.1 | 8.450000 | 91.5 | 51.957143 |
| **Sean Connery** | 7 | 20.3 | 5.8 | 3.383333 | 86.0 | 37.242857 |
| **Timothy Dalton** | 2 | 13.1 | 7.9 | 6.550000 | 68.8 | 62.750000 |

In [17]:
```
# Agrupamos por director y actor, y calculamos la media de 'Box Office'
df_jamesbond.groupby(['Director','Actor']).agg({'Box Office':'mean'})
```

Out[17]:

| Director | Actor | Box Office |
|---|---|---|
| **Cary Joji Fukunaga** | **Daniel Craig** | 396.800000 |
| **Guy Hamilton** | **Roger Moore** | 397.150000 |
| | **Sean Connery** | 631.450000 |
| **Irvin Kershner** | **Sean Connery** | 314.000000 |
| **John Glen** | **Roger Moore** | 366.133333 |
| | **Timothy Dalton** | 282.200000 |
| **Ken Hughes** | **David Niven** | 260.000000 |
| **Lee Tamahori** | **Pierce Brosnan** | 465.400000 |
| **Lewis Gilbert** | **Roger Moore** | 534.000000 |
| | **Sean Connery** | 514.200000 |
| **Marc Forster** | **Daniel Craig** | 514.200000 |
| **Martin Campbell** | **Daniel Craig** | 589.400000 |
| | **Pierce Brosnan** | 518.500000 |
| **Michael Apted** | **Pierce Brosnan** | 439.500000 |
| **Peter R. Hunt** | **George Lazenby** | 291.500000 |
| **Roger Spottiswoode** | **Pierce Brosnan** | 463.200000 |
| **Sam Mendes** | **Daniel Craig** | 834.500000 |
| **Terence Young** | **Sean Connery** | 613.566667 |

## Método `.value_counts()`

Una alternativa rápida para agrupar y contar la frecuencia de los valores únicas de una columna es utilizando el método `.value_counts()`. Este método, aunque no es una función de agregación en el sentido estricto del término, proporciona una forma sencilla de obtener un resumen de cuántas veces se repiten los valores en una columna específica.

```
serie.value_counts()
```

In [18]:
```
# Calculamos la cantidad de apariciones únicas de cada valor en la columna 'Actor'
df_jamesbond['Actor'].value_counts()
```

Out[18]:
```
Actor
Sean Connery      7
Roger Moore       7
Daniel Craig      5
Pierce Brosnan    4
Timothy Dalton    2
David Niven       1
George Lazenby    1
Name: count, dtype: int64
```

In [19]:
```
# Calculamos la cantidad de combinaciones únicas de valores en las columnas 'Actor' y 'Director'
df_jamesbond[['Actor', 'Director']].value_counts()
```

Out[19]:
```
Actor           Director
Sean Connery    Terence Young         3
Roger Moore     John Glen             3
Timothy Dalton  John Glen             2
Roger Moore     Guy Hamilton          2
Daniel Craig    Sam Mendes            2
Sean Connery    Guy Hamilton          2
Roger Moore     Lewis Gilbert         2
Sean Connery    Lewis Gilbert         1
                Irvin Kershner        1
Daniel Craig    Cary Joji Fukunaga    1
                Marc Forster          1
Pierce Brosnan  Michael Apted         1
                Martin Campbell       1
                Lee Tamahori          1
George Lazenby  Peter R. Hunt         1
David Niven     Ken Hughes            1
Daniel Craig    Martin Campbell       1
Pierce Brosnan  Roger Spottiswoode    1
Name: count, dtype: int64
```

👉 `.value_counts()` siempre devuelve una Series (otra estructura de Pandas), por lo que puede ser útil aplicar el método `.to_frame()` para convertirla en un DataFrame.

In [20]:
```
# Convertimos el resultado anterior en un DataFrame
df_jamesbond['Actor'].value_counts().to_frame()
```

Out[20]:

| Actor | count |
|---|---|
| **Sean Connery** | 7 |
| **Roger Moore** | 7 |
| **Daniel Craig** | 5 |
| **Pierce Brosnan** | 4 |
| **Timothy Dalton** | 2 |
| **David Niven** | 1 |
| **George Lazenby** | 1 |

In [21]:
```
# Renombramos la columna de conteo y reseteamos el índice del resultado anterior
df_jamesbond['Actor'].value_counts().to_frame().rename(columns={'Actor':'Total Films'}).reset_index(names='Actor')
```

Out[21]:

| | Actor | count |
|---|---|---|
| **0** | Sean Connery | 7 |
| **1** | Roger Moore | 7 |
| **2** | Daniel Craig | 5 |
| **3** | Pierce Brosnan | 4 |
| **4** | Timothy Dalton | 2 |
| **5** | David Niven | 1 |
| **6** | George Lazenby | 1 |

## 3.2 - Operaciones con fechas

En Python, las fechas y horas no son tipos de variable nativos. Para poder trabajar con fechas y realizar operaciones con ellas, es necesario importar librerías específicas.

La librería principal para este propósito es el módulo Datetime. Este módulo `datetime` incluye varias clases para manejar fechas y horas.

In [22]:
```
# Importamos el módulo Datetime y las clases que utilizaremos
from datetime import date, datetime, timedelta
```

### Operaciones básicas con fechas

#### Crear una fecha específica

Para crear una fecha específica en Python, se utiliza la función `date` de la siguiente manera:

```
date(year, month, day)
```

⚠ No se debe definir una variable con el nombre 'date', porque se sobreescribiría la referencia a la clase `date`, lo que implicaría que no se podría crear otra fecha con esta clase.

In [23]:
```
# Creamos una variable con una fecha específica con 'date()'
new_date = date(2024, 10, 14)
new_date
```

Out[23]:
```
datetime.date(2024, 10, 14)
```

Si además de la fecha, se desea incluir horas, minutos o segundos, se puede utilizar la clase `datetime`.

```
date(year, month, day, hour, minute, second, milisecond)
```

In [24]:
```
# Creamos una fecha específica incluyendo la hora y el minuto con 'datetime()'
datetime(2024, 10, 14, 18, 30)
```

Out[24]:
```
datetime.datetime(2024, 10, 14, 18, 30)
```

#### Obtener la fecha actual

La fecha actual se puede obtener mediante el método `.today()`, el cual devuelve la fecha del sistema en el formato año, mes y día.

```
date.today()
```

In [25]:
```
# Obtenemos la fecha actual
today = date.today()
today
```

Out[25]:
```
datetime.date(2025, 6, 15)
```

#### Realizar operaciones con una fecha

Es posible restar dos fechas para calcular la diferencia en días, devolviendo una variable de tipo *timedelta*. Sin embargo, no se puede sumar dos fechas directamente, sólo restarlas.

Para ajustar una fecha, ya sea sumando o restando días o semanas, se utiliza el objeto `timedelta()`, que permite realizar operaciones aritméticas con fechas.

El formato básico de `timedelta()` es el siguiente:

```
timedelta(weeks=0, days=0, hours=0, minutes=0)
```

In [26]:
```
# Restamos dos fechas
date(2024, 10, 14) - date(2024, 9, 14)
```

Out[26]:
```
datetime.timedelta(days=30)
```

In [27]:
```
# Restamos 7 días a la fecha
date(2024, 10, 14) + timedelta(days = -7)
```

Out[27]:
```
datetime.date(2024, 10, 7)
```

In [28]:
```
# Sumamos 6 semanas a la fecha
date(2024, 10, 14) + timedelta(weeks = 6)
```

Out[28]:
```
datetime.date(2024, 11, 25)
```

In [29]:
```
# Sumamos 30 minutos a la fecha y hora
datetime(2024, 10, 14, 18, 30) + timedelta(minutes = 30)
```

Out[29]:
```
datetime.datetime(2024, 10, 14, 19, 0)
```

### Convertir a tipo fecha

En muchos de los datasets que se importan, es común que las fechas se encuentren almacenadas como cadenas de texto o números, lo que implica convertirlas a un objeto de tipo *date* o *datetime*. Esto se puede solucionar con la función `.to_datetime()` de Pandas.

In [30]:
```
# Importamos un DataFrame con fechas
df_amzn = pd.read_csv('./data/amzn_stock.csv')
df_amzn
```

Out[30]:

| | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|
| **0** | 1997-05-15 | 0.121875 | 0.125000 | 0.096354 | 0.097917 | 1443120000 |
| **1** | 1997-05-16 | 0.098438 | 0.098958 | 0.085417 | 0.086458 | 294000000 |
| **2** | 1997-05-19 | 0.088021 | 0.088542 | 0.081250 | 0.085417 | 122136000 |
| **3** | 1997-05-20 | 0.086458 | 0.087500 | 0.081771 | 0.081771 | 109344000 |
| **4** | 1997-05-21 | 0.081771 | 0.082292 | 0.068750 | 0.071354 | 377064000 |
| **...** | ... | ... | ... | ... | ... | ... |
| **7050** | 2025-05-23 | 198.899994 | 202.369995 | 197.850006 | 200.990005 | 33393500 |
| **7051** | 2025-05-27 | 203.089996 | 206.690002 | 202.190002 | 206.020004 | 34892000 |
| **7052** | 2025-05-28 | 205.919998 | 207.660004 | 204.410004 | 204.720001 | 28549800 |
| **7053** | 2025-05-29 | 208.029999 | 208.809998 | 204.229996 | 205.699997 | 34650000 |
| **7054** | 2025-05-30 | 204.839996 | 205.990005 | 201.699997 | 205.009995 | 51679400 |

7055 rows × 6 columns

In [31]:
```
# Consultamos los tipos de variables del DataFrame
df_amzn.dtypes
```

Out[31]:
```
Date       object
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
dtype: object
```

In [32]:
```
# Convertimos la columna 'Date' a tipo fecha
df_amzn['Date'] = pd.to_datetime(df_amzn['Date'])
df_amzn.dtypes
```

Out[32]:
```
Date      datetime64[ns]
Open             float64
High             float64
Low              float64
Close            float64
Volume             int64
dtype: object
```

En caso de que Pandas no identifique automáticamente el formato de fecha, se puede especificar editando el parámetro opcional `format`.

In [33]:
```
# Convertimos la columna 'Date' a tipo fecha, especificando el formato
df_amzn['Date'] = pd.to_datetime(df_amzn['Date'], format = '%Y-%m-%d')
df_amzn.dtypes
```

Out[33]:
```
Date      datetime64[ns]
Open             float64
High             float64
Low              float64
Close            float64
Volume             int64
dtype: object
```

### Extraer campos a partir de fechas

A partir de una fecha de tipo *datetime*, se pueden extraer diversos componentes que resultan muy útiles. Estos componentes permiten descomponer una fecha en sus elementos constitutivos, facilitando el análisis y la manipulación de información temporal.

A continuación, se indican los principales campos que se pueden obtener a partir de una fecha:

| Campo | Descripción |
|---|---|
| `year` | Año |
| `month_name()` | Nombre completo del mes (January, February...) |
| `month` | Número del mes (1 a 12) |
| `day` | Número del día del mes (1 a 31) |

| | |
|---|---|
| `day_name()` | Nombre del día de la semana (Monday, Tuesday...) |
| `weekday` | Número del día de la semana (0 = lunes, 6 = domingo) |
| `quarter` | Trimestre del año (1 a 4) |
| `isocalendar().week` | Número de la semana del año (1 a 52/53) |
| `dayofyear` | Número del día del año (1 a 365/366) |

In [34]:
```
# Creamos una columna con el año
df_amzn['Year'] = df_amzn['Date'].dt.year

# Creamos una columna con el mes (texto)
df_amzn['Month Name'] = df_amzn['Date'].dt.month_name()

# Creamos una columna con el mes
df_amzn['Month'] = df_amzn['Date'].dt.month

# Creamos una columna con el día
df_amzn['Day'] = df_amzn['Date'].dt.day

# Creamos una columna con el día de la semana (texto)
df_amzn['Week Day'] = df_amzn['Date'].dt.day_name()

# Creamos una columna con el día de la semana
df_amzn['Week Day Number'] = df_amzn['Date'].dt.weekday + 1 # Sumamos 1 para que el lunes sea 1 y el domingo sea 7
```

In [35]:
```
# Creamos una columna con el trimestre
df_amzn['Quarter'] = df_amzn['Date'].dt.quarter

# Creamos una columna con el número de semana
df_amzn['Week Number'] = df_amzn['Date'].dt.isocalendar().week

# Creamos una columna con el día del año
df_amzn['Day of Year'] = df_amzn['Date'].dt.dayofyear
```

In [36]:
```
# Mostramos el DataFrame con los nuevos campos creados
df_amzn.tail()
```

Out[36]:

| | Date | Open | High | Low | Close | Volume | Year | Month Name | Month | Day | Week Day | Week Day Number | Quarter |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **7050** | 2025-05-23 | 198.899994 | 202.369995 | 197.850006 | 200.990005 | 33393500 | 2025 | May | 5 | 23 | Friday | 5 | 2 |
| **7051** | 2025-05-27 | 203.089996 | 206.690002 | 202.190002 | 206.020004 | 34892000 | 2025 | May | 5 | 27 | Tuesday | 2 | 2 |
| **7052** | 2025-05-28 | 205.919998 | 207.660004 | 204.410004 | 204.720001 | 28549800 | 2025 | May | 5 | 28 | Wednesday | 3 | 2 |
| **7053** | 2025-05-29 | 208.029999 | 208.809998 | 204.229996 | 205.699997 | 34650000 | 2025 | May | 5 | 29 | Thursday | 4 | 2 |
| **7054** | 2025-05-30 | 204.839996 | 205.990005 | 201.699997 | 205.009995 | 51679400 | 2025 | May | 5 | 30 | Friday | 5 | 2 |

## Agregar datos por componentes de fecha

Se pueden agregar datos agrupando por componentes de fecha que se hayan calculado, utilizándolos como variables categóricas.

In [37]:
```
# Agrupamos por año y trimestre para calcular la media de 'Close'
df_amzn.groupby(['Year', 'Quarter']).agg({'Close':'mean'})
```

Out[37]:

| Year | Quarter | Close |
|---|---|---|
| **1997** | **2** | 0.077759 |
| | **3** | 0.131816 |
| | **4** | 0.220105 |
| **1998** | **1** | 0.282681 |
| | **2** | 0.447464 |
| **...** | **...** | ... |
| **2024** | **2** | 183.703016 |
| | **3** | 182.457500 |
| | **4** | 204.584687 |
| **2025** | **1** | 217.002334 |
| | **2** | 190.685476 |

113 rows × 1 columns

In [38]:
```
# Calculamos la media de 'Close' agrupando por el año
amzn_yearly_close = df_amzn.groupby('Year').agg({'Close':'mean'})
amzn_yearly_close.tail(10) # Mostramos los últimos 10 años
```

Out[38]:

| Year | Close |
|---|---|
| **2016** | 34.976157 |
| **2017** | 48.408351 |
| **2018** | 82.086309 |
| **2019** | 89.459460 |
| **2020** | 134.042755 |
| **2021** | 167.193349 |
| **2022** | 126.098819 |
| **2023** | 121.372800 |
| **2024** | 184.628691 |
| **2025** | 206.165980 |

## Método `.shift()`

El método `.shift()` permite comparar un valor con sus valores anteriores o siguientes en la misma columna, ya que toma el valor de filas anteriores o posteriores.

```
df['colum_name'].shift(periods, fill_value)
```

- `periods` : número de períodos que se van a desplazar (puede ser positivo o negativo).
- `fill_value` : valor que se utiliza para los valores nulos (por defecto, *NaN*).

In [39]:
```
# Creamos una columna en la que se asigna los valores de 'Close' desplazados una fila hacia abajo
amzn_yearly_close['Last Year Close'] = amzn_yearly_close['Close'].shift(1)
amzn_yearly_close.tail(10)
```

Out[39]:

| Year | Close | Last Year Close |
|---|---|---|
| **2016** | 34.976157 | 23.906915 |
| **2017** | 48.408351 | 34.976157 |
| **2018** | 82.086309 | 48.408351 |
| **2019** | 89.459460 | 82.086309 |
| **2020** | 134.042755 | 89.459460 |
| **2021** | 167.193349 | 134.042755 |
| **2022** | 126.098819 | 167.193349 |
| **2023** | 121.372800 | 126.098819 |
| **2024** | 184.628691 | 121.372800 |
| **2025** | 206.165980 | 184.628691 |

In [40]:
```
# Calculamos el porcentaje del crecimiento anual
amzn_yearly_close['YOY Growth Rate'] = ((amzn_yearly_close['Close'] / amzn_yearly_close['Last Year Close']) - 1).round(2)
amzn_yearly_close.tail(10)
```

Out[40]:

| Year | Close | Last Year Close | YOY Growth Rate |
|---|---|---|---|
| **2016** | 34.976157 | 23.906915 | 0.46 |
| **2017** | 48.408351 | 34.976157 | 0.38 |
| **2018** | 82.086309 | 48.408351 | 0.70 |
| **2019** | 89.459460 | 82.086309 | 0.09 |
| **2020** | 134.042755 | 89.459460 | 0.50 |
| **2021** | 167.193349 | 134.042755 | 0.25 |
| **2022** | 126.098819 | 167.193349 | -0.25 |
| **2023** | 121.372800 | 126.098819 | -0.04 |
| **2024** | 184.628691 | 121.372800 | 0.52 |
| **2025** | 206.165980 | 184.628691 | 0.12 |

## 3.3 - Unir tablas de datos

Las uniones de DataFrames son fundamentales en el análisis de datos, ya que permiten combinar múltiples datasets para obtener información más completa y realizar análisis más profundos. En Pandas, existen varios métodos para unir dos o más DataFrames, y cada uno tiene características específicas sobre cómo se combinan las tablas.

Se pueden distinguir tres principales formas de unir tablas de datos, especificando si son por filas o por columnas:

| Método | Tipo de unión | Descripción |
|---|---|---|
| `pd.concat()` | Concatenación por filas | Apila DataFrames uno sobre otro. |
| | Concatenación por columnas | Combina DataFrames uno al lado del otro. |
| `.merge()` | Left Join | Incluye todas las filas del DataFrame principal y, del secundario, sólo aquellas donde la clave coincida con la del primero. |
| | Right Join | Incluye todas las filas del DataFrame secundario y, del principal, sólo aquellas donde la clave coincida con la del segundo. |
| | Inner Join | Incluye sólo filas con claves coincidentes en ambos DataFrames. |
| | Outer Join | Incluye todas las filas de ambos DataFrames. |
| `.join()` | Unión por índices | Combina DataFrames a partir de sus índices. |

### Concatenar tablas

La función `pd.concat()` permite unir DataFrames de manera vertical (apilándolos por filas) u horizontal (combinándolos por columnas).

### Concatenar tablas por filas

Al concatenar DataFrames verticalmente, se apilan uno sobre otro, aumentando el número de filas. Esta función es útil cuando se tienen DataFrames con la misma estructura, es decir, que contienen las mismas columnas.

La sintaxis básica es:

```
pd.concat([df_1, df_2, ...])
```

👉 El orden de las columnas no es relevante, siempre que los nombres sean iguales en ambos DataFrames.

⚠️ Si los nombres de las columnas no coinciden entre DataFrames, se agregan las columnas adicionales y se completan con `NaN` aquellas celdas donde falten datos en algún DataFrame

In [41]:
```
# Importamos dos DataFrames
df_aapl_23 = pd.read_csv('./data/aapl_stock_2023.csv')
df_aapl_24 = pd.read_csv('./data/aapl_stock_2024.csv')

# Convertimos las columnas 'Date' a tipo fecha
df_aapl_23['Date'] = pd.to_datetime(df_aapl_23['Date'])
df_aapl_24['Date'] = pd.to_datetime(df_aapl_24['Date'])
```

In [42]:
```
# Mostramos las primeras filas del primer DataFrame
df_aapl_23.head(3)
```

Out[42]:

| | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|
| **0** | 2023-01-03 | 130.279999 | 130.899994 | 124.169998 | 125.070000 | 112117500 |
| **1** | 2023-01-04 | 126.889999 | 128.660004 | 125.080002 | 126.360001 | 89113600 |
| **2** | 2023-01-05 | 127.129997 | 127.769997 | 124.760002 | 125.019997 | 80962700 |

In [43]:
```
# Mostramos las primeras filas del segundo DataFrame
df_aapl_24.head(3)
```

Out[43]:

| | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|
| **0** | 2024-01-02 | 185.789438 | 187.070068 | 182.553143 | 184.290421 | 82488700 |
| **1** | 2024-01-03 | 182.880742 | 184.528677 | 182.096477 | 182.910522 | 58414500 |
| **2** | 2024-01-04 | 180.825770 | 181.758939 | 179.565014 | 180.587524 | 71983600 |

In [44]:
```
# Unimos los dos DataFrames por filas
df_aapl = pd.concat([df_aapl_23, df_aapl_24])
df_aapl
```

Out[44]:

| | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|
| **0** | 2023-01-03 | 130.279999 | 130.899994 | 124.169998 | 125.070000 | 112117500 |
| **1** | 2023-01-04 | 126.889999 | 128.660004 | 125.080002 | 126.360001 | 89113600 |
| **2** | 2023-01-05 | 127.129997 | 127.769997 | 124.760002 | 125.019997 | 80962700 |
| **3** | 2023-01-06 | 126.010002 | 130.289993 | 124.889999 | 129.619995 | 87754700 |
| **4** | 2023-01-09 | 130.470001 | 133.410004 | 129.889999 | 130.149994 | 70790800 |
| **...** | ... | ... | ... | ... | ... | ... |
| **247** | 2024-12-24 | 254.875189 | 257.588630 | 254.675658 | 257.578674 | 23234700 |
| **248** | 2024-12-26 | 257.568678 | 259.474086 | 257.010028 | 258.396667 | 27237100 |
| **249** | 2024-12-27 | 257.209530 | 258.077462 | 252.451019 | 254.974930 | 42355300 |
| **250** | 2024-12-30 | 251.623020 | 252.889969 | 250.146586 | 251.593094 | 35557500 |
| **251** | 2024-12-31 | 251.832526 | 252.670501 | 248.829760 | 249.817383 | 39480700 |

502 rows × 6 columns

El parámetro `keys` permite agregar etiquetas en el eje de concatenación para rastrear el origen de las filas en el DataFrame resultante

In [45]:
```
# Añadimos etiquetas al unir los dos DataFrame para ayudar a identificar el origen de los datos
pd.concat([df_aapl_23, df_aapl_24], keys=[2023, 2024])
```

Out[45]:

| | | Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|---|---|
| **2023** | **0** | 2023-01-03 | 130.279999 | 130.899994 | 124.169998 | 125.070000 | 112117500 |
| | **1** | 2023-01-04 | 126.889999 | 128.660004 | 125.080002 | 126.360001 | 89113600 |
| | **2** | 2023-01-05 | 127.129997 | 127.769997 | 124.760002 | 125.019997 | 80962700 |
| | **3** | 2023-01-06 | 126.010002 | 130.289993 | 124.889999 | 129.619995 | 87754700 |
| | **4** | 2023-01-09 | 130.470001 | 133.410004 | 129.889999 | 130.149994 | 70790800 |
| **...** | **...** | ... | ... | ... | ... | ... | ... |
| **2024** | **247** | 2024-12-24 | 254.875189 | 257.588630 | 254.675658 | 257.578674 | 23234700 |
| | **248** | 2024-12-26 | 257.568678 | 259.474086 | 257.010028 | 258.396667 | 27237100 |
| | **249** | 2024-12-27 | 257.209530 | 258.077462 | 252.451019 | 254.974930 | 42355300 |
| | **250** | 2024-12-30 | 251.623020 | 252.889969 | 250.146586 | 251.593094 | 35557500 |
| | **251** | 2024-12-31 | 251.832526 | 252.670501 | 248.829760 | 249.817383 | 39480700 |

502 rows × 6 columns

#### Concatenar tablas por columnas

Si se desea unir DataFrames horizontalmente, uno al lado del otro, se utiliza también `pd.concat()` especificando el parámetro `axis=1`.

```
pd.concat([df_1, df_2, ...], axis=1)
```

In [46]:
```
# Unimos los dos DataFrames por columnas
pd.concat([df_aapl_23, df_aapl_24], axis=1)
```

Out[46]:

| | Date | Open | High | Low | Close | Volume | Date | Open | High | Low | Close | V |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2023-01-03 | 130.279999 | 130.899994 | 124.169998 | 125.070000 | 112117500.0 | 2024-01-02 | 185.789438 | 187.070068 | 182.553143 | 184.290421 | 824 |
| **1** | 2023-01-04 | 126.889999 | 128.660004 | 125.080002 | 126.360001 | 89113600.0 | 2024-01-03 | 182.880742 | 184.528677 | 182.096477 | 182.910522 | 584 |
| **2** | 2023-01-05 | 127.129997 | 127.769997 | 124.760002 | 125.019997 | 80962700.0 | 2024-01-04 | 180.825770 | 181.758939 | 179.565014 | 180.587524 | 719 |
| **3** | 2023-01-06 | 126.010002 | 130.289993 | 124.889999 | 129.619995 | 87754700.0 | 2024-01-05 | 180.666963 | 181.431354 | 178.860187 | 179.862839 | 623 |
| **4** | 2023-01-09 | 130.470001 | 133.410004 | 129.889999 | 130.149994 | 70790800.0 | 2024-01-08 | 180.766224 | 184.250716 | 180.180517 | 184.210999 | 591 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **247** | 2023-12-27 | 192.490005 | 193.500000 | 191.089996 | 193.149994 | 48087700.0 | 2024-12-24 | 254.875189 | 257.588630 | 254.675658 | 257.578674 | 232 |
| **248** | 2023-12-28 | 194.139999 | 194.660004 | 193.169998 | 193.580002 | 34049900.0 | 2024-12-26 | 257.568678 | 259.474086 | 257.010028 | 258.396667 | 272 |
| **249** | 2023-12-29 | 193.899994 | 194.399994 | 191.729996 | 192.529999 | 42628800.0 | 2024-12-27 | 257.209530 | 258.077462 | 252.451019 | 254.974930 | 423 |
| **250** | NaT | NaN | NaN | NaN | NaN | NaN | 2024-12-30 | 251.623020 | 252.889969 | 250.146586 | 251.593094 | 355 |
| **251** | NaT | NaN | NaN | NaN | NaN | NaN | 2024-12-31 | 251.832526 | 252.670501 | 248.829760 | 249.817383 | 394 |

252 rows × 12 columns

## Unir tablas por columnas o índices

Para unir tablas a partir de campos comunes o de índices, se emplean los métodos `.merge()` y `.join()`, respectivamente.

#### Unir tablas con campos en común (por columnas)

El método `.merge()` permite combinar DataFrames mediante uniones similares a las que se realizan en bases de datos SQL, es decir, en uno o más campos o claves comunes que están presentes en ambos DataFrames.

La sintaxis es la siguiente:

```
df_1.merge(df_2, how='join_type', left_on='key_1', right_on='key_2')
```

👉 En toda unión entre DataFrames, es fudamental que se identifiquen claramente las claves o *keys* (las columnas comunes utilizadas para emparejar las filas entre los datasets). Si no se especifican las claves, `.merge()` intentará utilizar cualquier columna con el mismo nombre en ambos DataFrames para realizar la unión. En caso de las claves tengan nombres diferentes en cada DataFrame, se deben utilizar los parámetros `left_on` y `right_on` dentro del método `.merge()`, indicando qué columnas son las claves para la unión.

Respecto a los tipos de uniones, este método admite cinco:

- **Left Join** ( `how='left'` )**:** devuelve todas las filas del DataFrame principal (izquierdo) y sólo las filas coincidentes del DataFrame secundario (derecho). Las filas del DataFrame principal que no encuentren coincidencia en el secundario tendrán valores `NaN` en las columnas correspondientes al secundario.
- **Left Join** ( `how='right'` )**:** devuelve todas las filas del DataFrame secundario (derecho) y sólo las filas coincidentes del DataFrame principal (izquierdo). Las filas del DataFrame secundario que no encuentren coincidencia en el principal tendrán valores `NaN` en las columnas correspondientes al principal.
- **Outer Join** ( `how='outer'` )**:** devuelve todas las filas de ambos DataFrames, completando con `NaN` en las columnas del DataFrame que no tenga coincidencia en el otro, y viceversa.
- **Inner Join** ( `how='inner'` )**:** devuelve sólo las filas coincidentes en ambos DataFrames, eliminando las filas sin coincidencias, por lo que no se generan `NaN`.
- **Cross Join** ( `how='cross'` )**:** devuelve todas las combinaciones posibles de filas entre ambos DataFrames. No requiere claves de unión y no se generan `NaN`.

In [47]:
```
# Unimos dos DataFrames por una columna en común ('Date')
df_amzn[['Date','Close']].merge(df_aapl[['Date','Close']], # Seleccionamos el DataFrame principal y secundario
                                how='left', # Especificamos el tipo de unión
                                left_on='Date', right_on='Date') # Especificamos las claves
```

Out[47]:

| | Date | Close_x | Close_y |
|---|---|---|---|
| **0** | 1997-05-15 | 0.097917 | NaN |
| **1** | 1997-05-16 | 0.086458 | NaN |
| **2** | 1997-05-19 | 0.085417 | NaN |
| **3** | 1997-05-20 | 0.081771 | NaN |
| **4** | 1997-05-21 | 0.071354 | NaN |
| **...** | ... | ... | ... |
| **7050** | 2025-05-23 | 200.990005 | NaN |
| **7051** | 2025-05-27 | 206.020004 | NaN |
| **7052** | 2025-05-28 | 204.720001 | NaN |
| **7053** | 2025-05-29 | 205.699997 | NaN |
| **7054** | 2025-05-30 | 205.009995 | NaN |

7055 rows × 3 columns

👉 El parámetro `suffixes` en el método `.merge()` es de mucha utilidad para diferenciar columnas que pueden tener el mismo nombre en ambos DataFrames.

In [48]:
```
# Repetimos la unión anterior añadiendo sufijos
df_amzn[['Date','Close']].merge(df_aapl[['Date','Close']], how='left', left_on='Date', right_on='Date', suffixes=(' AMZN'
```

Out[48]:

| | Date | Close AMZN | Close AAPL |
|---|---|---|---|
| **0** | 1997-05-15 | 0.097917 | NaN |
| **1** | 1997-05-16 | 0.086458 | NaN |
| **2** | 1997-05-19 | 0.085417 | NaN |
| **3** | 1997-05-20 | 0.081771 | NaN |
| **4** | 1997-05-21 | 0.071354 | NaN |
| **...** | ... | ... | ... |
| **7050** | 2025-05-23 | 200.990005 | NaN |
| **7051** | 2025-05-27 | 206.020004 | NaN |
| **7052** | 2025-05-28 | 204.720001 | NaN |
| **7053** | 2025-05-29 | 205.699997 | NaN |
| **7054** | 2025-05-30 | 205.009995 | NaN |

7055 rows × 3 columns

In [49]:
```
# Importamos dos DataFrames
df_queen_tracks = pd.read_csv('./data/queen_tracks.csv')
df_queen_albums = pd.read_csv('./data/queen_albums.csv')
```

In [50]:
```
# Mostramos las primeras del filas del primer DataFrame
df_queen_tracks.head()
```

Out[50]:

| | Album ID | Track | Year | Track Duration | Key | Mode | Tempo | Time Signature | Loudness | Energy | Danceability | Valence | Acousticness | Instrum |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Doing Alright | 1973 | 249.21 | 9 | 1 | 93.294 | 4 | -12.408 | 0.293 | 0.341 | 0.178 | 0.75600 | |
| **1** | 1 | Great King Rat | 1973 | 343.01 | 4 | 0 | 135.276 | 4 | -12.363 | 0.851 | 0.350 | 0.428 | 0.04040 | |
| **2** | 1 | Jesus | 1973 | 224.17 | 11 | 0 | 114.756 | 4 | -7.077 | 0.740 | 0.356 | 0.531 | 0.15300 | |
| **3** | 1 | Keep Yourself Alive - 2011 Mix | 1973 | 226.72 | 2 | 1 | 134.204 | 4 | -9.498 | 0.721 | 0.419 | 0.593 | 0.25000 | |
| **4** | 1 | Liar | 1973 | 383.92 | 2 | 1 | 149.084 | 4 | -7.227 | 0.787 | 0.261 | 0.267 | 0.00279 | |

In [51]:
```
# Mostramos el segundo DataFrame
df_queen_albums
```

Out[51]:

| | Album ID | Album | Release Year | Producers | Genre | Spotify Album |
|---|---|---|---|---|---|---|
| **0** | 1 | Queen | 1973 | Queen, Roy Baker, John Anthony | Hard Rock | 5SSpz2RyyEhLt5FDymT4Ph |
| **1** | 2 | Queen II | 1974 | Queen, Roy Baker, Robin Cable | Hard Rock | 2RKEso6nin3nhRyAd36Omv |
| **2** | 3 | Sheer Heart Attack | 1974 | Queen, Roy Baker | Hard Rock | 5CooX2xg5YibepSfjbRFNT |
| **3** | 4 | A Night At The Opera | 1975 | Queen, Roy Baker | Hard Rock | 75eP8LZolyNBpqIRyB5pvB |
| **4** | 5 | A Day At The Races | 1976 | Queen | Hard Rock | 3f45rzbU4dYQBTV9v5RFBB |
| **5** | 6 | News of the World | 1977 | Queen | Hard Rock | 3TKTjR4E3LAMfRsPeRsNhT |
| **6** | 7 | Jazz | 1978 | Queen, Roy Baker | Hard Rock | 5X3rA8To5GDOeIWdQyMEcE |
| **7** | 8 | The Game | 1980 | Queen, Reinhold Mack | Pop Rock | 1h0j80HhdzIMsUGUFiVkqa |
| **8** | 9 | Flash Gordon | 1980 | Queen, Reinhold Mack, Brian May | Pop Rock | 2SS9qutxzz0XZf4zmoQVdx |
| **9** | 10 | Hot Space | 1982 | Queen, Reinhold Mack | Pop Rock | 0fZCqpTHYq2k89uG6pPTYE |
| **10** | 11 | The Works | 1984 | Queen, Reinhold Mack | Pop Rock | 0FbnXAGmgmWBmNthZSgm43 |
| **11** | 12 | A Kind of Magic | 1986 | Queen, Reinhold Mack | Pop Rock | 34xBXeJgmQrn1wQvhyVCsw |
| **12** | 13 | The Miracle | 1989 | Queen, David Richards | Pop Rock | 1v5l2sZRE5Rweew5PoNFP9 |
| **13** | 14 | Innuendo | 1991 | Queen, David Richards | Hard Rock | 5yAM3CcaXF6DPRJW3oL6Ya |
| **14** | 15 | Made in Heaven | 1995 | Queen, David Richards, Justin Shirley-Smith, J... | Pop Rock | 2SFIh6siY4KYBNPHAV7xaI |

In [52]:
```
# Unimos dos DataFrames por una columna en común ('Album ID')
df_queen_albums[['Album ID','Album']].merge(df_queen_tracks, how='right', left_on='Album ID', right_on='Album ID')
```

Out[52]:

| | Album ID | Album | Track | Year | Track Duration | Key | Mode | Tempo | Time Signature | Loudness | Energy | Danceability | Valence | Acousticne |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Queen | Doing Alright | 1973 | 249.21 | 9 | 1 | 93.294 | 4 | -12.408 | 0.2930 | 0.341 | 0.1780 | 0.756 |
| **1** | 1 | Queen | Great King Rat | 1973 | 343.01 | 4 | 0 | 135.276 | 4 | -12.363 | 0.8510 | 0.350 | 0.4280 | 0.040 |
| **2** | 1 | Queen | Jesus | 1973 | 224.17 | 11 | 0 | 114.756 | 4 | -7.077 | 0.7400 | 0.356 | 0.5310 | 0.153 |
| **3** | 1 | Queen | Keep Yourself Alive - 2011 Mix | 1973 | 226.72 | 2 | 1 | 134.204 | 4 | -9.498 | 0.7210 | 0.419 | 0.5930 | 0.250 |
| **4** | 1 | Queen | Liar | 1973 | 383.92 | 2 | 1 | 149.084 | 4 | -7.227 | 0.7870 | 0.261 | 0.2670 | 0.002 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **172** | 15 | Made in Heaven | Mother Love | 1995 | 286.17 | 7 | 0 | 184.109 | 4 | -7.862 | 0.4120 | 0.373 | 0.2370 | 0.229 |
| **173** | 15 | Made in Heaven | My Life Has Been Saved | 1995 | 195.39 | 7 | 1 | 204.165 | 4 | -7.344 | 0.5840 | 0.344 | 0.3090 | 0.082 |
| **174** | 15 | Made in Heaven | Too Much Love Will Kill You | 1995 | 259.21 | 7 | 1 | 145.259 | 4 | -7.600 | 0.3960 | 0.386 | 0.2110 | 0.582 |
| **175** | 15 | Made in Heaven | Untitled | 1995 | 1354.93 | 2 | 1 | 129.457 | 5 | -31.607 | 0.0124 | 0.165 | 0.0311 | 0.681 |
| **176** | 15 | Made in Heaven | You Don't Fool Me | 1995 | 324.85 | 4 | 0 | 121.965 | 4 | -6.494 | 0.8290 | 0.590 | 0.5790 | 0.103 |

177 rows × 18 columns

**Unir tablas sin campos en común (por índices)**

El método `.join()` permite combinar DataFrames utilizando sus índices como clave de unión, lo que significa que los valores de los índices en ambos DataFrames se encuentran alineados. Este método puede ser más sencillo de usar que `.merge()` cuando se trabaja principalmente con índices, ya que no requiere especificar explícitamente las columnas de unión.

La sintaxis para unir tablas por los índices es la siguiente:

```
df_1.join(df_2)
```

Por defecto, realiza una unión de tipo left join, es decir, incluye todas las filas del DataFrame izquierdo y sólo aquellas del DataFrame derecho que tengan un índice coincidente. Sin embargo, también se pueden especificar otros tipos de uniones, como inner, outer y right, con el parámetro `how`.

In [53]:
```
# Creamos un primer DataFrame seleccionando dos columnas
df_1 = df_queen_albums[['Album','Release Year']]
df_1
```

Out[53]:

| | Album | Release Year |
|---|---|---|
| **0** | Queen | 1973 |
| **1** | Queen II | 1974 |
| **2** | Sheer Heart Attack | 1974 |
| **3** | A Night At The Opera | 1975 |
| **4** | A Day At The Races | 1976 |
| **5** | News of the World | 1977 |
| **6** | Jazz | 1978 |
| **7** | The Game | 1980 |
| **8** | Flash Gordon | 1980 |
| **9** | Hot Space | 1982 |
| **10** | The Works | 1984 |
| **11** | A Kind of Magic | 1986 |
| **12** | The Miracle | 1989 |
| **13** | Innuendo | 1991 |
| **14** | Made in Heaven | 1995 |

In [54]:
```
# Creamos un segundo DataFrame seleccionando dos columnas
df_2 = df_queen_albums[['Producers','Genre']]
df_2
```

Out[54]:

| | Producers | Genre |
|---|---|---|
| **0** | Queen, Roy Baker, John Anthony | Hard Rock |
| **1** | Queen, Roy Baker, Robin Cable | Hard Rock |
| **2** | Queen, Roy Baker | Hard Rock |
| **3** | Queen, Roy Baker | Hard Rock |
| **4** | Queen | Hard Rock |
| **5** | Queen | Hard Rock |
| **6** | Queen, Roy Baker | Hard Rock |
| **7** | Queen, Reinhold Mack | Pop Rock |
| **8** | Queen, Reinhold Mack, Brian May | Pop Rock |
| **9** | Queen, Reinhold Mack | Pop Rock |
| **10** | Queen, Reinhold Mack | Pop Rock |
| **11** | Queen, Reinhold Mack | Pop Rock |
| **12** | Queen, David Richards | Pop Rock |
| **13** | Queen, David Richards | Hard Rock |
| **14** | Queen, David Richards, Justin Shirley-Smith, J... | Pop Rock |

In [55]:
```
# Unimos los dos DataFrames por sus índices
df_1.join(df_2)
```

Out[55]:

| | Album | Release Year | Producers | Genre |
|---|---|---|---|---|
| **0** | Queen | 1973 | Queen, Roy Baker, John Anthony | Hard Rock |
| **1** | Queen II | 1974 | Queen, Roy Baker, Robin Cable | Hard Rock |
| **2** | Sheer Heart Attack | 1974 | Queen, Roy Baker | Hard Rock |
| **3** | A Night At The Opera | 1975 | Queen, Roy Baker | Hard Rock |
| **4** | A Day At The Races | 1976 | Queen | Hard Rock |
| **5** | News of the World | 1977 | Queen | Hard Rock |
| **6** | Jazz | 1978 | Queen, Roy Baker | Hard Rock |
| **7** | The Game | 1980 | Queen, Reinhold Mack | Pop Rock |
| **8** | Flash Gordon | 1980 | Queen, Reinhold Mack, Brian May | Pop Rock |
| **9** | Hot Space | 1982 | Queen, Reinhold Mack | Pop Rock |
| **10** | The Works | 1984 | Queen, Reinhold Mack | Pop Rock |
| **11** | A Kind of Magic | 1986 | Queen, Reinhold Mack | Pop Rock |
| **12** | The Miracle | 1989 | Queen, David Richards | Pop Rock |
| **13** | Innuendo | 1991 | Queen, David Richards | Hard Rock |
| **14** | Made in Heaven | 1995 | Queen, David Richards, Justin Shirley-Smith, J... | Pop Rock |

## 3.4 - Pivotar tablas de datos

En ocasiones, las tablas de datos están estructuradas de una manera que no resulta adecuada para la exploración y el análisis que se desea realizar. En estos casos, el pivoteo de tablas es una solución para reorganizar la información y facilitar su manipulación.

A través de esta técnica, los datos pueden transformarse de un formato largo o *long* a un formato ancho o *wide*, o viceversa.

### Convertir datos de formato *long* a *wide*

El método `.pivot()` toma una tabla que puede estar en formato *long* y la transforma en un formato *wide*. En términos simples, permite reestructurar los datos cambiando filas a columnas.

```
df.pivot(index='index_columns', columns='new_columns', values='value_columns')
```

- `index` : columna o columnas que se utilizan como índice en la nueva tabla.
- `columns` : columna o columnas cuyos valores se convierten en las columnas de la nueva tabla.
- `values` : columna o columnas cuyos valores que se utilizan para completar las celdas de la nueva tabla. Si se omite, se utilizan todos los valores disponibles.

In [56]:
```
# Importamos un DataFrame
df_eu_countries = pd.read_csv('./data/european_countries_long.csv')
df_eu_countries
```

Out[56]:

| | Country | Year | Indicator | Value |
|---|---|---|---|---|
| **0** | Albania | 1990 | Population | 3286542.00 |
| **1** | Albania | 2000 | Population | 3089027.00 |
| **2** | Albania | 2010 | Population | 2913021.00 |
| **3** | Albania | 2020 | Population | 2837849.00 |
| **4** | Albania | 1990 | Life Expectancy | 73.14 |
| **...** | ... | ... | ... | ... |
| **571** | United Kingdom | 2020 | Life Expectancy | 80.35 |
| **572** | United Kingdom | 1990 | GDP | 10931693.89 |
| **573** | United Kingdom | 2000 | GDP | 16655348.77 |
| **574** | United Kingdom | 2010 | GDP | 24854825.96 |
| **575** | United Kingdom | 2020 | GDP | 26978065.92 |

576 rows × 4 columns

In [57]:
```
# Pivotamos el DataFrame de formato long a wide
df_eu_countries.pivot(index='Country', columns=['Indicator', 'Year'], values='Value')
```

Out[57]:

| Indicator | Population | | | | Life Expectancy | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Year | 1990 | 2000 | 2010 | 2020 | 1990 | 2000 | 2010 | 2020 | 1990 | 2000 | 2010 |
| Country | | | | | | | | | | | |
| Albania | 3286542.0 | 3089027.0 | 2913021.0 | 2837849.0 | 73.14 | 75.40 | 77.94 | 76.99 | 20285.54 | 34803.55 | 119269.27 |
| Andorra | 53569.0 | 66097.0 | 71519.0 | 77700.0 | NaN | NaN | NaN | 82.50 | 10289.90 | 14326.06 | 34499.26 |
| Armenia | 3556539.0 | 3168523.0 | 2946293.0 | 2805608.0 | 68.82 | 70.62 | 73.16 | 72.17 | 22568.63 | 19115.64 | 92602.86 |
| Austria | 7677850.0 | 8011566.0 | 8363404.0 | 8916864.0 | 75.57 | 78.13 | 80.58 | 81.19 | 1664633.86 | 1972896.25 | 3922751.07 |
| Azerbaijan | 7175200.0 | 8048600.0 | 9054332.0 | 10093121.0 | 62.35 | 64.89 | 69.53 | 66.87 | 88848.48 | 52726.16 | 529092.95 |
| Belarus | 10189348.0 | 9979610.0 | 9483836.0 | 9379952.0 | 70.84 | 68.91 | 70.40 | 72.46 | NaN | 127367.80 | 572319.05 |
| Belgium | 9967379.0 | 10251250.0 | 10895586.0 | 11538604.0 | 76.05 | 77.72 | 80.18 | 80.70 | 2053317.48 | 2367924.60 | 4814208.83 |
| Bosnia and Herzegovina | 4494310.0 | 4179350.0 | 3811088.0 | 3318407.0 | 72.35 | 74.50 | 77.07 | 76.22 | 77534.78 | 55677.73 | 171763.16 |
| Bulgaria | 8718289.0 | 8170172.0 | 7395599.0 | 6934015.0 | 71.64 | 71.66 | 73.51 | 73.66 | 206320.91 | 132459.90 | 507609.29 |
| Croatia | 4777368.0 | 4468302.0 | 4295427.0 | 4047680.0 | 72.17 | 72.81 | 76.48 | 77.72 | 256502.13 | 221284.15 | 588364.06 |
| Cyprus | 788500.0 | 948237.0 | 1129686.0 | 1237537.0 | 73.54 | 76.57 | 79.67 | 81.39 | 55911.30 | 99858.44 | 257999.40 |
| Czech Republic | 10333355.0 | 10255063.0 | 10474410.0 | 10697858.0 | 71.38 | 74.97 | 77.42 | 78.18 | 407289.51 | 618281.66 | 2090699.41 |
| Denmark | 5140939.0 | 5339616.0 | 5547683.0 | 5831404.0 | 74.81 | 76.59 | 79.10 | 81.60 | 1382472.86 | 1641587.39 | 3219952.79 |
| Estonia | 1569174.0 | 1396985.0 | 1331475.0 | 1329522.0 | 69.48 | 70.42 | 75.43 | 78.60 | NaN | 56865.80 | 195234.77 |
| Finland | 4986431.0 | 5176209.0 | 5363352.0 | 5529543.0 | 74.81 | 77.47 | 79.87 | 81.93 | 1414383.46 | 1260195.43 | 2494243.11 |
| France | 58044701.0 | 60921384.0 | 65030575.0 | 67571107.0 | 76.60 | 79.06 | 81.66 | 82.18 | 12691796.17 | 13656396.61 | 26451878.82 |
| Georgia | 4802000.0 | 4077131.0 | 3786695.0 | 3722716.0 | 68.39 | 69.58 | 72.13 | 72.76 | 75000.00 | 30574.75 | 124269.08 |
| Germany | 79433029.0 | 82211508.0 | 81776930.0 | 83160871.0 | 75.09 | 77.93 | 79.99 | 81.04 | 17716712.07 | 19479819.91 | 33996678.20 |
| Greece | 10196792.0 | 10805808.0 | 11121341.0 | 10698599.0 | 76.94 | 77.89 | 80.39 | 81.29 | 978910.92 | 1304577.57 | 2971249.62 |
| Hungary | 10373988.0 | 10210971.0 | 10000023.0 | 9750149.0 | 69.32 | 71.25 | 74.21 | 75.57 | 343658.92 | 472184.06 | 1321753.50 |
| Iceland | 254826.0 | 281205.0 | 318041.0 | 366463.0 | 78.04 | 79.65 | 81.90 | 83.06 | 64687.36 | 90256.60 | 137511.62 |
| Ireland | 3513974.0 | 3805174.0 | 4560155.0 | 4985382.0 | 74.85 | 76.54 | 80.74 | 82.56 | 493056.32 | 1002076.10 | 2219135.61 |
| Italy | 56719240.0 | 56942108.0 | 59277417.0 | 59438851.0 | 76.97 | 79.78 | 82.04 | 82.20 | 11812226.54 | 11466768.94 | 21360999.55 |
| Latvia | 2663151.0 | 2367550.0 | 2097555.0 | 1900449.0 | 69.27 | 70.31 | 73.48 | 75.19 | NaN | 79588.53 | 239561.63 |
| Liechtenstein | 28765.0 | 33026.0 | 35926.0 | 38756.0 | NaN | 76.83 | 81.84 | 81.66 | 14215.09 | 24838.90 | 50823.37 |
| Lithuania | 3697838.0 | 3499536.0 | 3097282.0 | 2794885.0 | 71.16 | 72.02 | 73.27 | 74.98 | NaN | 115247.77 | 371286.94 |
| Luxembourg | 381850.0 | 436300.0 | 506953.0 | 630419.0 | 75.44 | 77.87 | 80.63 | 82.14 | 127787.93 | 212301.83 | 562139.86 |
| Malta | 354170.0 | 390087.0 | 414508.0 | 515332.0 | 75.88 | 78.35 | 81.40 | 82.35 | 25471.64 | 40695.16 | 90358.24 |
| Moldova | 2965978.0 | 2924668.0 | 2862354.0 | 2635130.0 | 68.07 | 66.42 | 69.36 | 70.17 | 35712.50 | 12884.29 | 69749.82 |
| Monaco | 30329.0 | 32465.0 | 33178.0 | 36922.0 | NaN | NaN | NaN | 86.09 | 24813.07 | 26544.63 | 53675.62 |
| Montenegro | 606372.0 | 604950.0 | 619428.0 | 621306.0 | 75.57 | 73.82 | 75.99 | 75.93 | NaN | 9842.93 | 41429.84 |
| Netherlands | 14951510.0 | 15925513.0 | 16615394.0 | 17441500.0 | 76.88 | 77.99 | 80.70 | 81.36 | 3183305.12 | 4174793.37 | 8473808.59 |
| North Macedonia | 2044174.0 | 2026350.0 | 1946298.0 | 1856124.0 | 71.23 | 72.95 | 75.00 | 74.40 | 46996.47 | 37728.59 | 94071.70 |
| Norway | 4241473.0 | 4490967.0 | 4889252.0 | 5379475.0 | 76.54 | 78.63 | 81.00 | 83.21 | 1197918.43 | 1714572.02 | 4310521.44 |
| Poland | 38110782.0 | 38258629.0 | 38042794.0 | 37899070.0 | 70.89 | 73.75 | 76.25 | 76.50 | 659777.48 | 1722204.52 | 4756966.14 |
| Portugal | 9983218.0 | 10289898.0 | 10573100.0 | 10297081.0 | 73.97 | 76.31 | 79.03 | 80.98 | 787138.60 | 1186051.93 | 2381130.03 |
| Romania | 23201835.0 | 22442971.0 | 20246871.0 | 19265250.0 | 69.74 | 71.16 | 73.46 | 74.25 | 382478.82 | 372537.40 | 1700293.59 |
| Russia | 147969406.0 | 146596869.0 | 142849468.0 | 145245148.0 | 68.89 | 65.48 | 68.84 | 71.34 | 5170144.46 | 2597101.42 | 15249167.15 |
| San Marino | 23132.0 | 26823.0 | 31608.0 | 34007.0 | NaN | NaN | NaN | 82.65 | NaN | 10051.59 | 18811.92 |
| Serbia | 7586000.0 | 7516346.0 | 7291436.0 | 6899126.0 | 70.23 | 71.58 | 74.34 | 74.48 | NaN | 68758.46 | 418194.69 |
| Slovakia | 5299187.0 | 5388720.0 | 5391428.0 | 5458827.0 | 70.93 | 73.05 | 75.11 | 76.87 | 127473.81 | 292425.59 | 911628.36 |
| Slovenia | 1998161.0 | 1988925.0 | 2048583.0 | 2102419.0 | 73.20 | 75.41 | 79.42 | 80.53 | 198179.23 | 202896.28 | 482082.40 |

| Indicator | Population | | | | Life Expectancy | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Year | 1990 | 2000 | 2010 | 2020 | 1990 | 2000 | 2010 | 2020 | 1990 | 2000 | 2010 |
| Country | | | | | | | | | | | |
| **Spain** | 38867322.0 | 40567864.0 | 46576897.0 | 47365655.0 | 76.84 | 78.97 | 81.63 | 82.33 | 5365585.91 | 5983633.13 | 14221082.00 |
| **Sweden** | 8558835.0 | 8872109.0 | 9378126.0 | 10353442.0 | 77.54 | 79.64 | 81.45 | 82.36 | 2618461.94 | 2628354.54 | 4958125.59 |
| **Switzerland** | 6715519.0 | 7184250.0 | 7824909.0 | 8638167.0 | 77.24 | 79.68 | 82.25 | 83.00 | 2657635.74 | 2792160.34 | 5988510.29 |
| **Turkey** | 54324142.0 | 64113547.0 | 73142150.0 | 83384680.0 | 67.71 | 71.86 | 75.07 | 75.85 | 1506555.00 | 2742946.23 | 7769672.66 |
| **Ukraine** | 51891400.0 | 49176500.0 | 45870741.0 | 44207754.0 | 70.10 | 67.68 | 70.27 | 71.19 | 795238.10 | 323750.84 | 1412091.70 |
| **United Kingdom** | 57247586.0 | 58892514.0 | 62766365.0 | 67081234.0 | 75.88 | 77.74 | 80.40 | 80.35 | 10931693.89 | 16655348.77 | 24854825.96 |

## Convertir datos de formato *wide* a *long*

El método `.melt()` se utiliza para transformar un DataFrame de un formato *wide* a un formato *long*. Es decir, permite desagregar columnas en filas.

```
df.melt(id_vars='id_columns', value_vars='value_columns', var_name='variable_name',
value_name='value_name')
```

- `id_vars` : columna o columnas que se mantienen fijas en la nueva tabla. Si no se especifican, se utilizan todas las columnas que no están en `value_vars` .
- `value_vars` : columna o columnas que se transforman en una única columna en la nueva. Si no se especifican, se utilizan todas las columnas que no están en `id_vars` .
- `var_name` : nombre de la nueva columna que contienen los nombres de las columnas transformadas en una única. Por defecto, se utiliza el nombre "variable".
- `value_name` : nombre de la nueva columna que contienen los valores de las columnas transformadas en una única. Por defecto, se utiliza el nombre "value".

In [58]:
```
# Importamos un DataFrame
df_eu_countries = pd.read_csv('./data/european_countries_wide.csv')
df_eu_countries.head()
```

Out[58]:

| | Country | Population 1990 | Population 2000 | Population 2010 | Population 2020 | Life Expectancy 1990 | Life Expectancy 2000 | Life Expectancy 2010 | Life Expectancy 2020 | GDP 1990 | GDP 2( |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Albania | 3286542.0 | 3089027.0 | 2913021.0 | 2837849.0 | 73.144000 | 75.404000 | 77.936000 | 76.989000 | 20285.54 | 34803 |
| **1** | Andorra | 53569.0 | 66097.0 | 71519.0 | 77700.0 | NaN | NaN | NaN | 82.500000 | 10289.90 | 14326 |
| **2** | Armenia | 3556539.0 | 3168523.0 | 2946293.0 | 2805608.0 | 68.821000 | 70.624000 | 73.160000 | 72.173000 | 22568.63 | 19115 |
| **3** | Austria | 7677850.0 | 8011566.0 | 8363404.0 | 8916864.0 | 75.568293 | 78.126829 | 80.580488 | 81.192683 | 1664633.86 | 1972896 |
| **4** | Azerbaijan | 7175200.0 | 8048600.0 | 9054332.0 | 10093121.0 | 62.352000 | 64.891000 | 69.529000 | 66.868000 | 88848.48 | 52726 |

In [59]:
```
# Pivotamos el DataFrame de formato wide a long
df_eu_countries.melt(id_vars = 'Country', var_name = 'Indicator', value_name = 'Value')
```

Out[59]:

| | Country | Indicator | Value |
|---|---|---|---|
| **0** | Albania | Population 1990 | 3286542.00 |
| **1** | Andorra | Population 1990 | 53569.00 |
| **2** | Armenia | Population 1990 | 3556539.00 |
| **3** | Austria | Population 1990 | 7677850.00 |
| **4** | Azerbaijan | Population 1990 | 7175200.00 |
| **...** | ... | ... | ... |
| **571** | Sweden | GDP 2020 | 5470541.74 |
| **572** | Switzerland | GDP 2020 | 7419994.06 |
| **573** | Turkey | GDP 2020 | 7203384.98 |
| **574** | Ukraine | GDP 2020 | 1566177.22 |
| **575** | United Kingdom | GDP 2020 | 26978065.92 |

576 rows × 3 columns

In [60]:
```
# Pivotamos los datos de formato wide a long especificando columnas a convertir en filas
df_eu_countries.melt(id_vars = 'Country', value_vars = ['Population 1990','Population 2000','Population 2010','Population
                     var_name = 'Indicator', value_name = 'Value')
```

Out[60]:

| | Country | Indicator | Value |
|---|---|---|---|
| **0** | Albania | Population 1990 | 3286542.0 |
| **1** | Andorra | Population 1990 | 53569.0 |
| **2** | Armenia | Population 1990 | 3556539.0 |
| **3** | Austria | Population 1990 | 7677850.0 |
| **4** | Azerbaijan | Population 1990 | 7175200.0 |
| **...** | ... | ... | ... |
| **187** | Sweden | Population 2020 | 10353442.0 |
| **188** | Switzerland | Population 2020 | 8638167.0 |
| **189** | Turkey | Population 2020 | 83384680.0 |
| **190** | Ukraine | Population 2020 | 44207754.0 |
| **191** | United Kingdom | Population 2020 | 67081234.0 |

192 rows × 3 columns

## 3.5 - Ejercicios

📘 Puedes encontrar las soluciones a los ejercicios aquí.

### Ejercicio 3.1

> Dataset a utilizar: `fortune_1000.csv`

**3.1A:** Calcula la suma total de la facturación ('Revenue') por sector.

**3.1B:** Repite el apartado anterior, pero filtrando únicamente por los sectores 'Retailing', 'Technology' y 'Wholesalers'.

#### Ejercicio 3.1A

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

#### Ejercicio 3.1B

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.2

> Dataset a utilizar: `star_wars.csv`

Extrae los 5 planetas ('Homeworld') que contienen el mayor número de personajes incluidos en el dataset.

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.3

> Dataset a utilizar: `world_countries.csv`

Calcula las siguientes métricas por continente:

- Población total.
- Esperanza de vida media.
- Producto Interior Bruto (PIB) per cápita medio ('GDP' / 'Population').

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.4

> Dataset a utilizar: `imdb_movies.csv`

**3.4A:** De los 10 países con más películas producidas en el siglo XXI, crea un DataFrame que incluya el conteo total de películas agrupadas por país, así como la puntuación media y máxima de IMDb, y la duración media y máxima de las películas.

**3.4B:** Lista la cantidad de películas disponibles en cada uno de los idiomas presentes en los datos.

#### Ejercicio 3.4A

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

#### Ejercicio 3.4B

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.5

> Dataset a utilizar: `laliga_results.csv`

**3.5A:** Añade un nuevo campo que indique el día de la semana, expresado en texto, correspondiente a cada partido.

**3.5B:** Muestra el total de goles que se han marcado en cada uno de los 7 días de la semana.

#### Ejercicio 3.5A

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

#### Ejercicio 3.5B

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.6

> Dataset a utilizar: `amzn_stock.csv`

**3.6A:** Crea un DataFrame que permita calcular la media mensual del valor de cotización de apertura ('Open') de los últimos dos años.

**3.6B:** Identifica la semana con mejor rendimiento en términos de cotización de cierre, calculando el mayor valor promedio de 'Close'.

**3.6C:** Repite el apartado anterior, pero esta vez calcula la semana con el mayor aumento porcentual en el valor de cotización de cierre en comparación con la semana anterior.

#### Ejercicio 3.6A

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

#### Ejercicio 3.6B

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

#### Ejercicio 3.6C

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.7

> Datasets a utilizar: `queen_albums.csv` & `queen_tracks.csv`

Crea un DataFrame en el que se integre la información de los álbumes y las canciones de Queen, mostrando para cada álbum la duración total de sus canciones ('Track Duration'), el valor promedio de energía ('Energy') y el promedio de "danceability" ('Danceability').

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.8

> Dataset a utilizar: `laliga_results.csv`

Crea la clasificación final de LaLiga 2023-2024 que muestre los puntos acumulados por cada uno de equipos a lo largo de la competición.

Pasos para realizar el ejercicio:

1. Crea un DataFrame que contenga los puntos ganados por los equipos locales ('HT Points') y otro con los puntos ganados por los equipos visitantes ('AT Points').
2. Combina los dos DataFrames creados en el paso anterior.
3. Agrupa el DataFrame combinado por equipo y suma los puntos totales ganados por cada uno.

In [ ]:
```
# Escribe la solución al ejercicio aquí
```

### Ejercicio 3.9

> Dataset a utilizar: `laliga_results.csv`

Crea una matriz de resultados que muestre los resultados de todos los partidos de LaLiga entre los 20 equipos.

Pasos para realizar el ejercicio:

1. Crea una nueva columna en la que se combinen los goles locales y los visitantes, mostrando los resultados de los partidos en el formato "goles_local - goles_visitante".
2. Pivota el DataFrame para crear una matriz donde las filas representen los equipos locales ('Home Team'), las columnas representen los equipos visitantes ('Away Team') y las celdas contengan los resultados de los partidos (la columna creada).

In [ ]:
```
# Escribe la solución al ejercicio aquí
```