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 - Soluciones a Ejercicios

Este notebook incluye las soluciones a los ejercicios del capítulo 3: Manipulación de Datos de la Introducción a Python para el Análisis de Datos.

3. Ejercicios resueltos

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

In [2]:
```
# Importamos el DataFrame de empresas estadounidenses
df_fortune = pd.read_csv('./data/fortune_1000.csv')
df_fortune.head()
```

Out[2]:

	Rank	Company	Sector	Industry	Location	Employees	Revenue	Profits	Change in Rank
0	1	Walmart	Retailing	General Merchandisers	Bentonville, AR	2100000	648125.0	15511.0	0.0
1	2	Amazon	Retailing	Internet Services and Retailing	Seattle, WA	1525000	574785.0	30425.0	0.0
2	3	Apple	Technology	Computers, Office Equipment	Cupertino, CA	161000	383285.0	96995.0	1.0
3	4	UnitedHealth Group	Health Care	Health Care: Insurance and Managed Care	Minnetonka, MN	440000	371622.0	22381.0	1.0
4	5	Berkshire Hathaway	Financials	Insurance: Property and Casualty (Stock)	Omaha, NE	396500	364482.0	96223.0	2.0

In [3]:
```
# Agrupamos por sector y calculamos la suma de ingresos para cada uno, ordenando de mayor a menor
df_fortune.groupby('Sector').agg({'Revenue':'sum'}).sort_values(by='Revenue', ascending=False)
```

Out[3]:

Sector	Revenue
Financials	3735496.8
Health Care	3104833.3
Retailing	2650287.8
Technology	2420894.6
Energy	2289632.9
Food, Beverages & Tobacco	697337.6
Transportation	611703.5
Motor Vehicles & Parts	611661.3
Industrials	574132.1
Wholesalers	512623.2
Business Services	491570.8
Telecommunications	484301.7
Food & Drug Stores	469792.3
Materials	402390.6
Aerospace & Defense	389260.7
Media	327392.8
Engineering & Construction	288798.4
Chemicals	285942.3
Hotels, Restaurants & Leisure	249135.1
Household Products	237277.2
Apparel	130599.2

Ejercicio 3.1B

In [4]:
```
# Creamos la condición para filtrar los sectores 'Retailing', 'Technology' y 'Wholesalers'
cond = df_fortune['Sector'].isin(['Retailing', 'Technology', 'Wholesalers'])

# Repetimos la agregación anterior aplicando la condición
df_fortune[cond].groupby('Sector').agg({'Revenue':'sum'}).sort_values(by='Revenue', ascending=False)
```

Out[4]:

Sector	Revenue
Retailing	2650287.8
Technology	2420894.6
Wholesalers	512623.2

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 [5]:
```
# Importamos el DataFrame de Star Wars
df_starwars = pd.read_csv('./data/star_wars.csv')
df_starwars.head()
```

Out[5]:

	Name	Birth Year	Homeworld	Species	Gender	Height	Mass	Skin Color	Hair Color	Eye Color	Films
0	Luke Skywalker	19.0	Tatooine	Human	male	172.0	77.0	fair	blond	blue	A New Hope, The Empire Strikes Back, Return of...
1	C-3PO	112.0	Tatooine	Droid	none	167.0	75.0	gold	none	yellow	A New Hope, The Empire Strikes Back, Return of...
2	R2-D2	33.0	Naboo	Droid	none	96.0	32.0	white, blue	none	red	A New Hope, The Empire Strikes Back, Return of...
3	Darth Vader	41.9	Tatooine	Human	male	202.0	136.0	white	none	yellow	A New Hope, The Empire Strikes Back, Return of...
4	Leia Organa	19.0	Alderaan	Human	female	150.0	49.0	light	brown	brown	A New Hope, The Empire Strikes Back, Return of...

In [6]:
```
# Agrupamos por planeta de origen y contamos el número de personajes en cada uno
df_starwars.groupby('Homeworld').agg({'Name':'count'})\
    .rename(columns={'Name':'Character Count'})\
    .sort_values(by='Character Count', ascending=False)\
    .head(5)
```

Out[6]:

Homeworld	Character Count
Naboo	11
Tatooine	9
Alderaan	3
Kamino	3
Coruscant	3

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 [7]:
```
# Importamos el DataFrame de países
df_countries = pd.read_csv('./data/world_countries.csv')
df_countries.head()
```

Out[7]:

	Country	Continent	Population	Life Expectancy	GDP (Millions)
0	Afghanistan	Asia	42239854	62.879	14502.16
1	Albania	Europe	2745972	76.833	22977.68
2	Algeria	Africa	45606480	77.129	239899.49
3	Andorra	Europe	80088	NaN	3727.67
4	Angola	Africa	36684202	61.929	84722.96

In [8]:
```
# Creamos una nueva columna que es el cálculo del PIB per cápita
df_countries['GDP per Capita'] = (df_countries['GDP (Millions)']*1000000 / df_countries['Population']).round(2)
```

In [9]:
```
# Agrupamos por continente y calculamos las métricas especificadas
df_countries.groupby('Continent').agg({'Population':'sum',
                                       'Life Expectancy':'mean',
                                       'GDP per Capita':'mean'}).round(2)
```

Out[9]:

Continent	Population	Life Expectancy	GDP per Capita
Africa	1458544005	63.26	2736.69
America	1035383058	73.49	15419.81
Asia	4690811271	74.14	14999.85
Europe	756924746	78.46	44311.48
Oceania	44869303	70.50	12956.33

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 [10]:
```
# Importamos el DataFrame de películas
df_movies = pd.read_csv('./data/imdb_movies.csv')
df_movies.head()
```

Out[10]:

	movie_title	title_year	duration	country	language	genres	content_rating	color	aspect_ratio	gross
0	Avatar	2009.0	178.0	USA	English	Action|Adventure|Fantasy|Sci-Fi	PG-13	Color	1.78	760505847.0
1	Pirates of the Caribbean: At World's End	2007.0	169.0	USA	English	Action|Adventure|Fantasy	PG-13	Color	2.35	309404152.0
2	Spectre	2015.0	148.0	UK	English	Action|Adventure|Thriller	PG-13	Color	2.35	200074175.0
3	The Dark Knight Rises	2012.0	164.0	USA	English	Action|Thriller	PG-13	Color	2.35	448130642.0
4	Star Wars: Episode VII - The Force Awakens ...	NaN	NaN	NaN	NaN	Documentary	NaN	NaN	NaN	NaN

5 rows × 28 columns

In [11]:
```
# Filtramos el DataFrame para incluir sólo las películas del siglo XXI
movies_XXI = df_movies[df_movies['title_year'] > 2000]
```

In [12]:
```
# Agrupamos por país y calculamos las métricas especificadas
movies_XXI.groupby('country').agg({'movie_title':'nunique', 'imdb_score':['mean','max'], 'duration':['mean','max']}).roun
    .sort_values(by=('movie_title', 'nunique'), ascending=False).head(10) # Ordenamos por conteo de películas
```

Out[12]:

country	movie_title nunique	imdb_score mean	imdb_score max	duration mean	duration max
USA	2453	6.22	9.1	105.52	280.0
UK	280	6.72	8.2	107.53	180.0
France	122	6.65	8.5	105.46	163.0
Canada	95	6.03	8.2	100.02	141.0
Germany	76	6.29	8.5	111.29	206.0
Australia	32	6.51	8.1	107.44	165.0
India	31	6.62	8.5	139.45	193.0
Spain	29	6.86	8.2	103.72	145.0
China	27	6.58	7.9	111.28	150.0
Mexico	15	6.69	7.7	111.33	148.0

Ejercicio 3.4B

In [13]:
```
# Calculamos la cantidad de películas disponibles en cada idioma
df_movies['language'].value_counts()
```

Out[13]:
```
language
English       4704
French          73
Spanish         40
Hindi           28
Mandarin        26
German          19
Japanese        18
Cantonese       11
Russian         11
Italian         11
Portuguese       8
Korean           8
Danish           5
Arabic           5
Hebrew           5
Swedish          5
Polish           4
Norwegian        4
Persian          4
Dutch            4
Chinese          3
Thai             3
Icelandic        2
Aboriginal       2
Indonesian       2
Zulu             2
Romanian         2
Dari             2
Panjabi          1
Vietnamese       1
Slovenian        1
Greek            1
Dzongkha         1
Tamil            1
Urdu             1
Telugu           1
Kannada          1
Czech            1
Hungarian        1
Bosnian          1
Filipino         1
Mongolian        1
Maya             1
Aramaic          1
Kazakh           1
Swahili          1
Name: count, dtype: int64
```

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 [14]:
```
# Importamos el DataFrame con los resultados de LaLiga
df_laliga = pd.read_csv('./data/laliga_results.csv')
df_laliga.head()
```

Out[14]:

	Date	Home Team	Away Team	HT Goals	AT Goals	HT Points	AT Points
0	11/08/2023	Almeria	Vallecano	0	2	0	3
1	11/08/2023	Sevilla	Valencia	1	2	0	3
2	12/08/2023	Real Sociedad	Girona	1	1	1	1
3	12/08/2023	Las Palmas	Mallorca	1	1	1	1
4	12/08/2023	Ath. Bilbao	Real Madrid	0	2	0	3

In [15]:
```
# Consultamos los tipos de variables del DataFrame
df_laliga.dtypes
```

Out[15]:
```
Date         object
Home Team    object
Away Team    object
HT Goals      int64
AT Goals      int64
HT Points     int64
AT Points     int64
dtype: object
```

In [16]:
```
# Convertimos 'Date' a tipo fecha
df_laliga['Date'] = pd.to_datetime(df_laliga['Date'], format='%d/%m/%Y')
df_laliga.dtypes
```

Out[16]:
```
Date         datetime64[ns]
Home Team            object
Away Team            object
HT Goals              int64
AT Goals              int64
HT Points             int64
AT Points             int64
dtype: object
```

In [17]:
```
# Creamos una columna con el día de la semana
df_laliga['Week Day'] = df_laliga['Date'].dt.day_name()
df_laliga.head()
```

Out[17]:

	Date	Home Team	Away Team	HT Goals	AT Goals	HT Points	AT Points	Week Day
0	2023-08-11	Almeria	Vallecano	0	2	0	3	Friday
1	2023-08-11	Sevilla	Valencia	1	2	0	3	Friday
2	2023-08-12	Real Sociedad	Girona	1	1	1	1	Saturday
3	2023-08-12	Las Palmas	Mallorca	1	1	1	1	Saturday
4	2023-08-12	Ath. Bilbao	Real Madrid	0	2	0	3	Saturday

Ejercicio 3.5B

In [18]:
```
# Creamos una columna con los goles totales de cada partido
df_laliga['Total Goals'] = df_laliga['HT Goals'] + df_laliga['AT Goals']
```

In [19]:
```
# Agrupamos por el día de la semana para calcular la suma total de goles para cada día
df_laliga.groupby('Week Day').agg({'Total Goals':'sum'}).sort_values(by='Total Goals', ascending=False)
```

Out[19]:

Week Day	Total Goals
Sunday	369
Saturday	347
Friday	91
Monday	82
Wednesday	50
Tuesday	36
Thursday	30

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 [20]:
```
# Importamos el DataFrame con las cotizaciones de Amazon
df_amzn = pd.read_csv('./data/amzn_stock.csv')
df_amzn.head()
```

Out[20]:

	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

In [21]:
```
# Consultamos los tipos de variables del DataFrame
df_amzn.dtypes
```

Out[21]:
```
Date       object
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
dtype: object
```

In [22]:
```
# Convertimos 'Date' a tipo fecha
df_amzn['Date'] = pd.to_datetime(df_amzn['Date'])
df_amzn.dtypes
```

Out[22]:
```
Date      datetime64[ns]
Open             float64
High             float64
Low              float64
Close            float64
Volume             int64
dtype: object
```

In [23]:
```
# Creamos los campos de año y mes
df_amzn['Year'] = df_amzn['Date'].dt.year
df_amzn['Month'] = df_amzn['Date'].dt.month
```

In [24]:
```
# Definimos condiciones para incluir sólo los registros de 2024 y 2025
cond_2024 = df_amzn['Year'] == 2024
cond_2025 = df_amzn['Year'] == 2025
```

```
# Filtramos por los años 2024 y 2025, y luego agrupamos por año y mes, calculando la media de los valores de apertura
df_amzn[cond_2024 | cond_2025].groupby(['Year', 'Month']).agg({'Open':'mean'})
```

Out[24]:

Year	Month	Open
2023	1	93.452000
	2	99.101579
	3	96.298262
	4	103.405791
	5	111.595454
	6	126.193810
	7	130.961000
	8	135.750000
	9	135.616000
	10	128.159546
	11	142.932382
	12	149.524500
2024	1	153.435714
	2	170.256000
	3	176.908499
	4	181.773182
	5	184.290001
	6	185.430527
	7	191.142272
	8	172.489999
	9	184.942000
	10	187.005654
	11	203.901998
	12	223.817620

Ejercicio 3.6B

In [25]:
```
# Creamos un campo con el número de semana y otro con el año
df_amzn['Week Number'] = df_amzn['Date'].dt.isocalendar().week
df_amzn['Year'] = df_amzn['Date'].dt.isocalendar().year
df_amzn.head()
```

Out[25]:

	Date	Open	High	Low	Close	Volume	Year	Month	Week Number
0	1997-05-15	0.121875	0.125000	0.096354	0.097917	1443120000	1997	5	20
1	1997-05-16	0.098438	0.098958	0.085417	0.086458	294000000	1997	5	20
2	1997-05-19	0.088021	0.088542	0.081250	0.085417	122136000	1997	5	21
3	1997-05-20	0.086458	0.087500	0.081771	0.081771	109344000	1997	5	21
4	1997-05-21	0.081771	0.082292	0.068750	0.071354	377064000	1997	5	21

In [26]:
```
# Calculamos el promedio de 'Close', agrupando por año y número de la semana
amzn_weekly_close = df_amzn.groupby(['Year','Week Number']).agg({'Close':'mean'})

# Obtenemos la semana con el mayor valor medio de cierre de cotización
top_week = amzn_weekly_close.sort_values(by='Close', ascending=False).head(1)
top_week
```

Out[26]:

Year	Week Number	Close
2025	6	236.725998

In [27]:
```
# Almacenamos el año, el número de la semana y el valor de cierre en variables
year = top_week.index[0][0]
week_number = top_week.index[0][1]
close_value = top_week['Close'].values[0]

# Imprimimos el resultado
print(f"El mayor cierre promedio se registró en la semana {week_number} del año {year}, con un valor de {close_value:.2f}")
```

```
El mayor cierre promedio se registró en la semana 6 del año 2025, con un valor de 236.73.
```

Ejercicio 3.6C

In [28]:
```
# Mostramos el DataFrame en el que agrupamos por año y número de la semana
amzn_weekly_close.head()
```

Out[28]:

		Close
Year	**Week Number**	
1997	**20**	0.092187
	21	0.076667
	22	0.076498
	23	0.076042
	24	0.080000

In [29]:
```
# Creamos una nueva columna 'Last Week Close' con el valor de cierre de la semana anterior utilizando el método '.shift()'
amzn_weekly_close['Last Week Close'] = amzn_weekly_close['Close'].shift(1)
amzn_weekly_close.head()
```

Out[29]:

		Close	Last Week Close
Year	**Week Number**		
1997	**20**	0.092187	NaN
	21	0.076667	0.092187
	22	0.076498	0.076667
	23	0.076042	0.076498
	24	0.080000	0.076042

In [30]:
```
# Calculamos la tasa de crecimiento entre semana y semana
amzn_weekly_close['WOW Growth Rate'] = ((amzn_weekly_close['Close'] / amzn_weekly_close['Last Week Close']) - 1).round(2)
amzn_weekly_close.head()
```

Out[30]:

		Close	Last Week Close	WOW Growth Rate
Year	**Week Number**			
1997	**20**	0.092187	NaN	NaN
	21	0.076667	0.092187	-0.17
	22	0.076498	0.076667	-0.00
	23	0.076042	0.076498	-0.01
	24	0.080000	0.076042	0.05

In [31]:
```
# Obtenemos la semana con la mayor tasa de crecimiento
top_growth_week = amzn_weekly_close.sort_values(by='WOW Growth Rate', ascending=False).head(1)
top_growth_week
```

Out[31]:

		Close	Last Week Close	WOW Growth Rate
Year	**Week Number**			
2001	**15**	0.63975	0.4362	0.47

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 [32]:
```
# Importamos el DataFrame de canciones de Queen
df_queen_tracks = pd.read_csv('./data/queen_tracks.csv')
df_queen_tracks.head(3)
```

Out[32]:

	Album ID	Track	Year	Track Duration	Key	Mode	Tempo	Time Signature	Loudness	Energy	Danceability	Valence	Acousticness	Instrume
0	1	Doing Alright	1973	249.21	9	1	93.294	4	-12.408	0.293	0.341	0.178	0.7560	
1	1	Great King Rat	1973	343.01	4	0	135.276	4	-12.363	0.851	0.350	0.428	0.0404	
2	1	Jesus	1973	224.17	11	0	114.756	4	-7.077	0.740	0.356	0.531	0.1530	

In [33]:
```
# Importamos el DataFrame de álbumes de Queen
df_queen_albums = pd.read_csv('./data/queen_albums.csv')
df_queen_albums.head(3)
```

Out[33]:

	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

In [34]:
```
# Unimos los DataFrames para combinar la información de cada álbum con los datos de sus canciones
df_queen_merged = df_queen_albums[['Album ID', 'Album']].merge(df_queen_tracks[['Album ID', 'Track', 'Track Duration', 'E
                                                               how='left', left_on='Album ID', right_on='Album ID')
df_queen_merged
```

Out[34]:

	Album ID	Album	Track	Track Duration	Energy	Danceability
0	1	Queen	Doing Alright	249.21	0.2930	0.341
1	1	Queen	Great King Rat	343.01	0.8510	0.350
2	1	Queen	Jesus	224.17	0.7400	0.356
3	1	Queen	Keep Yourself Alive - 2011 Mix	226.72	0.7210	0.419
4	1	Queen	Liar	383.92	0.7870	0.261
...	...	...	...	...	...	...
172	15	Made in Heaven	Mother Love	286.17	0.4120	0.373
173	15	Made in Heaven	My Life Has Been Saved	195.39	0.5840	0.344
174	15	Made in Heaven	Too Much Love Will Kill You	259.21	0.3960	0.386
175	15	Made in Heaven	Untitled	1354.93	0.0124	0.165
176	15	Made in Heaven	You Don't Fool Me	324.85	0.8290	0.590

177 rows × 6 columns

In [35]:
```
# Agrupamos por álbum para calcular la duración total y el promedio de 'Energy' y 'Danceability'
df_queen_merged.groupby(['Album ID', 'Album']).agg({'Track Duration':'sum', 'Energy':'mean', 'Danceability':'mean'}).roun
    .rename(columns={'Track Duration':'Total Duration', 'Energy':'AVG Energy', 'Danceability':'AVG Danceability'})
```

Out[35]:

Album ID	Album	Total Duration	AVG Energy	AVG Danceability
1	Queen	2322.53	0.66	0.35
2	Queen II	2443.36	0.64	0.33
3	Sheer Heart Attack	2336.53	0.63	0.43
4	A Night At The Opera	2584.79	0.56	0.40
5	A Day At The Races	2654.86	0.57	0.40
6	News of the World	2355.06	0.57	0.50
7	Jazz	2680.15	0.66	0.50
8	The Game	2137.04	0.70	0.62
9	Flash Gordon	2514.07	0.49	0.35
10	Hot Space	2614.39	0.63	0.68
11	The Works	2454.22	0.69	0.53
12	A Kind of Magic	3022.23	0.67	0.46
13	The Miracle	2806.81	0.75	0.52
14	Innuendo	3232.62	0.70	0.43
15	Made in Heaven	4222.15	0.52	0.38

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 [36]:
```
# Importamos el DataFrame con los resultados de LaLiga
df_laliga = pd.read_csv('./data/laliga_results.csv')
df_laliga
```

Out[36]:

	Date	Home Team	Away Team	HT Goals	AT Goals	HT Points	AT Points
0	11/08/2023	Almeria	Vallecano	0	2	0	3
1	11/08/2023	Sevilla	Valencia	1	2	0	3
2	12/08/2023	Real Sociedad	Girona	1	1	1	1
3	12/08/2023	Las Palmas	Mallorca	1	1	1	1
4	12/08/2023	Ath. Bilbao	Real Madrid	0	2	0	3
...	...	...	...	...	...	...	...
375	25/05/2024	Real Madrid	Betis	0	0	1	1
376	26/05/2024	Getafe	Mallorca	1	2	0	3
377	26/05/2024	Celta	Valencia	2	2	1	1
378	26/05/2024	Las Palmas	Alaves	1	1	1	1
379	26/05/2024	Sevilla	Barcelona	1	2	0	3

380 rows × 7 columns

In [37]:
```
# Creamos un DataFrame con los puntos de los equipos locales
df_laliga_home = df_laliga[['Home Team', 'HT Points']]
```

```
df_laliga_home.head()
```

Out[37]:

	Home Team	HT Points
0	Almeria	0
1	Sevilla	0
2	Real Sociedad	1
3	Las Palmas	1
4	Ath. Bilbao	0

In [38]:
```
# Creamos un DataFrame con los puntos de los equipos visitantes
df_laliga_away = df_laliga[['Away Team', 'AT Points']]
df_laliga_away.head()
```

Out[38]:

	Away Team	AT Points
0	Vallecano	3
1	Valencia	3
2	Girona	1
3	Mallorca	1
4	Real Madrid	3

In [39]:
```
# Renombramos los nombres de columnas de los dos DataFrames para que sean iguales
df_laliga_home = df_laliga_home.rename(columns = {'Home Team':'Club', 'HT Points':'Points'})
df_laliga_away = df_laliga_away.rename(columns = {'Away Team':'Club', 'AT Points':'Points'})

# Unimos los DataFrames por filas con la función 'pd.concat()'
df_laliga_points = pd.concat([df_laliga_home, df_laliga_away])
df_laliga_points
```

Out[39]:

	Club	Points
0	Almeria	0
1	Sevilla	0
2	Real Sociedad	1
3	Las Palmas	1
4	Ath. Bilbao	0
...	...	...
375	Betis	1
376	Mallorca	3
377	Valencia	1
378	Alaves	1
379	Barcelona	3

760 rows × 2 columns

In [40]:
```
# Agrupamos por equipo y sumamos los puntos de cada equipo, ordenando los resultados por la cantidad de puntos
df_laliga_points.groupby('Club').agg({'Points':'sum'}).sort_values(by='Points', ascending=False)
```

Out[40]:

Club	Points
Real Madrid	95
Barcelona	85
Girona	81
Atl. Madrid	76
Ath. Bilbao	68
Real Sociedad	60
Betis	57
Villarreal	53
Valencia	49
Alaves	46
Osasuna	45
Getafe	43
Celta	41
Sevilla	41
Las Palmas	40
Mallorca	40
Vallecano	38
Cadiz	33
Almeria	21
Granada	21

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 [41]:
```
# Importamos el DataFrame con los resultados de LaLiga
df_laliga = pd.read_csv('./data/laliga_results.csv')
df_laliga
```

Out[41]:

	Date	Home Team	Away Team	HT Goals	AT Goals	HT Points	AT Points
0	11/08/2023	Almeria	Vallecano	0	2	0	3
1	11/08/2023	Sevilla	Valencia	1	2	0	3
2	12/08/2023	Real Sociedad	Girona	1	1	1	1
3	12/08/2023	Las Palmas	Mallorca	1	1	1	1
4	12/08/2023	Ath. Bilbao	Real Madrid	0	2	0	3
...	...	...	...	...	...	...	...
375	25/05/2024	Real Madrid	Betis	0	0	1	1
376	26/05/2024	Getafe	Mallorca	1	2	0	3
377	26/05/2024	Celta	Valencia	2	2	1	1
378	26/05/2024	Las Palmas	Alaves	1	1	1	1
379	26/05/2024	Sevilla	Barcelona	1	2	0	3

380 rows × 7 columns

In [42]:
```
# Creamos la columna de resultados concatenando los goles locales y visitantes
df_laliga['Result'] = df_laliga['HT Goals'].astype(str) + ' - ' + df_laliga['AT Goals'].astype(str)
df_laliga.head()
```

Out[42]:

	Date	Home Team	Away Team	HT Goals	AT Goals	HT Points	AT Points	Result
0	11/08/2023	Almeria	Vallecano	0	2	0	3	0 - 2
1	11/08/2023	Sevilla	Valencia	1	2	0	3	1 - 2
2	12/08/2023	Real Sociedad	Girona	1	1	1	1	1 - 1
3	12/08/2023	Las Palmas	Mallorca	1	1	1	1	1 - 1
4	12/08/2023	Ath. Bilbao	Real Madrid	0	2	0	3	0 - 2

In [43]:
```
# Pivotamos el DataFrame, convirtiendo el equipo local en el índice, el equipo visitante en las columnas y los resultados
df_laliga_pivoted = df_laliga.pivot(index='Home Team', columns='Away Team', values='Result')
df_laliga_pivoted.fillna('N/A') # Sustituimos los valores 'NaN' por 'N/A'
```

Out[43]:

Away Team / Home Team	Alaves	Almeria	Ath. Bilbao	Atl. Madrid	Barcelona	Betis	Cadiz	Celta	Getafe	Girona	Granada	Las Palmas	Mallorca	Osasuna	M
Alaves	N/A	1 - 0	0 - 2	2 - 0	1 - 3	1 - 1	1 - 0	3 - 0	1 - 0	2 - 2	3 - 1	0 - 1	1 - 1	0 - 2	
Almeria	0 - 3	N/A	0 - 0	2 - 2	0 - 2	0 - 0	6 - 1	2 - 3	1 - 3	0 - 0	3 - 3	1 - 2	0 - 0	0 - 3	
Ath. Bilbao	2 - 0	3 - 0	N/A	2 - 0	0 - 0	4 - 2	3 - 0	4 - 3	2 - 2	3 - 2	1 - 1	1 - 0	4 - 0	2 - 2	
Atl. Madrid	2 - 1	2 - 1	3 - 1	N/A	0 - 3	2 - 1	3 - 2	1 - 0	3 - 3	3 - 1	3 - 1	5 - 0	1 - 0	1 - 4	
Barcelona	2 - 1	3 - 2	1 - 0	1 - 0	N/A	5 - 0	2 - 0	3 - 2	4 - 0	2 - 4	3 - 3	1 - 0	1 - 0	1 - 0	
Betis	0 - 0	3 - 2	3 - 1	0 - 0	2 - 4	N/A	1 - 1	2 - 1	1 - 1	1 - 1	1 - 0	1 - 0	2 - 0	2 - 1	
Cadiz	1 - 0	1 - 1	0 - 0	2 - 0	0 - 1	0 - 2	N/A	2 - 2	1 - 0	0 - 1	1 - 0	0 - 0	1 - 1	1 - 1	
Celta	1 - 1	1 - 0	2 - 1	0 - 3	1 - 2	2 - 1	1 - 1	N/A	2 - 2	0 - 1	1 - 0	4 - 1	0 - 1	0 - 2	
Getafe	1 - 0	2 - 1	0 - 2	0 - 3	0 - 0	1 - 1	1 - 0	3 - 2	N/A	1 - 0	2 - 0	3 - 3	1 - 2	3 - 2	
Girona	3 - 0	5 - 2	1 - 1	4 - 3	4 - 2	3 - 2	4 - 1	1 - 0	3 - 0	N/A	7 - 0	1 - 0	5 - 3	2 - 0	
Granada	2 - 0	1 - 1	1 - 1	0 - 1	2 - 2	1 - 1	2 - 0	1 - 2	1 - 1	2 - 4	N/A	1 - 1	3 - 2	3 - 0	
Las Palmas	1 - 1	0 - 1	0 - 2	2 - 1	1 - 2	2 - 2	1 - 1	2 - 1	2 - 0	0 - 2	1 - 0	N/A	1 - 1	1 - 1	
Mallorca	0 - 0	2 - 2	0 - 0	0 - 1	2 - 2	0 - 1	1 - 1	1 - 1	0 - 0	1 - 0	1 - 0	1 - 0	N/A	3 - 2	
Osasuna	1 - 0	1 - 0	0 - 2	0 - 2	1 - 2	0 - 2	2 - 0	0 - 3	3 - 2	2 - 4	2 - 0	1 - 1	1 - 1	N/A	
Real Madrid	5 - 0	3 - 2	2 - 0	1 - 1	3 - 2	0 - 0	3 - 0	4 - 0	2 - 1	4 - 0	2 - 0	2 - 0	1 - 0	4 - 0	
Real Sociedad	1 - 1	2 - 2	3 - 0	0 - 2	0 - 1	0 - 0	2 - 0	1 - 1	4 - 3	1 - 1	5 - 3	2 - 0	1 - 0	0 - 1	
Sevilla	2 - 3	5 - 1	0 - 2	1 - 0	1 - 2	1 - 1	0 - 1	1 - 2	0 - 3	1 - 2	3 - 0	1 - 0	2 - 1	1 - 1	
Valencia	0 - 1	2 - 1	1 - 0	3 - 0	1 - 1	1 - 2	2 - 0	0 - 0	1 - 0	1 - 3	1 - 0	1 - 0	0 - 0	1 - 2	
Vallecano	2 - 0	0 - 1	0 - 1	0 - 7	1 - 1	2 - 0	1 - 1	0 - 0	0 - 0	1 - 2	2 - 1	0 - 2	2 - 2	2 - 1	
Villarreal	1 - 1	2 - 1	2 - 3	1 - 2	3 - 4	1 - 2	0 - 0	3 - 2	1 - 1	1 - 2	5 - 1	1 - 2	1 - 1	3 - 1	