

TIENDA - EDA + Ingeniería de Características

ÍNDICE DEL NOTEBOOK

- 1.** Importaciones y configuración inicial.
- 2.** Carga del dataset (Store.csv).
- 3.** Vista rápida del DataFrame.
- 4.** Data Cleaning.
- 5.** Feature Engineering Básico.
- 6.** Preparación y primeros EDA.
- 7.** EDA por ciudad, estado, región y tiempo.
- 8.** EDA por meses, segmentos, shipping, etc.
- 9.** Ship mode, distribuciones y correlación.
- 10.** EDA automático (ydata-profiling).
- 11.** Ingeniería de características y preparación para ML (scikit-learn).
 - 11.1.** Selección de atributos.
 - 11.2.** Transformaciones (log y escalado).
 - 11.3.** Nuevos atributos.
 - 11.4.** Dataset listo para ML con one-hot + train_test_split

1. IMPORTACIONES Y CONFIGURACIÓN INICIAL

```
[1]:  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
sns.set_style('darkgrid')  
  
• import numpy as np --> Importa Numpy con alias np. Usado para operaciones numéricas.  
• import pandas as pd --> Importa Pandas como pd para trabajar con dataframes.  
• import matplotlib.pyplot as plt --> Módulo de gráficos básicos de Matplotlib.  
• %matplotlib inline --> Hace que los gráficos se vean dentro del notebook.  
• import seaborn as sns --> Importa Seaborn para gráficos estadísticos bonitos.  
• sns.set_style('darkgrid') --> Estilo visual de los gráficos: fondo gris con rejilla.
```

2. CARGA DEL DATASET (Store.csv)

```
[2]:  
from pandas.core.reshape import encoding  
df=pd.read_csv ('Sample - Superstore.csv',  
encoding = 'ISO-8859-1',  
parse_dates=['Order Date', 'Ship Date'])  
df.head()  
  
...  


| index | Row ID | Order ID       | Order Date          | Ship Date           | Ship Mode      | Customer ID | Customer Name   | Segment   | Country       | City            | State      | Postal Code | Region | Product ID      | Category        | Sub-Category | Product Name                                                                                                                                      |
|-------|--------|----------------|---------------------|---------------------|----------------|-------------|-----------------|-----------|---------------|-----------------|------------|-------------|--------|-----------------|-----------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| 0     | 1      | CA-2016-152156 | 2016-11-08 00:00:00 | 2016-11-11 00:00:00 | Second Class   | CG-12520    | Claire Gute     | Consumer  | United States | Henderson       | Kentucky   | 42420       | South  | FUR-BO-10001798 | Furniture       | Bookcases    | Bush Bush<br>Collection<br>Bookcase                                                                                                               |
| 1     | 2      | CA-2016-152156 | 2016-11-08 00:00:00 | 2016-11-11 00:00:00 | Second Class   | CG-12520    | Claire Gute     | Consumer  | United States | Henderson       | Kentucky   | 42420       | South  | FUR-CH-10000454 | Furniture       | Chairs       | Hon Deluxe<br>Fabric<br>Upholstered<br>Stacking<br>Chairs,<br>Rounded Bac<br>Self-Adhesive<br>Address Labels<br>for<br>Typewriters b<br>Universal |
| 2     | 3      | CA-2016-138688 | 2016-06-12 00:00:00 | 2016-06-18 00:00:00 | Second Class   | DV-13045    | Darrin Van Huff | Corporate | United States | Los Angeles     | California | 90036       | West   | OFF-LA-10000240 | Office Supplies | Labels       | Bretford<br>CR4500<br>Series Slim<br>Rectangular<br>Table                                                                                         |
| 3     | 4      | US-2015-108966 | 2015-10-11 00:00:00 | 2015-10-18 00:00:00 | Standard Class | SO-20335    | Sean O'Donnell  | Consumer  | United States | Fort Lauderdale | Florida    | 33311       | South  | FUR-TA-10000577 | Furniture       | Tables       | Eldon Fold 'N<br>Roll Cart System                                                                                                                 |
| 4     | 5      | US-2015-108966 | 2015-10-11 00:00:00 | 2015-10-18 00:00:00 | Standard Class | SO-20335    | Sean O'Donnell  | Consumer  | United States | Fort Lauderdale | Florida    | 33311       | South  | OFF-ST-10000760 | Office Supplies | Storage      |                                                                                                                                                   |



Show 25 over page



EXPLICACIÓN CELDA:



- pd.read_csv(...)
  - Lee el CSV desde esa ruta.
  - encoding='ISO-8859-1' -> para soportar caracteres especiales.
  - parse_dates=['Order Date','Ship Date'] -> convierte esas columnas a tipo fecha.
- df -> DataFrame principal con todos los datos.
- df.head() -> Muestra las primeras filas para ver como es la tabla.

```

3. VISTA RÁPIDA DEL DATAFRAME

The screenshot shows three code cells in a Jupyter Notebook interface:

- Cell 4:** df.info()
Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   Row ID           9994 non-null   int64  
 1   Order ID         9994 non-null   object  
 2   Order Date       9994 non-null   datetime64[ns]
 3   Ship Date        9994 non-null   datetime64[ns]
 4   Ship Mode        9994 non-null   object  
 5   Customer ID     9994 non-null   object  
 6   Customer Name    9994 non-null   object  
 7   Segment          9994 non-null   object  
 8   Country          9994 non-null   object  
 9   City              9994 non-null   object  
 10  State             9994 non-null   object  
 11  Postal Code      9994 non-null   int64  
 12  Region            9994 non-null   object  
 13  Product ID       9994 non-null   object  
 14  Category          9994 non-null   object  
 15  Sub-Category      9994 non-null   object  
 16  Product Name      9994 non-null   object  
 17  Sales              9994 non-null   float64 
 18  Quantity           9994 non-null   int64  
 19  Discount            9994 non-null   float64 
 20  Profit              9994 non-null   float64 
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
```
- Cell 5:** Muestra:
 - nº filas y columnas,
 - tipo de cada columna,
 - si hay nulos.
- Cell 6:** df.describe()
Output:

	Row ID	Order Date	Ship Date	Postal Code	Sales	Quantity	Discount	Profit
count	9994.000000	9994	9994	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	4997.500000	2016-04-30 00:07:12.259355648	2016-05-03 23:06:58.571142912	55190.379428	229.858001	3.789574	0.156203	28.656896
min	1.000000	2014-01-03 00:00:00	2014-01-07 00:00:00	1040.000000	0.444000	1.000000	0.000000	-6599.978000
25%	2499.250000	2015-05-23 00:00:00	2015-05-27 00:00:00	23223.000000	17.280000	2.000000	0.000000	1.728750
50%	4997.500000	2016-06-26 00:00:00	2016-06-29 00:00:00	56430.500000	54.490000	3.000000	0.200000	8.666500
75%	7495.750000	2017-05-14 00:00:00	2017-05-18 00:00:00	90008.000000	209.940000	5.000000	0.200000	29.364000
max	9994.000000	2017-12-30 00:00:00	2018-01-05 00:00:00	99301.000000	22638.480000	14.000000	0.800000	8399.976000
std	2885.163629	NaN	NaN	32063.693350	623.245101	2.225110	0.206452	234.260108
- Cell 7:** Estadísticas de columnas numéricas: media, desviación estándar, min, max, porcentajes, etc.
- Cell 8:** df.describe(include='object')
Output:

	order ID	Ship Mode	Customer ID	Customer Name	Segment	country	city	State	Region	Product ID	Category	Sub-Category	Product Name
count	9994	9994	9994	9994	9994	9994	9994	9994	9994	9994	9994	9994	9994
unique	5009	4	793	793	3	1	531	49	4	1862	3	17	1850
top	CA-2017-100111	Standard Class	WB-21850	William Brown	Consumer	United States	New York City	California	West	OFF-PA-10001970	Office Supplies	Binders	Staple envelope
freq	14	5968	37	37	5191	9994	915	2001	3203	19	6026	1523	48
- Cell 9:** Estadísticas de columnas categóricas (tipo object): count, nº de valores únicos, valor más frecuente, frecuencia.

4. LIMPIEZA BÁSICA (DATA CLEANING)

The screenshot shows three code cells in a Jupyter Notebook interface:

- Cell 1:** DATA CLEANING
df.duplicated().sum()
np.int64(0)
df.duplicated().sum() -> Cuenta cuántas filas duplicadas hay en el DataFrame.
- Cell 2:** df.drop_duplicates(inplace=True)
Elimina filas duplicadas.
inplace=True -> modifica df directamente.
- Cell 3:** df.isna().sum()
No hay valores nulos
Row ID 0
Order ID 0
Order Date 0
Ship Date 0
Ship Mode 0
Customer ID 0
Customer Name 0
Segment 0
Country 0

```

City      0
State     0
Postal Code 0
Region    0
Product ID 0
Category   0
Sub-Category 0
Product Name 0
Sales      0
Quantity   0
Discount   0
Profit     0
dtype: int64

• df.isna().sum() -> Cuenta nulos por columna.
• El comentario nos indica que no hay valores nulos.

```

5. FEATURE ENGINEERING BÁSICO

```

[10]  os  df['Cost']=df['Sales']-df['Profit']
      df['profit_margin']=(df['Profit']/df['Sales'])*100

• df['Cost'] -> Crea una columna de coste:
  • Coste=Ventas - Beneficio.

• df['profit_margin'] -> Margen de beneficio en %:
  • (Profit/Sales)*100.

[11]  os  df['is_gain']=np.where(df['Profit']>0,1,0)

• np.where(condición, valor_si_True, valor_si_False)
• Crea columna binaria:
  • 1 si Profit > 0 (ganancia),
  • 0 si Profit ≤ 0 (pérdida).

[12]  os  df['days_to_ship']=(df['Ship Date']-df['Order Date']).dt.days.astype(int)

• Resta fechas (Fecha pedido - Fecha envío) -> Timedelta (es un objeto de un intervalo de tiempo).
• .dt.days -> Convierte el Timedelta a nº de días.
• .astype(int) -> asegura tipo entero.
• df['Days_to_ship'] -> Crea días que tarda en enviarse.

```



```

[13]  os  df['sales_per_quantity']=df['Sales']/df['Quantity']

• df['sales_per_quantity'] -> Crea "precio medio por unidad": ventas totales/cantidad.

[14]  os  df['month_order']=df['Order Date'].dt.month
      df['year_order']=df['Order Date'].dt.year

• Extrae mes y año de la fecha de pedido.

[15]  os  df.columns

• Index(['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode', 'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State', 'Postal Code', 'Region', 'Product ID', 'Category', 'Sub-Category', 'Product Name', 'Sales', 'Quantity', 'Discount', 'Profit', 'Cost', 'Profit Margin', 'Is Gain', 'Days To Ship', 'Sales Per Quantity', 'Month Order', 'Year Order'], dtype='object')

• Muestra todas las columnas del DataFrame.

[16]  os  len(df.columns)

• Cuenta cuántas columnas hay.

```

6. PREPARACIÓN Y PRIMEROS EDA

The screenshot shows a Jupyter Notebook interface with three code cells. The first cell contains code to filter and set the index of a DataFrame, followed by a call to .head() to show the first few rows. The second cell shows the resulting DataFrame with columns: Row ID, Ship Mode, Customer Name, Segment, City, State, Country, Region, Category, Sub-Category, Product Name, ..., Quantity, Discount, Cost, Profit, profit_margin, is_gain, and Dis. The third cell contains a list of notes and a block of code for calculating totals.

```
[17]: df_used=df[['Row ID','Ship Mode','Customer Name','Segment','City','State','Country','Region','Category','Sub-Category','Product Name','...','Quantity','Discount','Cost','Profit','profit margin','is_gain','Days to ship','sales_per_quantity','month_order','year_order']] df_used.set_index('Row ID',inplace=True) df_used.head()
```

Row ID	Ship Mode	Customer Name	Segment	City	State	Country	Region	Category	Sub-Category	Product Name	...	Quantity	Discount	Cost	Profit	profit margin	is_gain	Dis
1	Second Class	Claire Gute	Consumer	Henderson	Kentucky	United States	South	Furniture	Bookcases	Bush Somerset Collection Bookcase	...	2	0.00	220.0464	41.9136	16.00	1	
2	Second Class	Claire Gute	Consumer	Henderson	Kentucky	United States	South	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	...	3	0.00	512.3580	219.5820	30.00	1	
3	Second Class	Darrin Van Huff	Corporate	Los Angeles	California	United States	West	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	...	2	0.00	7.7486	6.8714	47.00	1	
4	Standard Class	Sean O'Donnell	Consumer	Fort Lauderdale	Florida	United States	South	Furniture	Tables	Bretford CR4500 Series Slim Rectangular	...	5	0.45	1340.6085	-383.0310	-40.00	0	

Variables Terminal ✓ 4:40 Python 3

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell displays a table with 5 rows and 21 columns. The second cell contains a list of notes and a block of code for calculating totals.

Table
5 Standard Class Sean O'Donnell Consumer Fort Lauderdale Florida United States South Office Supplies Storage Eldon Fold 'N Roll Cart System ... 2 0.20 19.8516 2.5164 11.25 1
5 rows × 21 columns

• df_used=df[[...]] -> Crea un subconjunto con las columnas que vas a usar.
• set_index('Row ID') -> Pone Row ID como índice (identificador único).
• df_used.head() -> vista rápida.

```
[18]: # Suma de Ventas sum_sales=df_used['Sales'].sum()

# Suma de Ganancias sum_profit=df_used['Profit'].sum()

# Suma de Costes sum_cost=df_used['Cost'].sum()

# Suma de Cantidadessum_quantity=df_used['Quantity'].sum()

# Suma de Descuentos sum_discount=df_used['Discount'].sum()

print('sum_sales:',sum_sales)
print('sum_profit:',sum_profit)
print('sum_cost:',sum_cost)
print('sum_quantity:',sum_quantity)
print('sum_discount:',sum_discount)
```

Variables Terminal ✓ 4:40 Python 3

The screenshot shows a Jupyter Notebook interface with one code cell. The cell displays the results of the calculations from the previous cell, including the total sales, profit, cost, quantity, and discount.

```
sum_sales: 2297200.8603000003
sum_profit: 286397.0217
sum_cost: 2010801.8386
sum_quantity: 37873
sum_discount: 1561.09
```

• Calcula la suma total de:
◦ ventas, beneficios, costes, cantidades y descuentos.
• Guarda cada suma en una variable.
• Imprime todas las sumas totales.

Reto2-Tienda.ipynb

```
[20] 0s
bars=plt.bar(['sum_sales','sum_profit','sum_cost'],
             [sum_sales,sum_profit,sum_cost],
             color=[ "#7777C0", "#B2B89F", "#CBCBCB"])
plt.title('sum of sales, profit and cost')
ax=plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.bar_label(bars)
plt.show()
```

...
Reto2-Tienda.ipynb

```
[21] 0s
total_profit_margin=(sum_profit/sum_sales)*100
total_profit_margin
... np.float64(12.467217240315604)
```

- plt.bar(...) -> Gráfico de barras con las tres sumas totales.
- plt.title(...) -> Título del gráfico.
- plt.gca() -> Obtiene los ejes actuales.
- ax.spines[...] -> Oculta bordes superior y derecho para estética.
- plt.bar_label(bars) -> Pone etiquetas con los valores encima de cada barra.
- plt.show() -> Muestra el gráfico.

...
Reto2-Tienda.ipynb

```
[38] 0s
sns.countplot(data=dr_used,x= category ,hue= category , palette='Blues')
plt.title('Category')
plt.show()
```

- sns.countplot -> Cuenta cuántos registros hay por categoría.
- Eje X: Category.
- palette='Blues' -> Colores azules.
- Titulo del gráfico.
- show -> Muestra que **Office Supplies** es la más frecuente.

Retorno de ejecución

```
[23] 0 s
sales_furniture=df.loc[df['Category']=='Furniture','Sales'].sum()
sales_office=df.loc[df['Category']=='Office Supplies','Sales'].sum()
sales_technology=df.loc[df['Category']=='Technology','Sales'].sum()

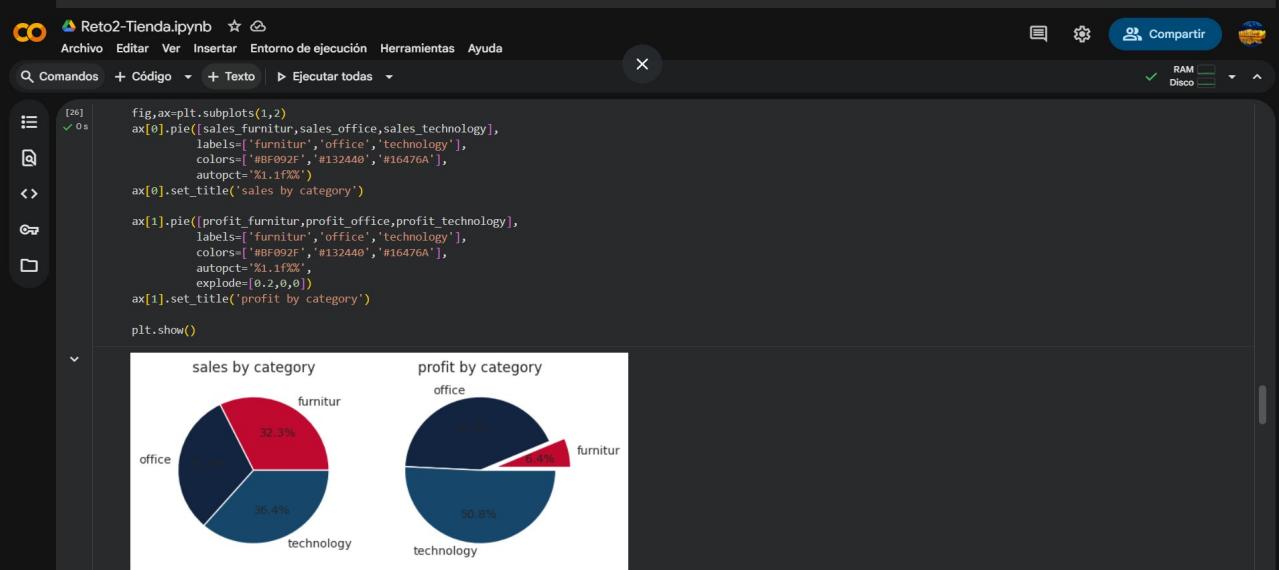
print('sales furniture:',sales_furniture)
print('sales office:',sales_office)
print('sales technology:',sales_technology)

profit_furniture=df.loc[df['Category']=='Furniture','Profit'].sum()
profit_office=df.loc[df['Category']=='Office Supplies','Profit'].sum()
profit_technology=df.loc[df['Category']=='Technology','Profit'].sum()

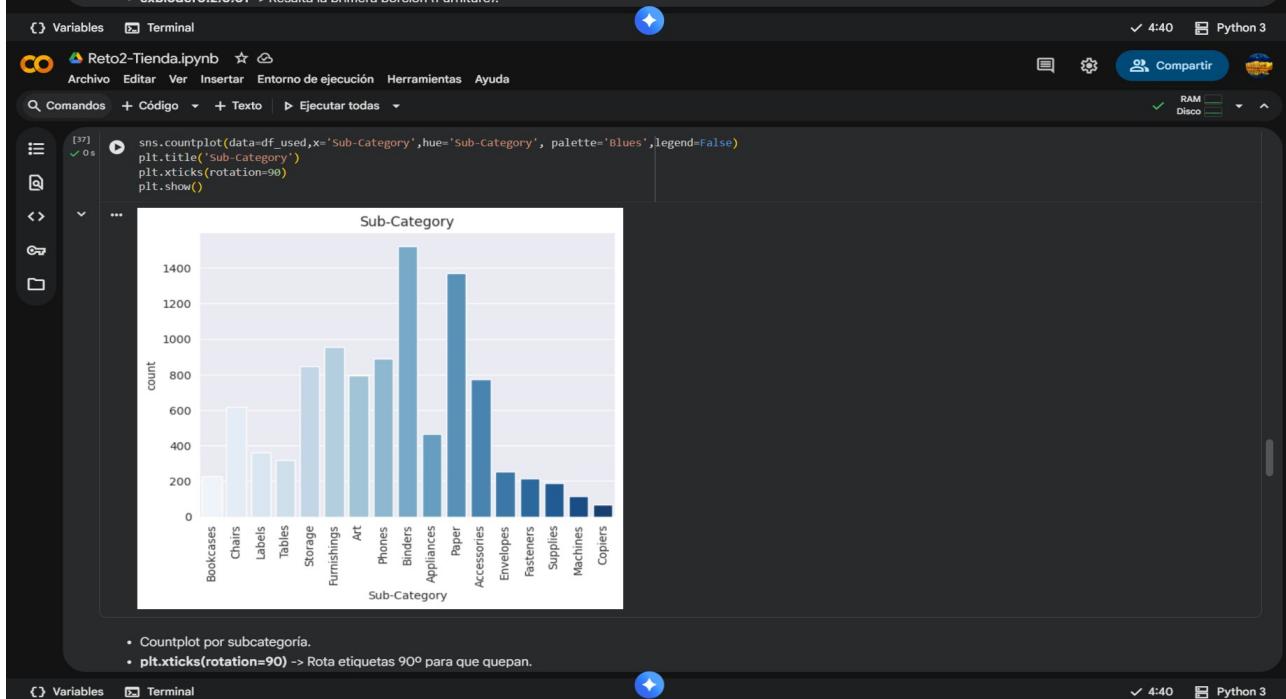
print('profit furniture:',profit_furniture)
print('profit office:',profit_office)
print('profit technology:',profit_technology)

...
sales_furniture: 741999.7953
sales_office: 719047.032
sales_technology: 836154.033
-----
profit_furniture: 18451.272800000006
profit_office: 122498.8008
profit_technology: 145454.9481

• df['Category']=='...' -> Crea un filtro booleano por categoría.
• df.loc[filtro, 'Sales'] -> Selecciona las filas donde Category = ...
• .sum() -> Suma esas 'Sales' de la selección ...
• Imprime las ventas totales de los tres resultados.
• Mismo estructura para 'Profit'.
```



- fig,ax=plt.subplots(1,2) -> Crea una figura con dos subgráficos en una fila.
- Primer gráfico (ax[0]): pastel de ventas por categoría.
- Segundo (ax[1]): pastel de beneficios por categoría.
- explode=0.2,0,0 -> Resalta la primera porción (Furniture).



Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comandos Código Texto Ejecutar todas

RAM Disco

order1=df_used.groupby('Sub-Category')['Sales'].sum().sort_values(ascending=False)
order2=df_used.groupby('Sub-Category')['Profit'].sum().sort_values(ascending=False)

- df_used.groupby('Sub-Category') -> Agrupar las filas por subcategoría:
 - order1='Sales'.sum() -> Por cada grupo de subcategoría suma ventas.
 - order2='Profit.sum()' -> Por cada grupo de subcategoría suma ganancias.
 - .sort_values(ascending=False) -> Ordena de mayor a menor.
- se guardan en las variables para usarlo en gráficos posteriores.

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comandos Código Texto Ejecutar todas

RAM Disco

```
plt.figure(1)
sns.barplot(data=df_used,x='Sub-Category',y='Sales',hue='Sub-Category',
            palette='YlGnBu',legend=False,order=order1.index,estimator='sum')
plt.title('Sales by Sub-Category')
plt.xticks(rotation=90)
plt.show()

plt.figure(2)
sns.barplot(data=df_used,x='Sub-Category',y='Profit',hue='Sub-Category',
            palette='YlGnBu',legend=False,order=order2.index,estimator='sum')
plt.title('Profit by Sub-Category')
plt.xticks(rotation=90)
plt.show()
```

Sales by Sub-Category

Sub-Category	Sales
Phones	~320,000
Chairs	~300,000
Storage	~220,000
Tables	~210,000
Binders	~200,000
Linens	~190,000
Copiers	~170,000
Cases	~140,000
Furnishings	~110,000
Art	~90,000
Supplies	~80,000
Labels	~50,000
Envelopes	~30,000

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

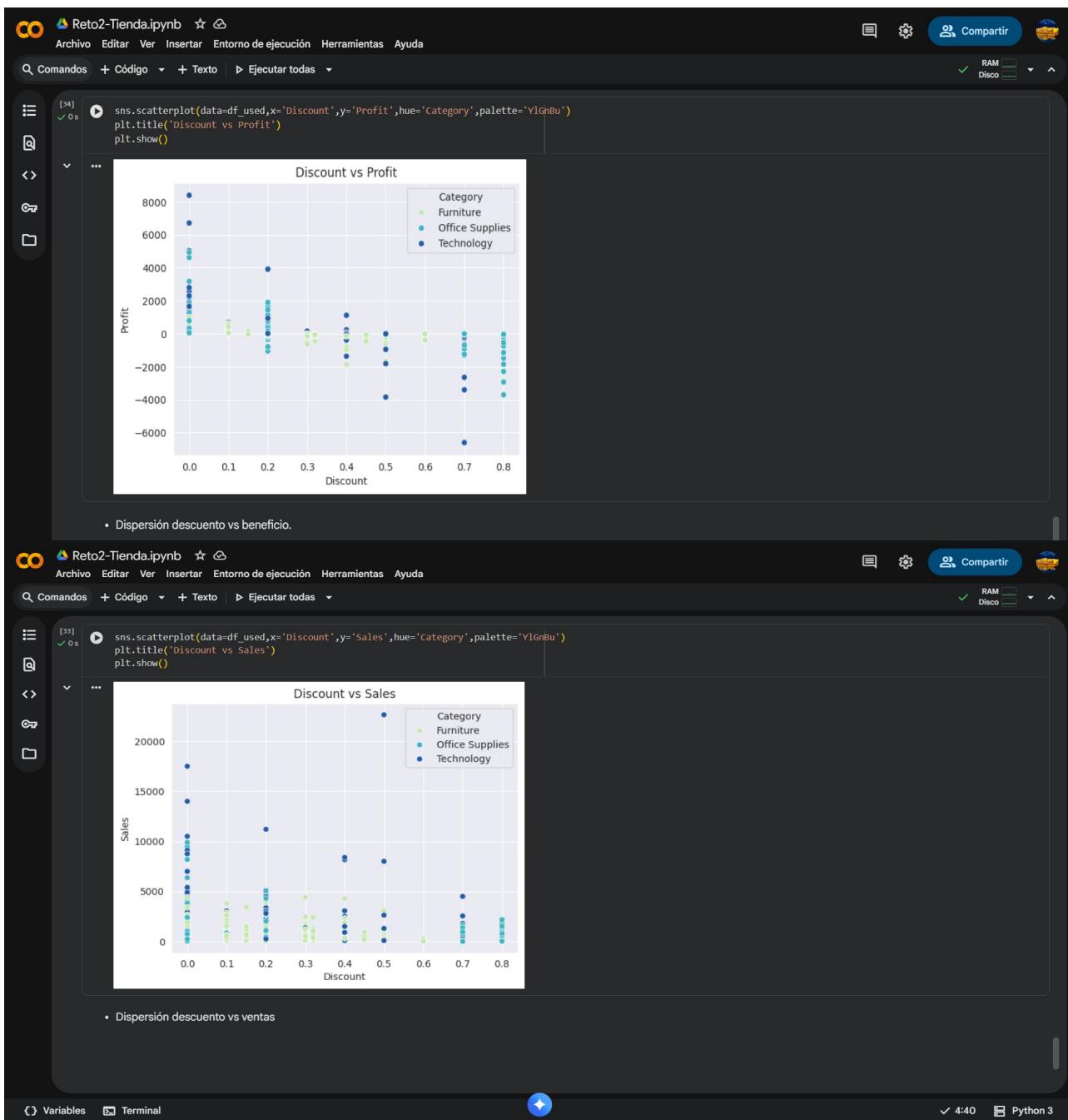
Comandos Código Texto Ejecutar todas

RAM Disco

Profit by Sub-Category

Sub-Category	Profit
Copiers	~55,000
Phones	~45,000
Accessories	~40,000
Paper	~35,000
Binders	~30,000
Chairs	~25,000
Storage	~20,000
Appliances	~18,000
Furnishings	~15,000
Envelopes	~10,000
Art	~8,000
Labels	~7,000
Machines	~5,000
Fasteners	~3,000
Supplies	~2,000
Bookcases	~1,000
Tables	~-10,000

- Creamos dos gráficos:
 - Primer: barra de ventas por subcategoría.
 - Segundo: barra de ganancias por subcategoría.
 - order=order1.index/order2.index -> Ordena de mayor a menor.
 - estimator='sum' -> Usa la suma como medida.



7. EDA POR CIUDAD, ESTADO, REGIÓN Y TIEMPO

Reto2-Tienda.ipynb

```
[66] 0 s sales_for_cities=df_used.pivot_table(index='City', values='Sales',aggfunc='sum').sort_values(by='Sales',ascending=False)
sales_for_cities=sales_for_cities.head(15)
sales_for_cities
```

city	Sales
New York City	256368.1610
Los Angeles	175851.3410
Seattle	119540.7420
San Francisco	112669.0920
Philadelphia	109077.0130
Houston	64504.7604
Chicago	48539.5410
San Diego	47521.0290
Jacksonville	44713.1830
Springfield	43054.3420
Detroit	42446.9440
Columbus	38706.2430
Newark	28576.1190
Columbia	25283.3240
Lafayette	25036.2000

Pasos siguientes: Generar código con sales_for_cities | New interactive sheet

Variables Terminal

Reto2-Tienda.ipynb

```
Ejecutar todas las celdas del cuaderno.
```

- df_used.pivot_table -> Crea una tabla dinámica.
 - Filas: index='City' (ciudades).
 - Valor: values='Sales',aggfunc='sum' (suma de ventas).
- Ordena de mayor a menor por 'Sales'.
- Muestra el resultado.

Reto2-Tienda.ipynb

```
bars=sns.barplot(data=sales_for_cities,x=sales_for_cities.index,y='Sales',color='red')
bars.patches[0].set_hatch('/')
plt.title('top 15 Sales for city')
plt.xticks(rotation=90)
plt.show()
```

Variables Terminal

• Gráfico de barras con las 15 ciudades con más ventas.
 • `bars.patches[0].set_hatch('/')` -> Pone la primera barra "rayada" para destacarla.

```
[68] 0 s
sales_for_state=df_usa.pivot_table(index='State',values='Sales',aggfunc='sum').sort_values(by='Sales',ascending=False)
sales_for_state=sales_for_state.head(15)
sales_for_state
```

State	Sales
California	457687.6315
New York	310876.2710
Texas	170188.0458
Washington	138641.2700
Pennsylvania	116511.9140
Florida	89473.7080
Illinois	80166.1010
Ohio	78258.1360
Michigan	76269.6140
Virginia	70636.7200
North Carolina	55603.1640
Indiana	53555.3600
Georgia	49095.8400
Kentucky	36591.7500
New Jersey	35764.3120

Variables Terminal ✓ 21:24 Python 3

• Igual pero con estado (state).

```
[69] 0 s
bars=sns.barplot(data=sales_for_state,x=sales_for_state.index,y='Sales',color='blue')
bars.patches[0].set_hatch('/')
plt.title('top 15 Sales for State')
plt.xticks(rotation=90)
plt.show()
```

Variables Terminal ✓ 21:24 Python 3

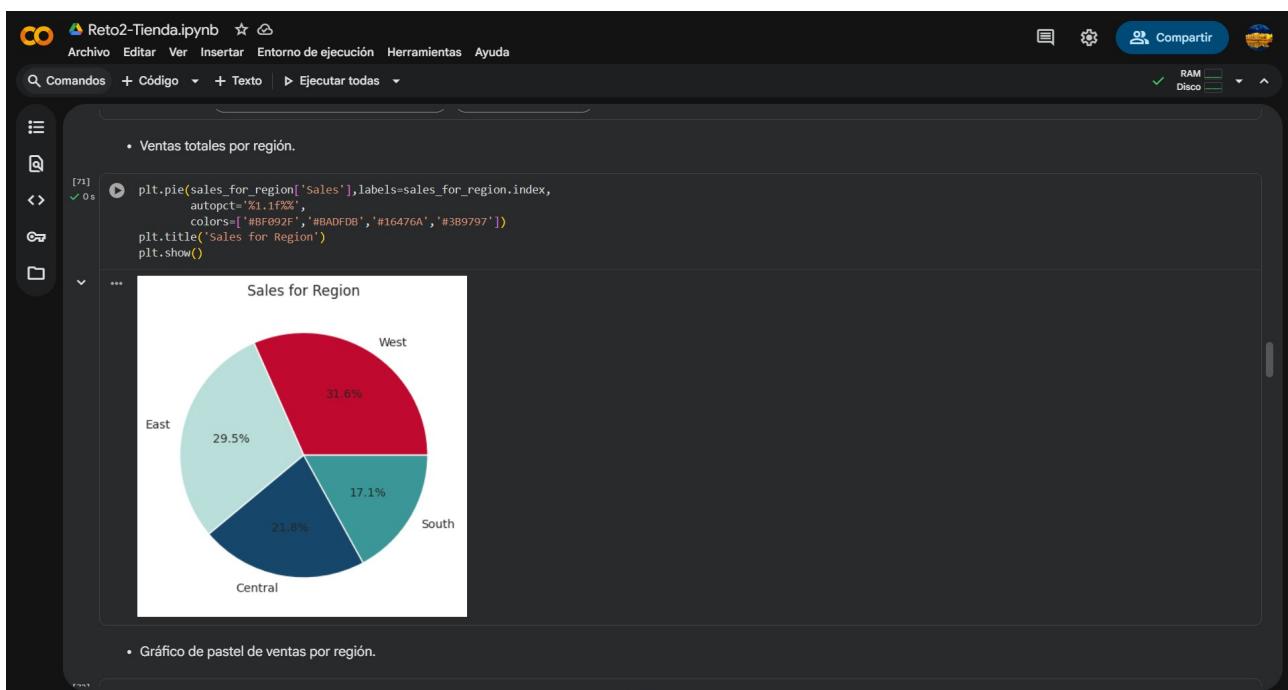
• Gráfico de barras de venta por estado(top 15).

```
[70] 0 s
sales_for_region=df_usa.pivot_table(index='Region',values='Sales',aggfunc='sum').sort_values(by='Sales',ascending=False)
sales_for_region
```

Region	Sales
West	72545.8245
East	678781.2400
Central	501239.8908
South	391721.9050

Pasos siguientes: Generar código con `sales_for_region` New interactive sheet

• Ventas totales por región.



Customer Name	Sales
Sean Miller	25043.050
Tamara Chand	19052.218
Raymond Buch	15117.339
Tom Ashbrook	14595.620
Adrian Barton	14473.571
Ken Lonsdale	14175.229
Sanjit Chand	14142.334
Hunter Lopez	12873.298
Sanjit Engle	12209.438
Christopher Conant	12129.072
Todd Sumrall	11891.751
Greg Tran	11820.120
Becky Martin	11789.630
Seth Vernon	11479.950
Caroline Jumper	11164.974
Clay Ludtke	10880.546
Maria Etezadi	10663.728

Pasos siguientes: Generar código con top_20_customer | New interactive sheet

• Clientes ordenados de mayor a menor por ventas, top 20.

Retorno de ejecución

```
[73] 0 s
sales_for_product=df_used.pivot_table(index='Product Name',values='Sales',aggfunc=['sum','count'])
sales_for_product=sales_for_product.sort_values(by=['sum','Sales'],ascending=False)
sales_for_product.head(20)
```

Product Name	sum	count
	Sales	Sales
Canon ImageCLASS 2200 Advanced Copier	61599.8240	5
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind	27453.3840	10
Cisco TelePresence System EX90 Videoconferencing Unit	22638.4800	1
HON 5400 Series Task Chairs for Big and Tall	21870.5760	8
GBC DocuBind TL300 Electric Binding System	19823.4790	11
GBC Ibimaster 500 Manual ProClick Binding System	19024.5000	9
Hewlett Packard LaserJet 3310 Copier	18839.6860	8
HP Designjet T520 Inkjet Large Format Printer - 24" Color	18374.8950	3
GBC DocuBind P400 Electric Binding System	17965.0680	6
High Speed Automatic Electric Letter Opener	17030.3120	3
Lexmark MX611dhe Monochrome Laser Printer	16829.9010	4
Martin Yale Chadless Opener Electric Letter Opener	16656.2000	6
Ibico EPK-21 Electric Binding System	15875.9160	3
Riverside Palais Royal Lawyers Bookcase, Royale Cherry Finish	15610.9656	5
3D Systems Cube Printer, 2nd Generation, Magenta	14299.8900	2
Samsung Galaxy Mega 6.3	13943.6680	6

Retorno de ejecución

```
[74] 0 s
Apple iPhone 5 12996.6000 6
Bretford Rectangular Conference Table Tops 12995.2915 12
Global Troy Executive Leather Low-Back Tilter 12975.3820 9
Canon PC1060 Personal Laser Copier 11619.8340 4
```

Pasos siguientes: Generar código con sales_for_product | New interactive sheet

- Tabla dinámica por producto:
 - suma de ventas(sum) y nº de apariciones(count). Ordena por suma de ventas descendente y muestra los 20 primeros.

Retorno de ejecución

```
[75] 0 s
profit_for_product=df_used.pivot_table(index='Product Name',values='Profit',aggfunc=['sum','count'])
profit_for_product=profit_for_product.sort_values(by=['sum','Profit'],ascending=False)
profit_for_product.head(20)
```

Product Name	sum	count
	Profit	Profit
Canon ImageCLASS 2200 Advanced Copier	25199.9280	5
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind	7753.0390	10
Hewlett Packard LaserJet 3310 Copier	6983.8836	8
Canon PC1060 Personal Laser Copier	4570.9347	4
HP Designjet T520 Inkjet Large Format Printer - 24" Color	4094.9766	3
Ativa V4110MDD Micro-Cut Shredder	3772.9461	2
3D Systems Cube Printer, 2nd Generation, Magenta	3717.9714	2
Plantronics Savi W720 Multi-Device Wireless Headset System	3696.2820	7
Ibico EPK-21 Electric Binding System	3345.2823	3
Zebra ZM400 Thermal Label Printer	3343.5360	2
Honeywell Enviracaire Portable HEPA Air Cleaner for 17' x 22' Room	3247.0200	8
Hewlett Packard 610 Color Digital Copier / Printer	3124.9375	7
Plantronics CS510 - Over-the-Head monaural Wireless Headset System	3085.0325	10
Canon Imageclass D680 Copier / Fax	2799.9600	5
Fellowes PB300 Plastic Comb Binding Machine	2518.0551	6
Ibico Ibimaster 300 Manual Binding System	2318.3370	8
Logitech Z-906 Speaker sys - home theater - 5.1-CH	2243.9320	4

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comandos Código Texto Ejecutar todas

RAM Disco

GBC DocuBind TL300 Electric Binding System 2233.5051 11
Razer Tiamat Over Ear 7.1 Surround Sound PC Gaming Headset 2155.8922 7
Canon PC940 Copier 2092.4535 3

Pasos siguientes: Generar código con profit_for_product New interactive sheet

- Igual que el anterior pero con beneficios.

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comandos Código Texto Ejecutar todas

RAM Disco

```
[75] 0s
profit_for_product=df_usd.pivot_table(index='Product Name',values='Profit',aggfunc=['sum','count'])
profit_for_product=profit_for_product.sort_values(by='sum','Profit',ascending=True)
profit_for_product.head(20)
```

Product Name	sum	count
Cubify CubeX 3D Printer Double Head Print	-8879.9704	3
Lexmark MX611dhe Monochrome Laser Printer	-4589.9730	4
Cubify CubeX 3D Printer Triple Head Print	-3839.9904	1
Chromcraft Bull-Nose Wood Oval Conference Tables & Bases	-2876.1156	5
Bush Advantage Collection Racetrack Conference Table	-1934.3976	7
GBC DocuBind P400 Electric Binding System	-1878.1662	6
Cisco TelePresence System EX90 Videoconferencing Unit	-1811.0784	1
Martin Yale Chainless Opener Electric Letter Opener	-1299.1836	6
Balt Solid Wood Round Tables	-1201.0581	4
BoxOffice By Design Rectangular and Half-Moon Meeting Room Tables	-1148.4375	3
Riverside Furniture Oval Coffee Table, Oval End Table, End Table with Drawer	-1147.4000	5
Epson TM-T88V Direct Thermal Printer - Monochrome - Desktop	-1057.2300	2
Hon 2090 Pillow Soft Series Mid Back Swivel/Tilt Chairs	-989.0496	6
O'Sullivan 4-Shelf Bookcase in Odessa Pine	-975.0988	7
Bretford Just In Time Height-Adjustable Multi-Task Work Tables	-964.1940	4
Zebra GK420t Direct Thermal/Thermal Transfer Printer	-938.2800	1

Variables Terminal 21:24 Python 3

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comandos Código Texto Ejecutar todas

RAM Disco

3.6 Cubic Foot Counter Height Office Refrigerator -872.0752 5
Bevis Oval Conference Table, Walnut -856.1144 8
Tenneco Single-Tier Lockers -825.7480 8
BPI Conference Tables -795.9725 5

Pasos siguientes: Generar código con profit_for_product New interactive sheet

- ascending=True -> Ordena de menor a mayor beneficio(productos que más pierden)

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Comandos Código Texto Ejecutar todas

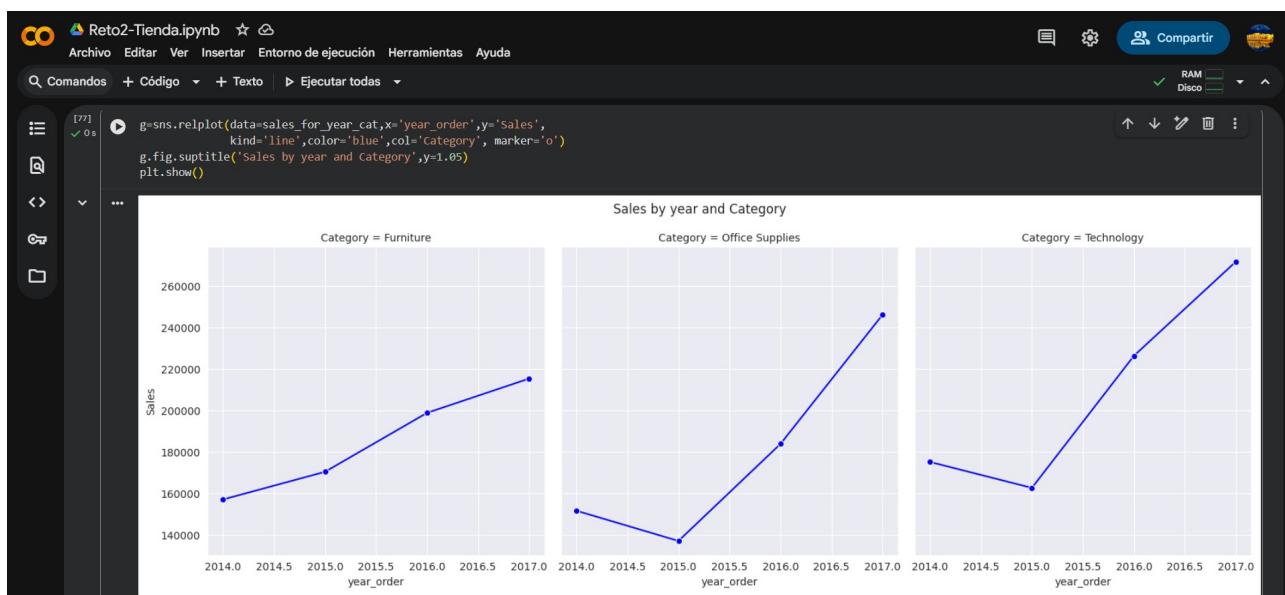
RAM Disco

```
[76] 0s
sales_for_year_cat=df_usd.groupby(['year_order','Category'],as_index=False)['Sales'].sum()
```

year_order	Category	Sales
0	2014	Furniture 157192.8531
1	2014	Office Supplies 151776.4120
2	2014	Technology 175278.2330
3	2015	Furniture 170518.2370
4	2015	Office Supplies 137233.4630
5	2015	Technology 162780.8090
6	2016	Furniture 188901.4360
7	2016	Office Supplies 183939.9820
8	2016	Technology 226364.1800
9	2017	Furniture 215387.2692
10	2017	Office Supplies 246097.1750
11	2017	Technology 271730.8110

Pasos siguientes: Generar código con sales_for_year_cat New interactive sheet

- Agrupa por año y categoría la suma de ventas.



Retorno de ejecución

- sns.relplot(...) -> Gráfico de relación de Seaborn, da una figura con uno o varios subgráficos.
- kind='line' -> Le dice a Seaborn que quiere una gráfica de líneas.
- col='Category' -> Divide el gráfico en columnas, una por categoría(facet grid), un gráfico por cada columna.
- marker='o' -> Añade puntos en cada valor de la línea por estética.
- g.fig.suptitle('Sales by year and Category',y=1.05):
 - sns.relplot devuelve un objeto g.
 - g.fig es la figura completa.
 - suptitle="super title"=título general para toda la figura.
 - y=1.05 ajusta la posición vertical para que el título no quede encima de los gráficos.

8. EDA POR MESES, SEGMENTOS, SHIPPING, ETC

Retorno de ejecución

```
[215] 0 s
sales_by_month=df_used.groupby(['month_order','year_order'], as_index=False)[['Sales']].sum()
```

	month_order	year_order	Sales
0	1	2014	14236.8950
1	1	2015	18174.0756
2	1	2016	18542.4910
3	1	2017	43971.3740
4	2	2014	4519.8920
5	2	2015	11951.4110
6	2	2016	22978.8150
7	2	2017	20301.1334
8	3	2014	55691.0090
9	3	2015	38726.2520
10	3	2016	51715.8750
11	3	2017	58872.3528
12	4	2014	28285.3450
13	4	2015	34195.2085
14	4	2016	38750.0390
15	4	2017	36521.5361
16	5	2014	23648.2870
17	5	2015	30131.6865
18	5	2016	56987.7280

Retorno de ejecución

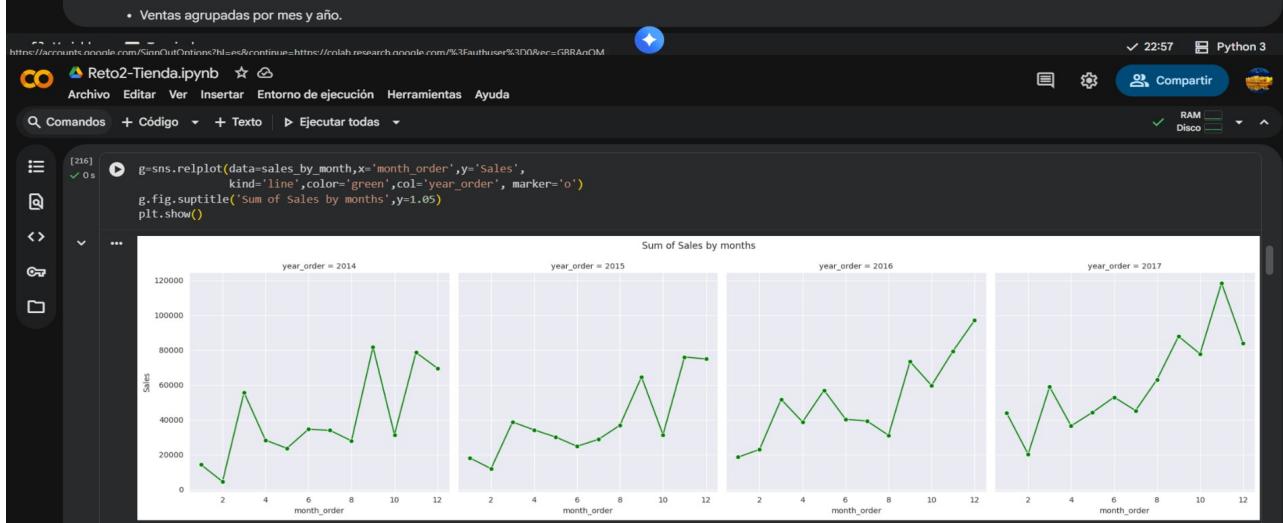
```

29      8    2015  36898.3322
30      8    2016  31115.3743
31      8    2017  63120.8880
32      9    2014  81777.3508
33      9    2015  64595.9180
34      9    2016  73410.0249
35      9    2017  87866.9520
36     10    2014  31453.3930
37     10    2015  31404.9235
38     10    2016  59687.7450
39     10    2017  77776.9232
40     11    2014  78628.7167
41     11    2015  75972.5635
42     11    2016  79411.9688
43     11    2017  118447.8250
44     12    2014  69545.6205
45     12    2015  74919.5212
46     12    2016  96999.0430
47     12    2017  83829.3188

```

Pasos siguientes: Generar código con sales_by_month | New interactive sheet

- Ventas agrupadas por mes y año.



Retorno de ejecución

```
[218] 0 s
sales_year=df_used.pivot_table(index='year_order',values=['Sales','Profit'],aggfunc='sum')
sales_year.sort_values(by='Sales',ascending=False)
```

year_order	Profit	Sales
2017	93439.2696	733215.2552
2016	81795.1743	609205.5980
2014	49543.9741	484247.4981
2015	61618.6037	470532.5090

• Ventas y beneficios totales por año.

• Ordena de mayor a menor venta.

Retorno de ejecución

```
[218] 0 s
sales_mon=df_used.pivot_table(index='month_order',values='Sales',columns='year_order',aggfunc='sum',margins=True)
sales_mon
```

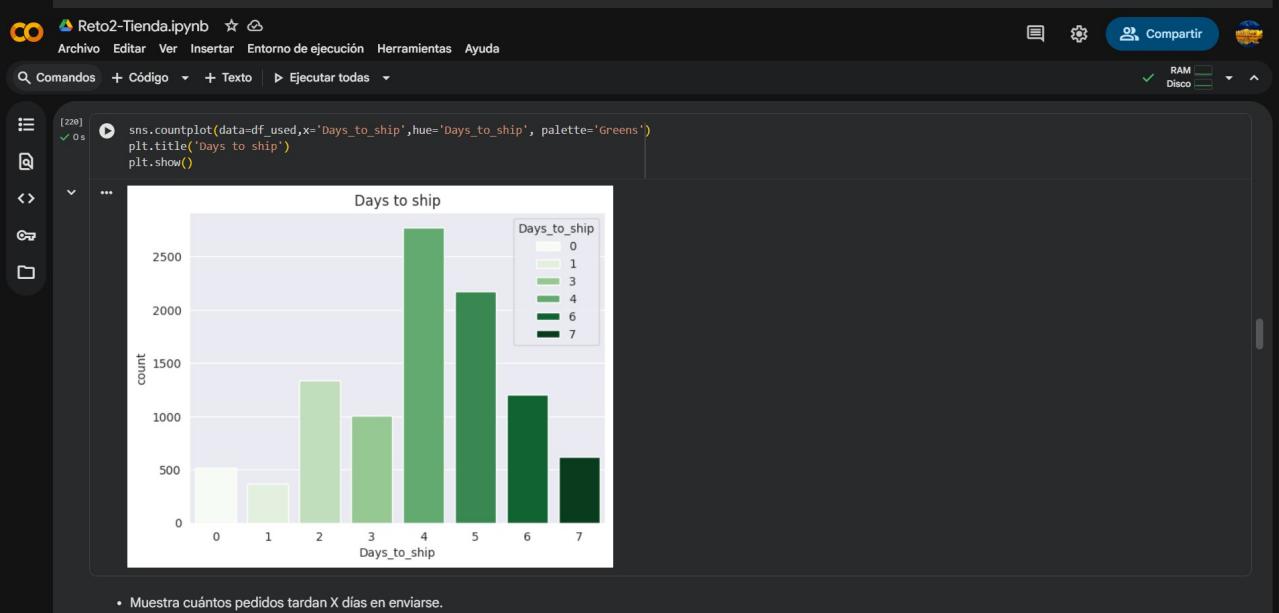
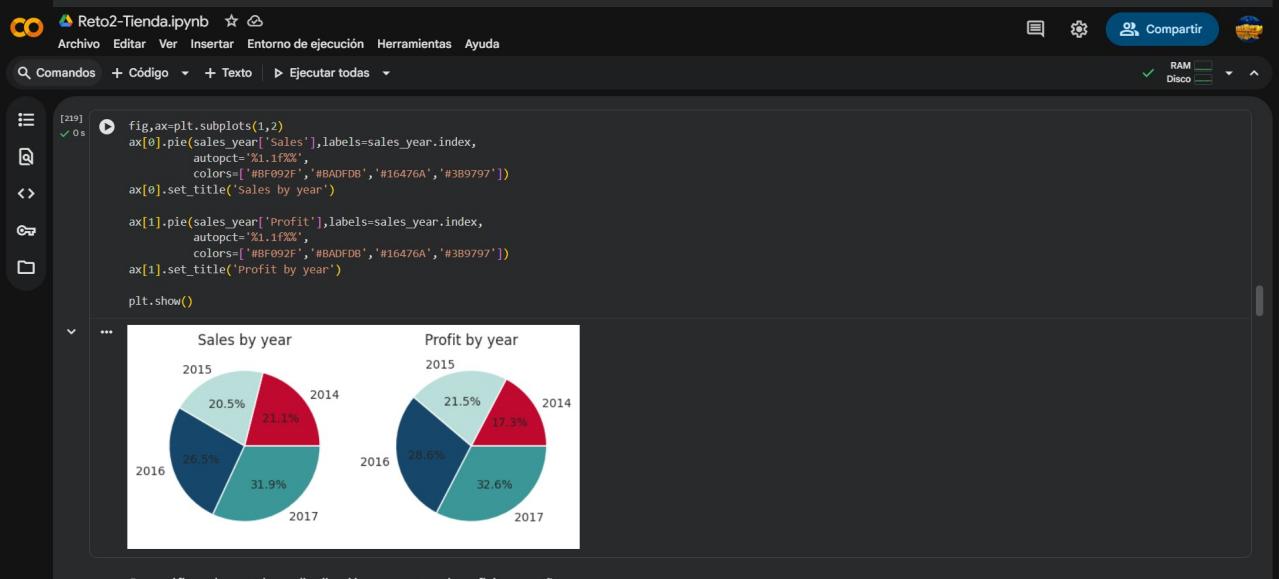
year_order	2014	2015	2016	2017	All							
month_order	1	2	3	4	5	6	7	8	9	10	11	12
1	14236.8950	18174.0756	18542.4910	43971.3740	9.492484e+04							
2	4519.8920	11951.4110	22978.8150	20301.1334	5.975125e+04							
3	55691.0090	38726.2520	51715.8750	58872.3528	2.050055e+05							
4	28295.3450	34195.2085	38750.0390	36521.5361	1.377621e+05							
5	23648.2870	30131.6865	56987.7280	44261.1102	1.550288e+05							
6	34595.1276	24797.2920	40344.5340	52981.7257	1.527187e+05							
7	33946.3930	28765.3250	39261.9630	45264.4160	1.472381e+05							

Reto2-Tienda.ipynb

8	27909.4685	36898.3322	31115.3743	63120.8880	1.590441e+05
9	81777.3508	64595.9180	73410.0249	87866.6520	3.076499e+05
10	31453.3930	31404.9235	59687.7450	77776.9232	2.003230e+05
11	78628.7167	75972.5635	79411.9658	118447.8250	3.524611e+05
12	69545.6205	74919.5212	96999.0430	83829.3188	3.252935e+05
All	484247.4981	470532.5090	609205.5980	733215.2552	2.297201e+06

Pasos siguientes: Generar código con sales_mon New interactive sheet

- Tabla tipo "matriz":
 - Cada fila es un mes.
 - Sales es lo que quiero calcular.
 - Cada columna será un año diferente.
 - margins=True -> Añade una fila y una columna extra llamadas All:
 - La fila All muestra la suma total por columna(ventas totales por año).
 - La columna All muestra la suma total por fila(ventas totales en un mes sumando todos los años).



Reto2-Tienda.ipynb

```
sales_state=df.groupby(['State','Category'])['Sales'].sum().sort_values(ascending=False).reset_index()
sales_state=sales_state.head(20)
sales_state
```

	State	Category	Sales
0	California	Technology	159271.0820
1	California	Furniture	156064.6015
2	California	Office Supplies	142351.9480
3	New York	Technology	127483.5000
4	New York	Furniture	93372.7290
5	New York	Office Supplies	90020.0420
6	Texas	Technology	65104.2240
7	Texas	Furniture	60593.2918
8	Washington	Technology	50536.7100
9	Washington	Furniture	48020.1520
10	Florida	Technology	46968.0360
11	Texas	Office Supplies	44490.5300
12	Pennsylvania	Technology	42215.2690
13	Washington	Office Supplies	40084.4080
14	Pennsylvania	Furniture	39354.9310
15	Michigan	Office Supplies	37723.7590
16	Ohio	Technology	35675.9920
17	Pennsylvania	Office Supplies	34941.7140
18	Illinois	Technology	31983.6730

Variables Terminal 22:57 Python 3

Reto2-Tienda.ipynb

Pasos siguientes: Generar código con sales_state New interactive sheet

- Agrupa ventas por estado y categoría.
- Ordena descendente y toma top 20 combinaciones.

Reto2-Tienda.ipynb

```
[222] g=sns.catplot(data=sales_state,x='State',y='Sales',
                 kind='bar', color='red',col='Category')
g.fig.supertitle('Sales by State and Category',y=1.05)
g.set_x_labels(rotation=90)
plt.show()
```

Sales by State and Category

Category = Technology

State	Sales
California	160000
New York	125000
Texas	60000
Washington	50000
Florida	45000
Pennsylvania	40000
Michigan	35000
Ohio	35000
Illinois	30000

Category = Furniture

State	Sales
California	160000
New York	90000
Texas	60000
Washington	50000
Florida	45000
Pennsylvania	40000
Michigan	35000
Ohio	35000
Illinois	30000

Category = Office Supplies

State	Sales
California	140000
New York	85000
Texas	55000
Washington	45000
Florida	40000
Pennsylvania	35000
Michigan	30000
Ohio	30000
Illinois	30000

Variables Terminal 22:57 Python 3

Reto2-Tienda.ipynb

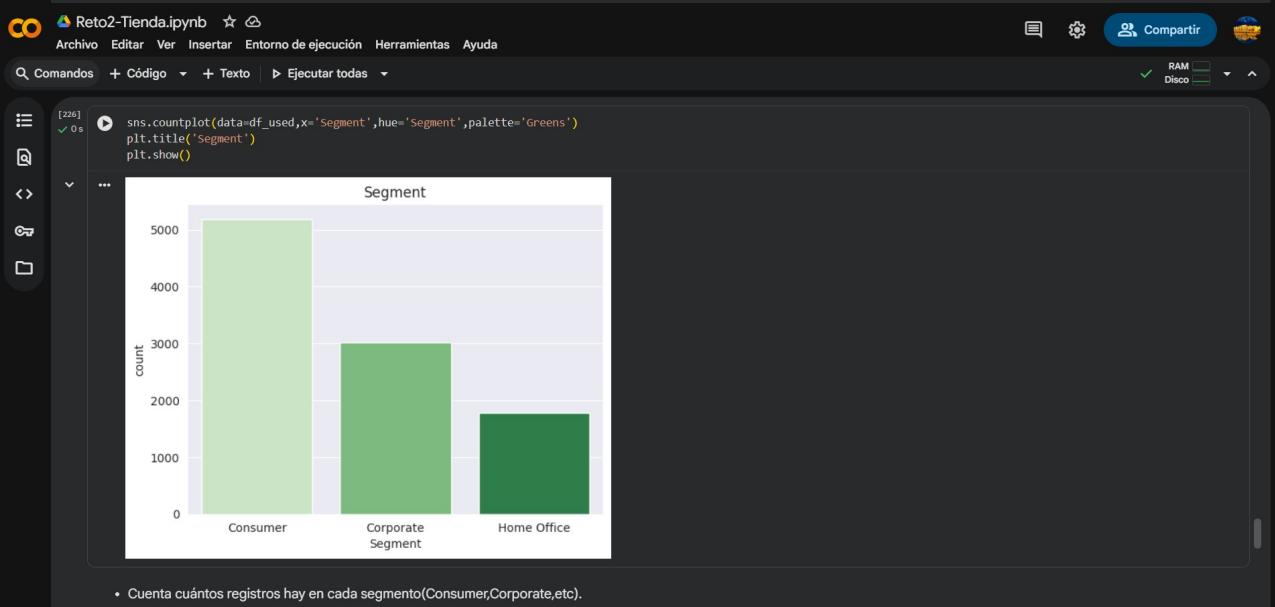
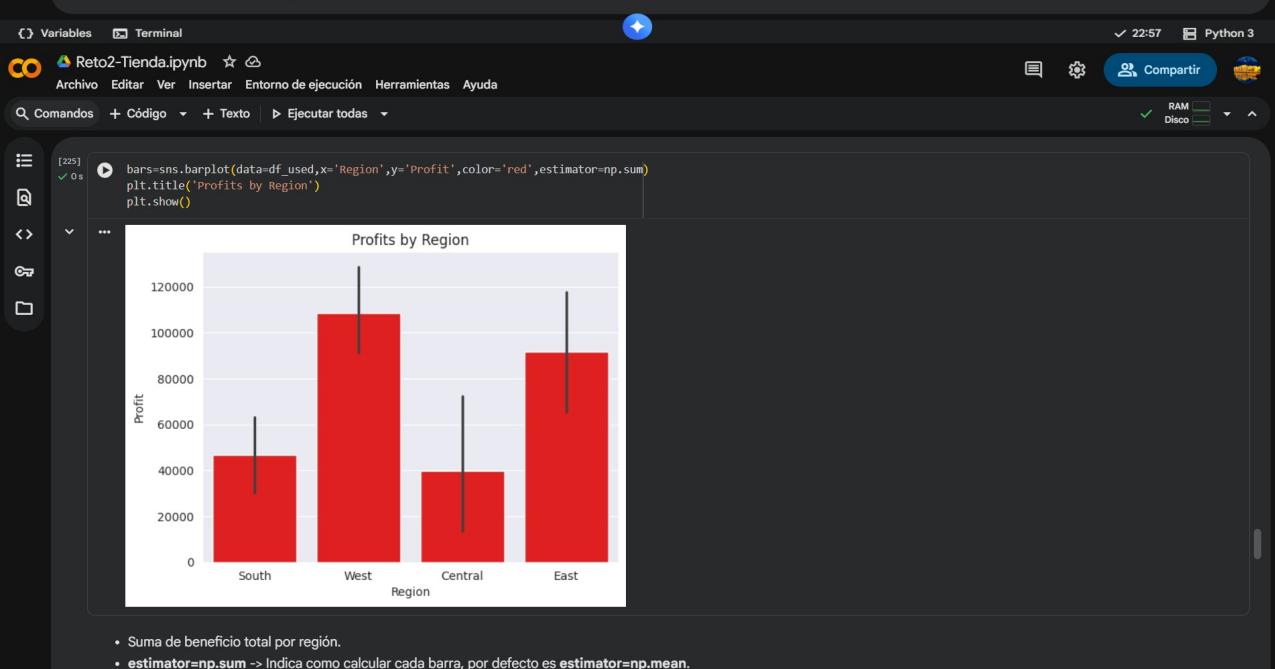
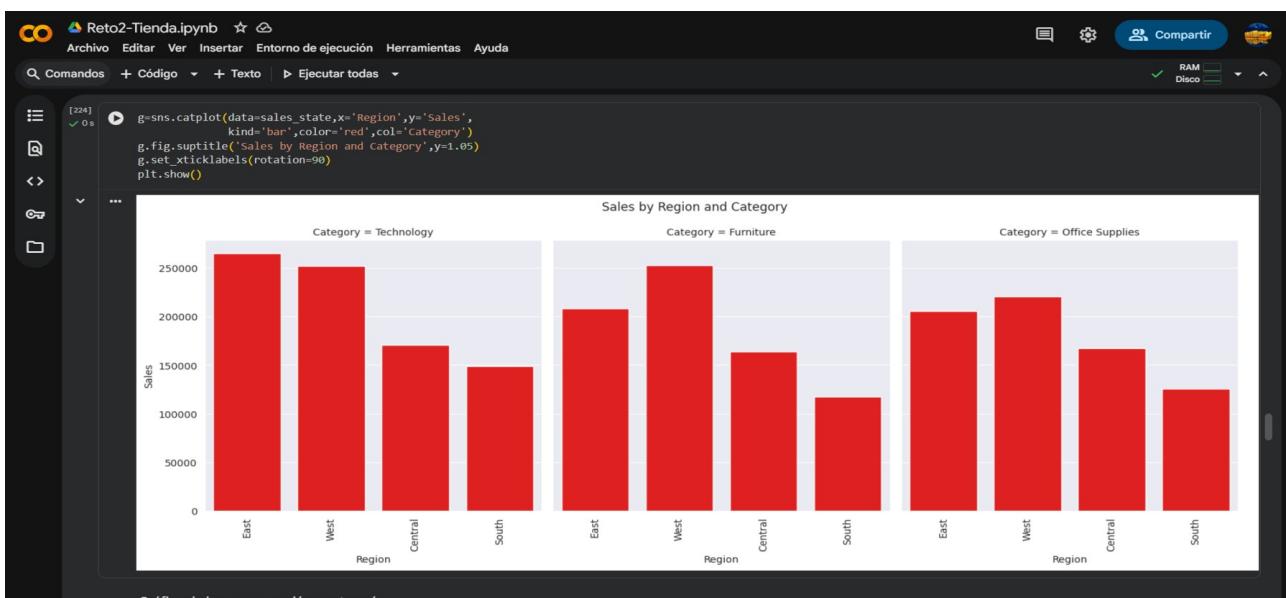
Pasos siguientes: Generar código con sales_state New interactive sheet

- Igual pero agrupando por región + categoría.

```
[223] sales_state=df.groupby(['Region','Category'])['Sales'].sum().sort_values(ascending=False).reset_index()
sales_state=sales_state.head(20)
sales_state
```

	Region	Category	Sales
0	East	Technology	264973.9810
1	West	Furniture	252612.7435
2	West	Technology	251991.8320
3	West	Office Supplies	220853.2490
4	East	Furniture	208291.2040
5	East	Office Supplies	205516.0560
6	Central	Technology	170416.3120
7	Central	Office Supplies	167026.4150
8	Central	Furniture	163797.1638
9	South	Technology	148771.9080
10	South	Office Supplies	125651.3130
11	South	Furniture	117298.6840

Variables Terminal 22:57 Python 3



Reto2-Tienda.ipynb

```
[227] df_used.pivot_table(index='Segment',values='Sales',aggfunc='sum').sort_values(by='Sales',ascending=False)
```

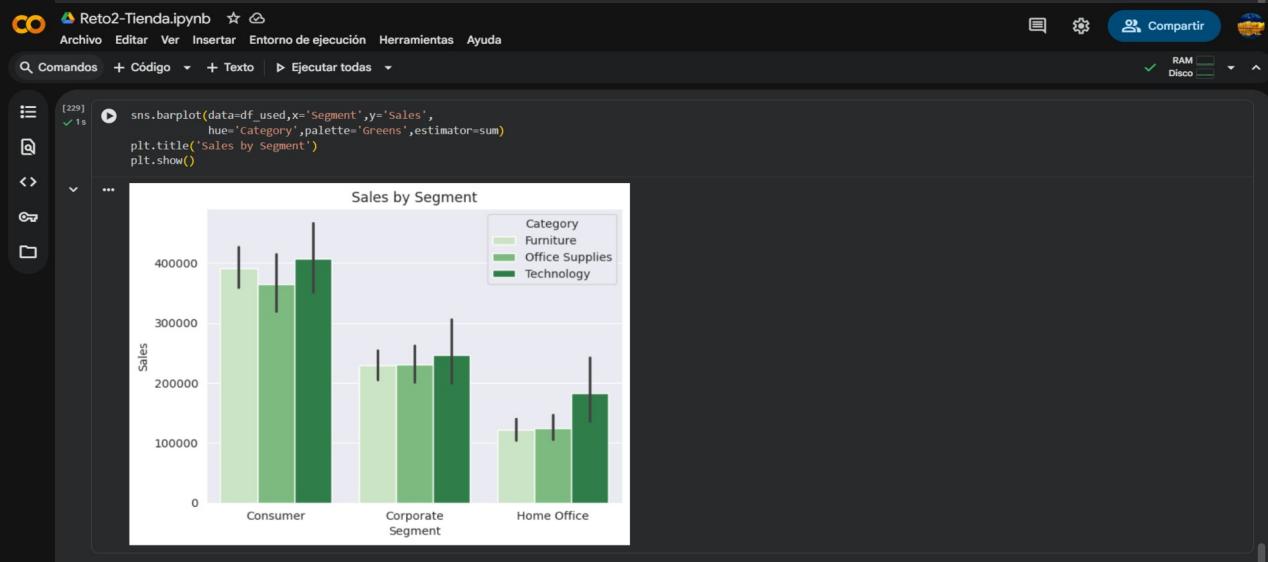
Segment	Sales
Consumer	1.161401e+06
Corporate	7.061464e+05
Home Office	4.296531e+05

- Ventas por segmento, ordenadas de mayor a menor.

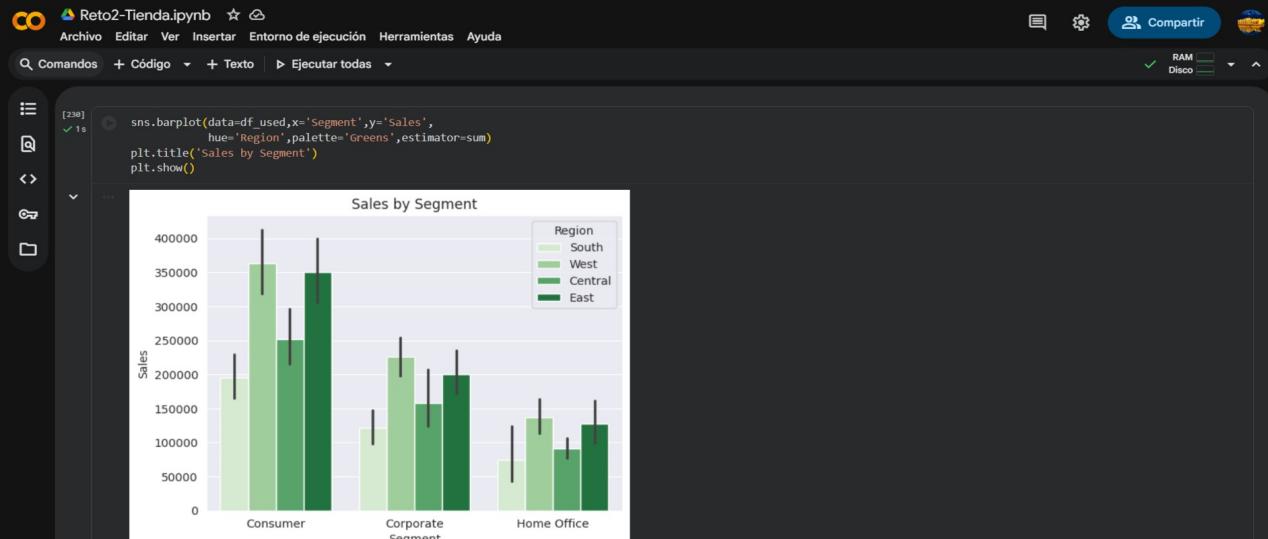
```
[228] df_used.pivot_table(index='Segment',values='Profit',aggfunc='sum').sort_values(by='Profit',ascending=False)
```

Segment	Profit
Consumer	134119.2092
Corporate	91979.1340
Home Office	60298.6785

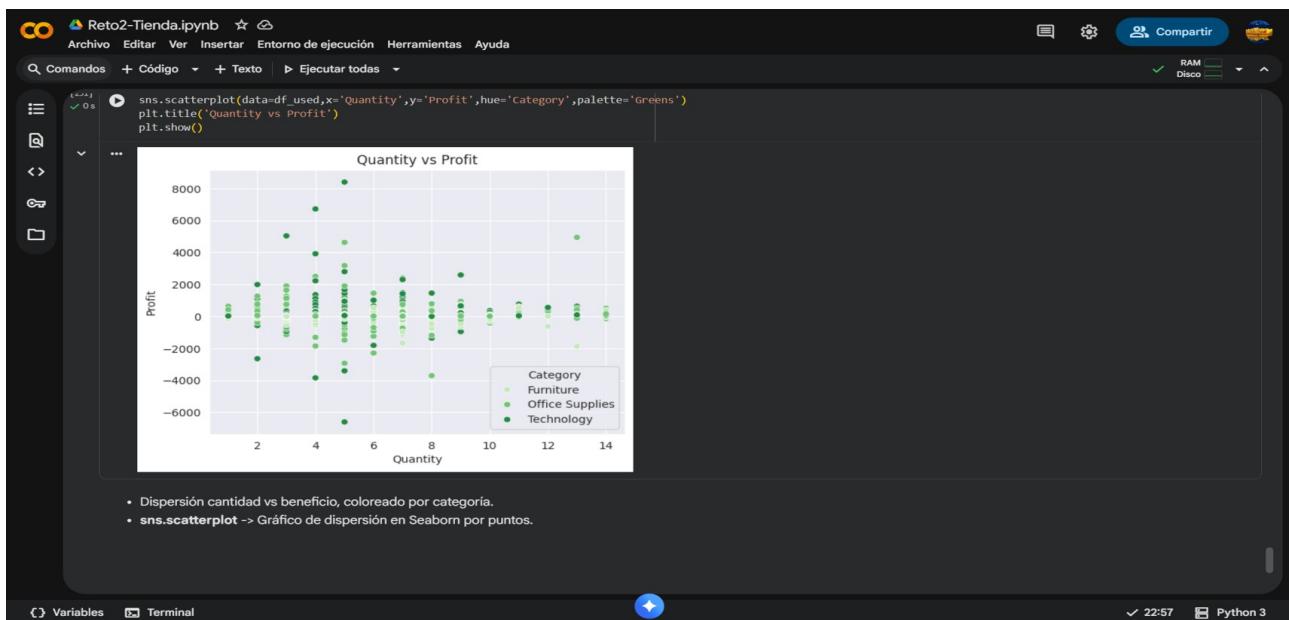
- Beneficio por segmento.



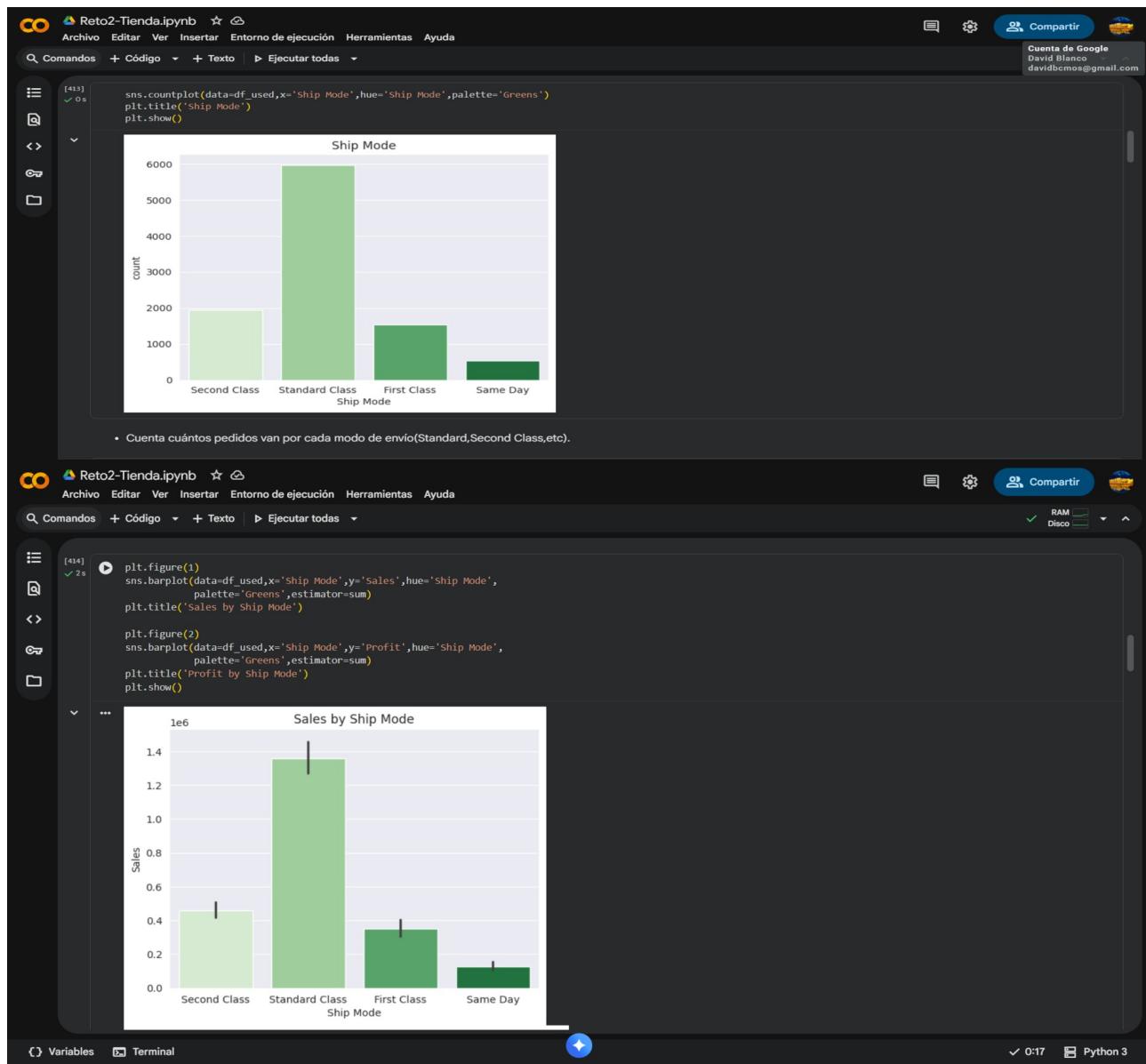
- Gráfico de barras apiladas por segmento y categoría(sales).

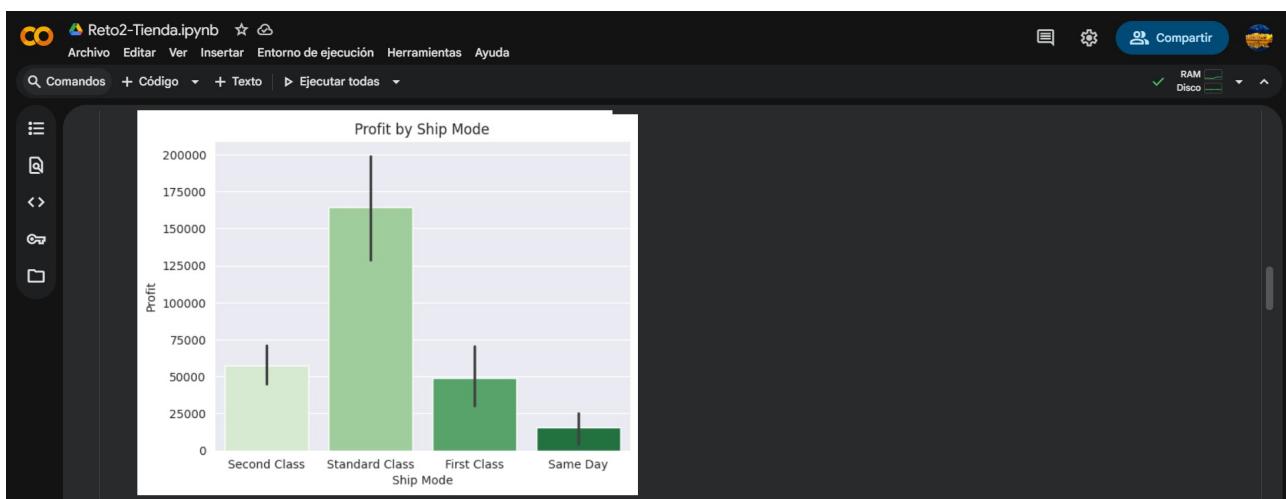


- Similar pero separa por región.

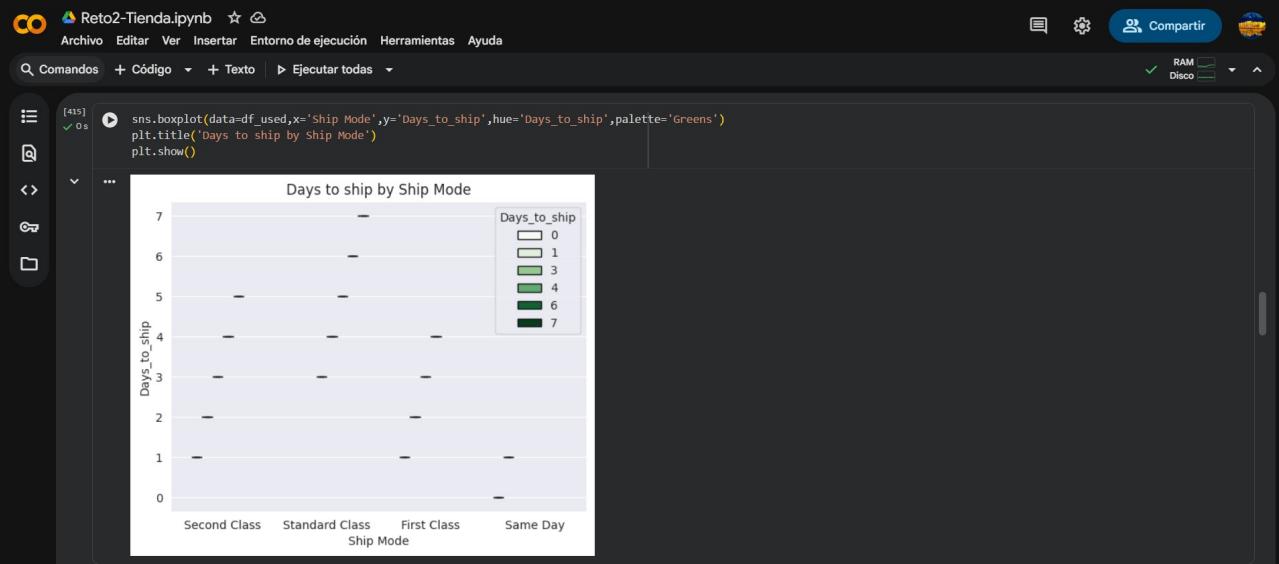


9. SHIP MODE, DISTRIBUCIONES Y CORRELACIÓN

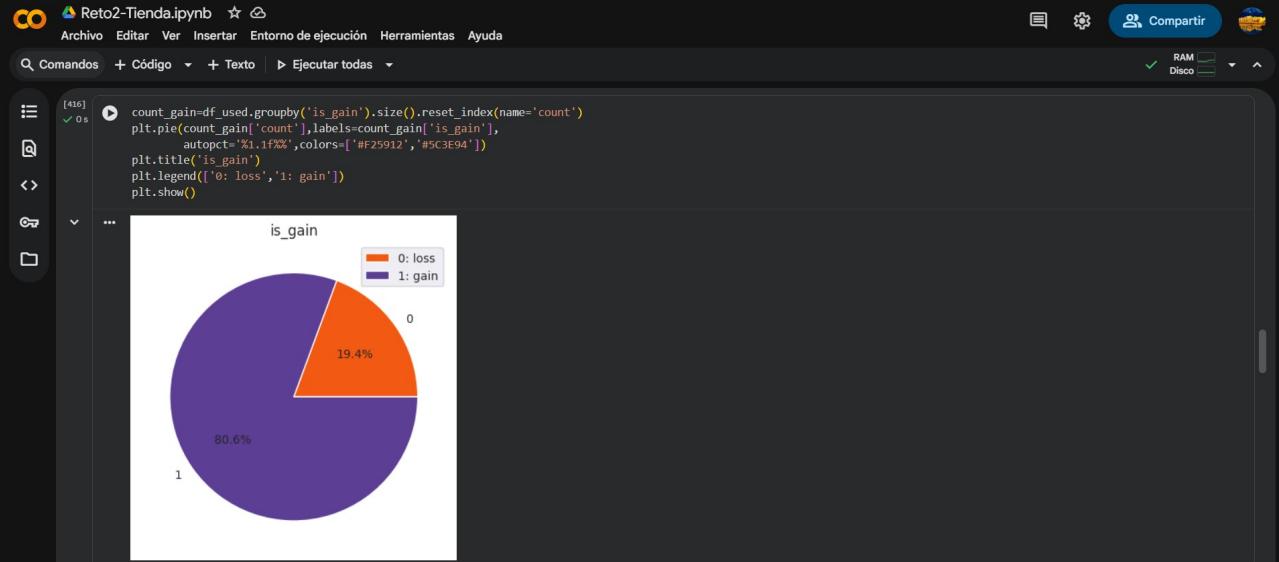




- plt.figure(1) -> Gráfico 1: ventas por modo de envío.
- plt.figure(2) -> Gráfico 2: beneficios por modo de envío.



- sns.boxplot -> Gráfico de cajas de Seaborn.
- Boxplot del nº de días para enviar según el modo de envío.



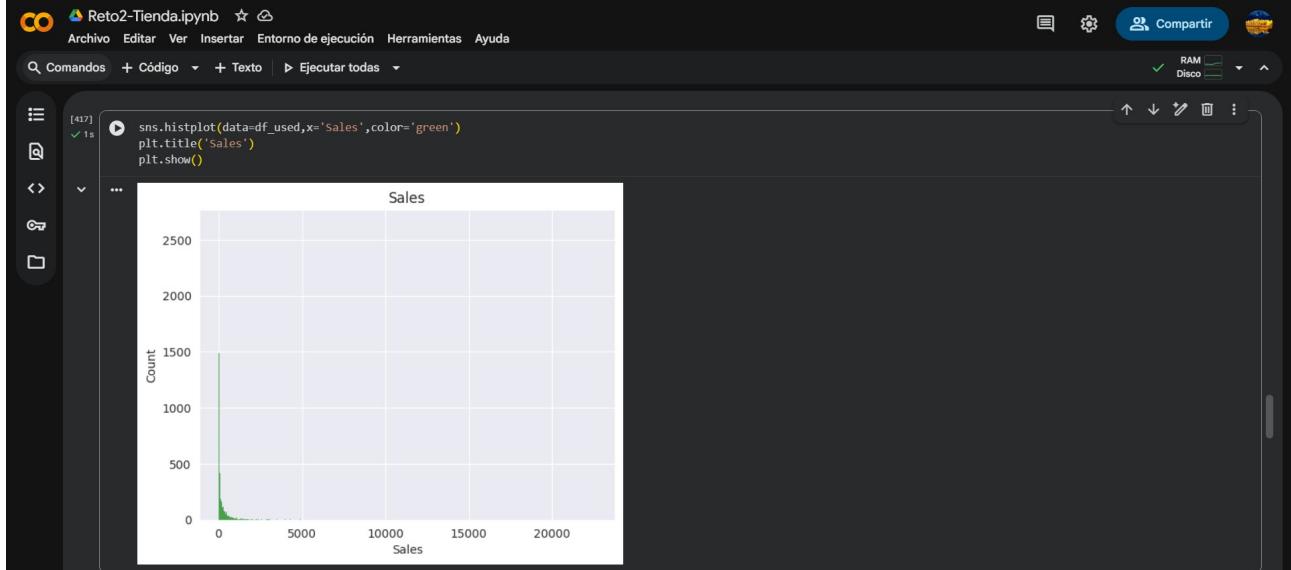
- df_used.groupby('is_gain') -> Agrupa el dataframe por la columna is_gain.
- is_gain tiene valores:
 - 1 -> el pedido generó beneficio.
 - 0 -> el pedido generó pérdida.

Retorno de ejecución

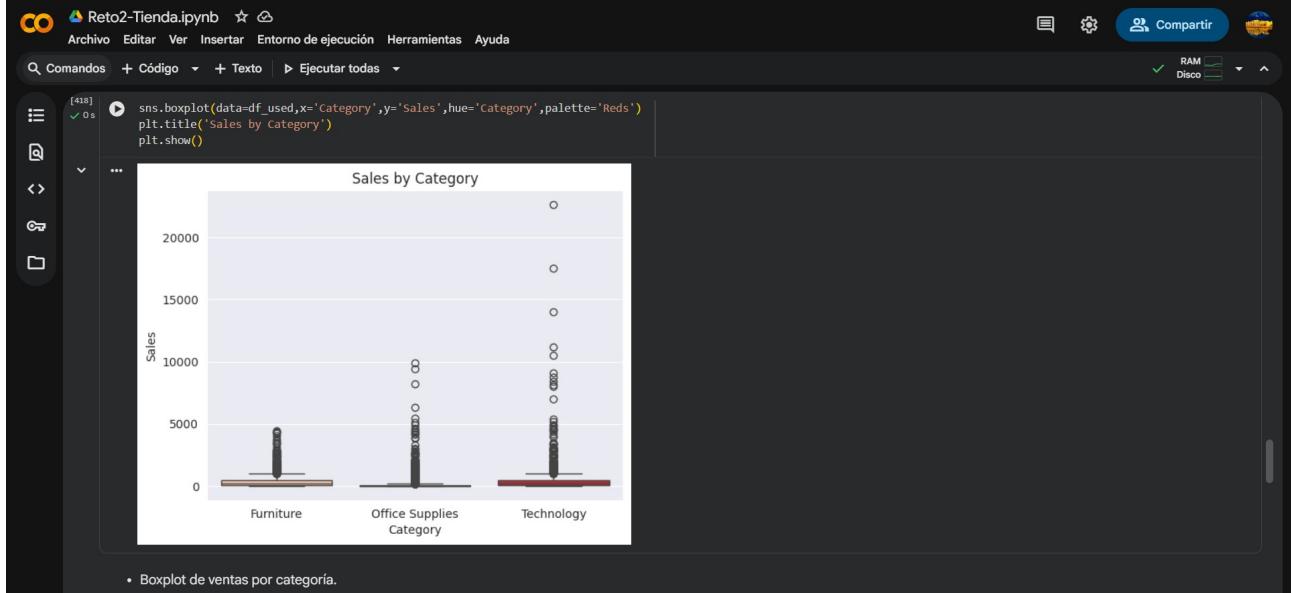
```
• groupby crea dos grupos:

- Grupo 0: todas las filas donde is_gain=0.
- Grupo 1:todas las filas donde is_gain=1.

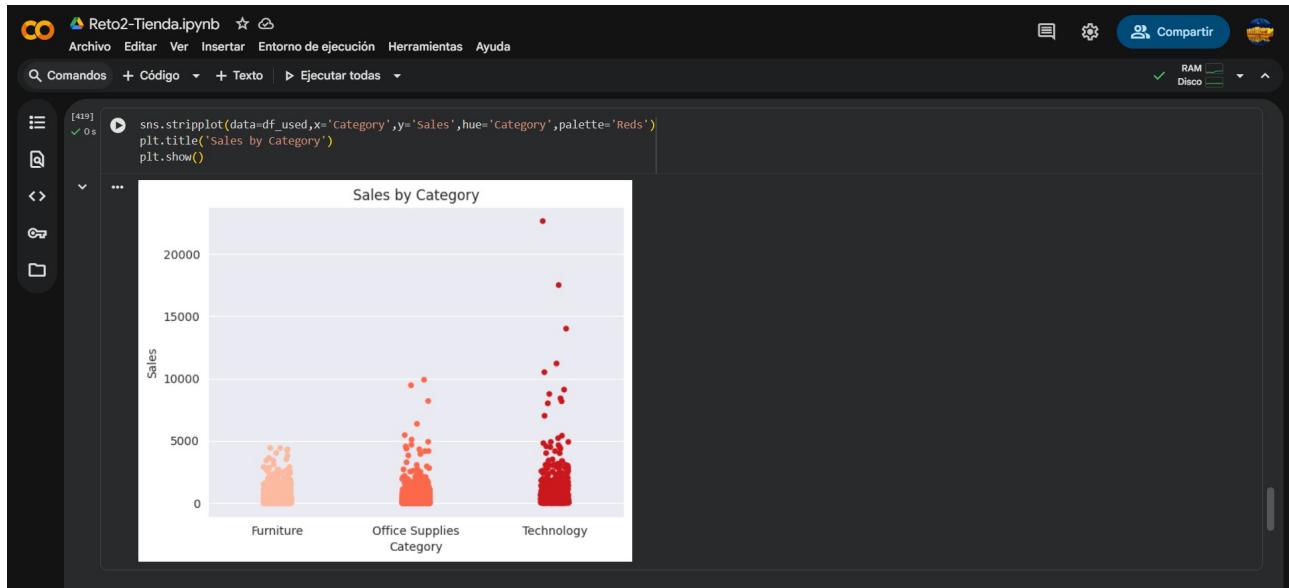
• size() -> Cuenta cuántas filas hay en cada grupo.  
• .reset_index(name='count') -> Convierte ese dataframe ordenado con dos columnas y se guarda en la variable count_gain.  
• count_gain['count'] -> Son los nº que va a usar para calcular el tamaño de la porción en el gráfico de pastel.  
• labels=count_gain['is_gain'] -> Etiquetas en el pastel(0 ó 1).  
• plt.legend(['0: loss','1: gain']) -> Añade leyenda manual para que se explique mejor.
```



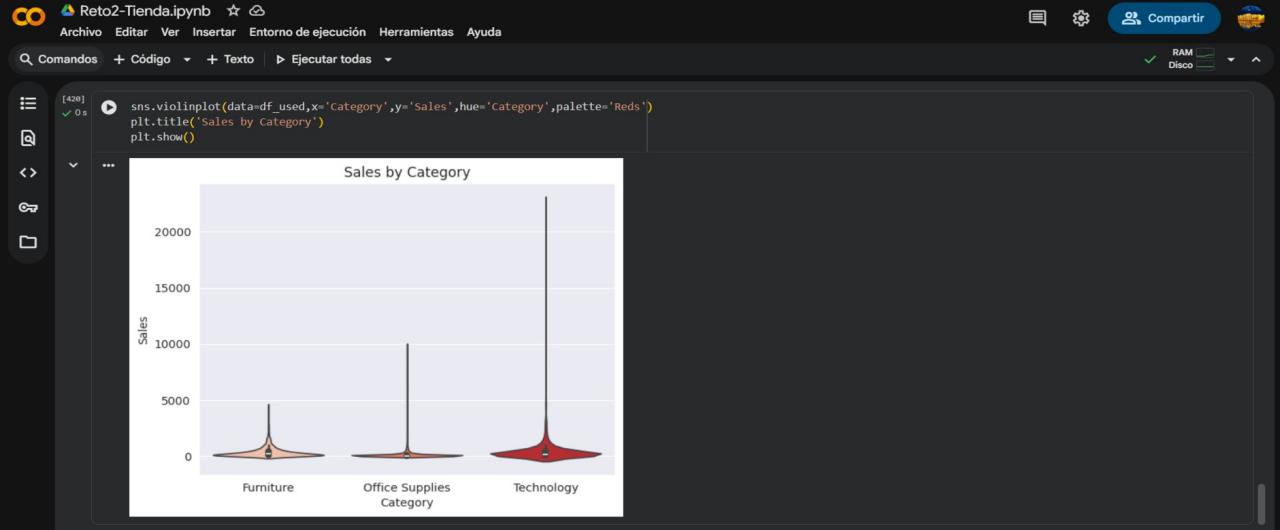
- sns.histplot -> Histograma de Seaborn.
- Crea un histograma de ventas.



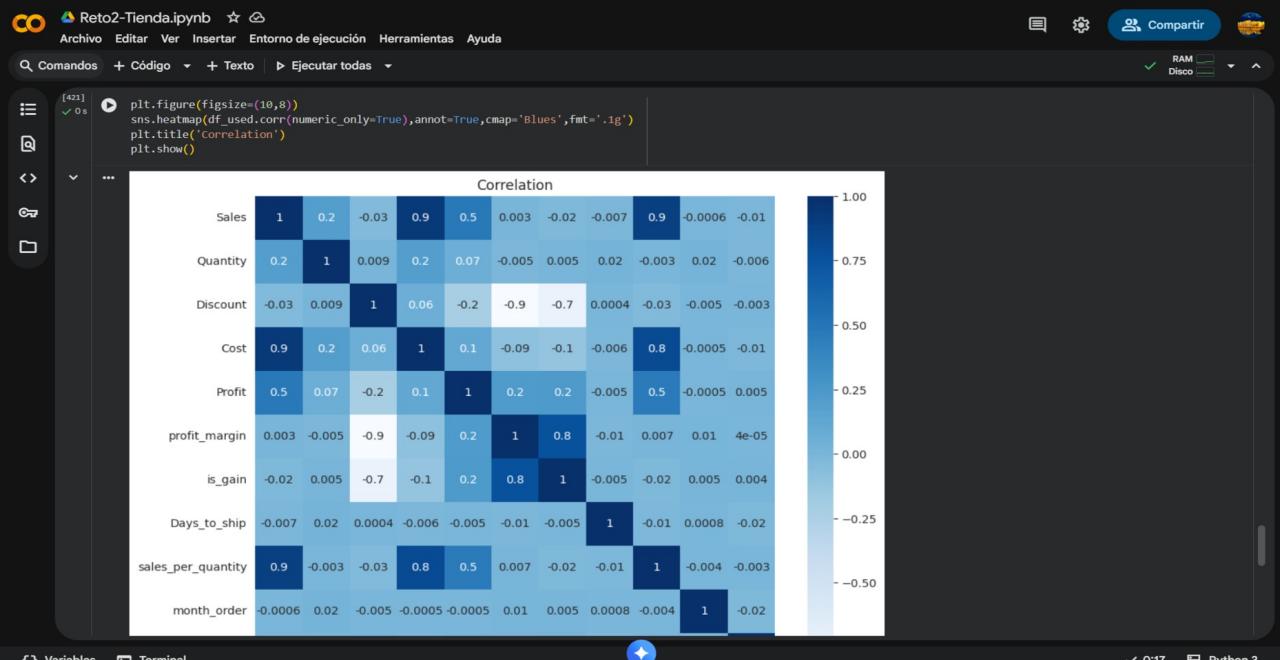
- Boxplot de ventas por categoría.

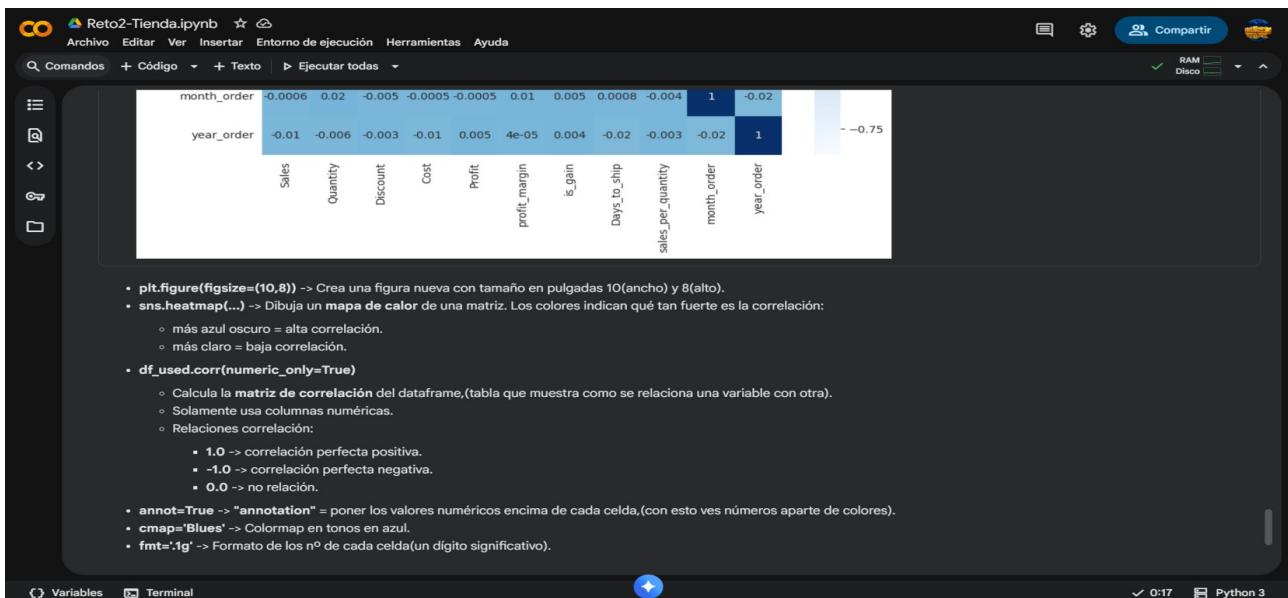


- `sns.stripplot` -> Gráfico de puntos donde cada punto representa una fila del dataset.



- `sns.violinplot` -> Gráfico que muestra la forma de distribución (igual a un histograma), pero vertical y suavizado.





10. EDA AUTOMÁTICO (YDATA-PROFILING)

```
#Instalar lib ydata-profiling
!pip install ydata-profiling

try:
    from ydata_profiling import ProfileReport

    profile=ProfileReport(
        df,
        title='EDA Automático - Store',
        explorative=True
    )
    profile.to_file('eda_automatico_store.html')
    print('Informe generado: eda_automatico_store.html')
except Exception as e:
    print("No se pudo generar el perfil automático. Error:")
    print(e)

    • Collecting ydata-profiling
    Downloading ydata_profiling-4.18.0-py2.py3-none-any.whl.metadata (22 kB)
    Requirement already satisfied: scipy<1.17,>=1.8 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.16.3)
    Requirement already satisfied: pandas<1.4.0,<3.0,>1.5 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.10.0)
    Requirement already satisfied: matplotlib<3.10,>=3.5 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.10.0)
    Requirement already satisfied: pydantic<3,>2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.11.10)
    Requirement already satisfied: pyYAML<6.1,>=6.0.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (6.0.3)
    Requirement already satisfied: jinja2<3.2,>=3.1.6 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.1.6)
    Collecting visions<0.8.2,>=0.7.5 (from visions[type_image_path]<0.8.2,>=0.7.5-ydata-profiling)
    Downloading visions-0.8.1-py3-none-any.whl.metadata (11 kB)
    Requirement already satisfied: numpy<2.12,>=1.22 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.0.2)
    Collecting minify-html<0.15.0 (from ydata-profiling)
    Downloading minify_html-0.18.0-py3.12-manylinux2_21_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
    Collecting filetype<1.0.0 (from ydata-profiling)
    Downloading filetype-1.2.0-py2.py3-none-any.whl.metadata (6.5 kB)
    Collecting phik<0.13,>=0.12.5 (from ydata-profiling)
    Downloading phik-0.12.5-cp312-cp312-manylinux_2_24_x86_64_manylinux_2_28_x86_64.whl.metadata (5.6 kB)
    Requirement already satisfied: requests<3,>=2.32.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.32.4)
    Requirement already satisfied: tqdm<5,>=4.66.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.67.1)
    Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.13.2)
    Collecting multimethod<2.1.1 (from ydata-profiling)
```

https://accounts.google.com/signOutOptions?hl=es&continue=https://colab.research.google.com/%3Fauthuser%3D0&ec=GBRAQm

```
✓ 0:58 Python 3
```

```
#Instalar lib ydata-profiling
!pip install ydata-profiling

try:
    from ydata_profiling import ProfileReport

    profile=ProfileReport(
        df,
        title='EDA Automático - Store',
        explorative=True
    )
    profile.to_file('eda_automatico_store.html')
    print('Informe generado: eda_automatico_store.html')
except Exception as e:
    print("No se pudo generar el perfil automático. Error:")
    print(e)

    • Collecting ydata-profiling
    Downloading ydata_profiling-4.18.0-py2.py3-none-any.whl.metadata (22 kB)
    Requirement already satisfied: scipy<1.17,>=1.8 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.16.3)
    Requirement already satisfied: pandas<1.4.0,<3.0,>1.5 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.10.0)
    Requirement already satisfied: matplotlib<3.10,>=3.5 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.10.0)
    Requirement already satisfied: pydantic<3,>2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.11.10)
    Requirement already satisfied: pyYAML<6.1,>=6.0.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (6.0.3)
    Requirement already satisfied: jinja2<3.2,>=3.1.6 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.1.6)
    Collecting visions<0.8.2,>=0.7.5 (from visions[type_image_path]<0.8.2,>=0.7.5-ydata-profiling)
    Downloading visions-0.8.1-py3-none-any.whl.metadata (11 kB)
    Requirement already satisfied: numpy<2.12,>=1.22 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.0.2)
    Collecting minify-html<0.15.0 (from ydata-profiling)
    Downloading minify_html-0.18.0-py3.12-manylinux2_21_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
    Collecting filetype<1.0.0 (from ydata-profiling)
    Downloading filetype-1.2.0-py2.py3-none-any.whl.metadata (6.5 kB)
    Collecting phik<0.13,>=0.12.5 (from ydata-profiling)
    Downloading phik-0.12.5-cp312-cp312-manylinux_2_24_x86_64_manylinux_2_28_x86_64.whl.metadata (5.6 kB)
    Requirement already satisfied: requests<3,>=2.32.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.32.4)
    Requirement already satisfied: tqdm<5,>=4.66.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.67.1)
    Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.13.2)
    Collecting multimethod<2.1.1 (from ydata-profiling)
```

Installing collected packages: puremagic, minify-html, filetype, multimethod, dacie, imagehash, visions, phik, ydata-profiling
Successfully installed dacite-1.9.2 filetype-1.2.0 imagehash-4.3.2 minify-html-0.18.1 multimethod-1.12 phik-0.12.5 puremagic-1.30 visions-0.8.1 ydata-profiling-4.18.0
Summarize dataset 100%

0% | 0/28 [00:00:00, ?it/s] 679.7/679.7 kB 37.0 MB/s eta 0:00:00
Downloaded visions-0.8.1-py3-none-any.whl (105 kB)
100% | 105.4/105.4 kB 6.7 MB/s eta 0:00:00
Downloaded puremagic-1.30-py3-none-any.whl (43 kB)
100% | 43.3/43.3 kB 2.3 MB/s eta 0:00:00
Generating report structure: 100% 1/1 [00:07<00:00, 7.36it/s]
Render HTML 100% 1/1 [00:00<00:00, 1.61it/s]
Export report to file: 100% 1/1 [00:00<00:00, 26.80it/s]
Informe generado: eda_automatico_store.html

- Instala librería **ydata-profiling**.
- Intenta importar **ProfileReport**.
- **ProfileReport(...)** genera un informe EDA completo:
 - Distribuciones,nulos,correlaciones,etc.
- **to_file(...)** -> guarda el informe en HTML
- Si falla capture la excepción y la imprime.

11. INGENIERÍA DE CARACTERÍSTICAS Y PREPARACIÓN PARA ML (SCIKIT-LEARN)

11.1. Selección de atributos

The screenshot shows two versions of a Jupyter Notebook cell in Google Colab. The top part displays the code and its execution output, while the bottom part shows the code with explanatory notes.

```
[424]: from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd

# Aseguramos que trabajamos con columnas numéricas y la existencia de Profit
if 'Profit' in df.columns:
    X_num = df.select_dtypes(include=['number']).drop(columns=['Profit'])
    y = df['Profit']

    # Rellenamos de manera sencilla los nulos
    X_num = X_num.fillna(X_num.median())

    # 1.- SelectKBest (relación lineal)
    selector = SelectKBest(score_func=f_regression, k=min(5, X_num.shape[1]))
    selector.fit(X_num, y)
    selected_cols_kbest = X_num.columns[selector.get_support()].tolist()
    print('Top mejores columnas numéricas según SelectKBest:', selected_cols_kbest)

    # 2.- Importancia de características con RandomForest (no lineal)
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_num, y)
    importances = pd.Series(rf.feature_importances_, index=X_num.columns).sort_values(ascending=False)
    print('\nTop Importancia de variables numéricas (RandomForest):')
    print(importances.head(10))
else:
    print('No se encontró la columna Profit en el dataframe')

... Top mejores columnas numéricas según SelectKBest: ['Sales', 'Discount', 'profit_margin', 'is_gain', 'sales_per_quantity']

Top Importancia de variables numéricas (RandomForest):
Sales          0.471834
Cost           0.225899
profit_margin  0.214574
sales_per_quantity  0.032256
```

The bottom part of the screenshot contains explanatory notes:

- Imports scikit-learn (sklearn) and Pandas (pd).
- Checks if 'Profit' exists in the DataFrame.
- Creates a copy of the DataFrame (X_num) dropping the 'Profit' column.
- Fills null values in X_num with the median.
- Performs SelectKBest on X_num and y to find the top 5 most correlated features.
- Trains a RandomForestRegressor on X_num and y.
- Prints the feature importances.
- Notes that the first part of the code is for linear correlation (SelectKBest) and the second part is for non-linear importance (RandomForestRegressor).
- Explains the parameters used in SelectKBest: score_func=f_regression (F-test for regression), k=min(5, X_num.shape[1]) (selects up to 5 features).
- Explains the parameters used in RandomForestRegressor: n_estimators=100, random_state=42.
- Explains the importance calculation: pd.Series(rf.feature_importances_, index=X_num.columns).sort_values(ascending=False).
- Notes that the output shows the top 10 most important features.

11.2. Transformaciones (log y escalado)

The screenshot shows a Google Colab notebook titled "Reto2-Tienda.ipynb". The code cell [69] contains Python code for transforming data:

```
# Transformación logarítmica de Sales
if 'Sales' in df.columns:
    df['Sales_log'] = np.log1p(df['Sales'])
    print(df[['Sales','Sales_log']].head())

# Escalado estándar de Profit
if 'Profit' in df.columns:
    scaler = StandardScaler()
    df['Profit_scaled'] = scaler.fit_transform(df[['Profit']])
    print(df[['Profit','Profit_scaled']].head())
```

The output shows two dataframes:

	Sales	Sales_log
0	261.9600	5.572002
1	731.9400	6.597064
2	14.6200	2.748552
3	957.5775	6.865450
4	22.3680	3.151368

	Profit	Profit_scaled
0	41.9136	0.056593
1	219.5820	0.815054
2	6.8714	-0.093002
3	-383.0310	-1.757484
4	2.5164	-0.111593

A callout box highlights the following notes:

- `np.log1p(x)` -> aplica $\log(1 + x)$ para estabilizar distribuciones muy sesgadas.
- Crea `Sales log`.
- `StandardScaler()` -> consiste en que `Profit` tenga media 0 y varianza 1.
- `fit_transform` -> aprende media/desviación y transforma los datos.
- Imprimen tanto la transformación como el escalado estándar.

At the bottom, the URL is https://accounts.google.com/SignOutOptions?hl=es&continue=https://colab.research.google.com/%3Fauthuser%3D0&ec=GBRAqQM, the time is 16:52, and the Python version is Python 3.

11.3. Nuevos atributos

The screenshot shows a Google Colab notebook titled "Reto2-Tienda.ipynb". The code cell [72] contains Python code for creating new attributes:

```
# Atributo 1: margen de beneficio si no existiera
if 'Sales' in df.columns and 'Profit' in df.columns:
    if 'Profit_Margin' not in df.columns and 'profit_margin' not in df.columns:
        df['Profit_Margin'] = df['Profit'] / df['Sales']
        print(df[['Sales','Profit','Profit_Margin']].head())
    else:
        print('Ya existe una columna de margen de beneficio (Profit_Margin / profit_margin)')

# Atributo 2: nivel de descuento (bajo/medio/alto)
if 'Discount' in df.columns:
    df['Discount_Level'] = pd.cut(
        df['Discount'],
        bins=[-0.001,0.1,0.3,1],
        labels=['Bajo','Medio','Alto']
    )
    print(df[['Discount','Discount_Level']].head())

# Atributo 3: variables temporales.
if 'Order Date' in df.columns:
    if not np.issubdtype(df['Order Date'].dtype, np.datetime64):
        df['Order Date'] = pd.to_datetime(df['Order Date'])
    df['Order Year'] = df['Order Date'].dt.year
    df['Order Month'] = df['Order Date'].dt.month
    df['Order DayofWeek'] = df['Order Date'].dt.dayofweek
    print(df[['Order Date','Order Year','Order Month','Order DayofWeek']].head())
```

Atributo 1:

- Comprobación de que el DataFrame contenga Sales y Profit.
- Evita duplicados si ya existiera Profit_Margin o profit_margin.
- Margen de beneficio = beneficio / ventas.
- Imprime las 5 primeras líneas.

Atributo 2:

- Comprobación de que exista Discount.
- pd.cut(df['Discount'].bins[-0.001,0.1,0.3,1],labels=['Bajo','Medio','Alto']) -> Se categoriza el descuento en 3 niveles.
 - pd.cut divide un valor numérico en rangos(bins).
 - -0.001 - 0.1 -> 'Bajo'.
 - 0.1 - 0.3 -> 'Medio'.
 - 0.3 - 1 -> 'Alto'.

Esto convierte el descuento en una variable categorica, ideal para modelos ML.

- Muestra las 5 primeras filas.

Atributo 3:

- Comprueba que existe Order Date
- if not np.issubdtype(df['Order Date'].dtype, np.datetime64):
 - Si la columna Order Date no está en formato fecha entonces la convierte con pd.to_datetime.
 - Si Order Date fuera texto no se podría extraer año, mes, etc.
- Crea columnas para:
 - Año (Order Year) -> df['Order Date'].dt.year
 - Mes (Order Month)-> df['Order Date'].dt.month
 - Día de la semana (Order DayOfWeek)-> df['Order Date'].dt.dayofweek
- Imprime 5 filas para verificar.

11.4. Dataset listo para ML con one-hot + train_test_split

```
[73] # Copia para no perder el dataframe original.
data=df.copy()

# Detectar columnas categóricas.
cat_cols = []
for c in ['Category', 'Sub-Category', 'Segment', 'Region']:
    if c in data.columns:
        cat_cols.append(c)

# One-hot encoding.
if cat_cols:
    data = pd.get_dummies(data, columns=cat_cols, drop_first=True)

# Eliminar columnas no numéricas residuales.
data=data.select_dtypes(include=['number'])

# Separar Profit como variable a predecir.
if 'Profit' in data.columns:
    y=data['Profit']
    X=data.drop(columns=['Profit'])

    # Rellenamos nulos por mediana.
    X=X.fillna(X.median())

    # Train/test split
    X_train,X_test,y_train,y_test=train_test_split(
        X,y,
        test_size=0.2,
        random_state=42
    )

    print('Shape X_train:', X_train.shape)
    print('Shape X_test:', X_test.shape)
else:
    print('No se encontro la columna Profit en los datos numéricos codificados')
```

Reto2-Tienda.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

Compartir RAM Disco

Comandos Código Texto Ejecutar todas

Shape X_train: (7995, 18)
Shape X_test: (1999, 18)

- data = df.copy() -> copia el dataframe y así trabajamos sobre una copia para no romper la original.
- Preparamos cat_cols como una lista de columnas categóricas a codificar.
- if c in data.columns: -> añade a la lista las que realmente existen.
- pd.get_dummies(...,drop_first=True):
 - One-hot encoding -> convierte las categorías en columnas 0/1.
 - drop_first=True -> evita multicolinealidad eliminando una categoría por variable.
- data.select_dtypes(include=['number']) -> Se queda con columnas numéricas(dummies, numéricas originales).
- Comprueba que Profit sigue existiendo:
 - y=data['Profit'] -> objetivo.
 - X=data.drop(columns=['Profit']) -> variables explicativas.
- X.fillna(X.median()) -> rellena nulos numéricos con la mediana y evitamos que el modelo reviente.
- Usamos scikit-learn:
 - train_test_split(X,y,test_size=0.2,random_state=42) -> divide datos en:
 - 80% entrenamiento (X_train,y_train).
 - 20% test (X_test,y_test).
 - random_state=42 -> reproducible, asegura que la división sea siempre igual.
 - Imprime los tamaños de train y test.
 - Si no se encontrara Profit, muestra un mensaje de error.

Variables Terminal 18:21 Python 3