

# ***Análisis de Ventas***

## ***Base de Datos 'sales\_company'***

*Entrega de Avance – Parte 1*

*Productos, Ventas y Clientes*

*Carlos Juan Figueroa*

*Bootcamp: Soy Henry – Data Engineer*

*Fecha de entrega: 11 de junio de 2025*



**DATA ENGINEER**

# Índice

1. *Introducción*
2. *Metodología*
3. *Resultados y Análisis*
  - 3.1. *Productos y Vendedores Top-5*
  - 3.2. *Clientes Únicos y Penetración*
  - 3.3. *Categorías de los Productos Top-5*
  - 3.4. *Top-10 Productos Globales*
4. *Monitoreo Automático de Ventas – Trigger*
5. *Optimización de Consultas*
6. *Feature Engineering y Limpieza de Datos*
  - 6.1 Cálculo de *TotalPriceCalculated*
  - 6.2 *Detección de Outliers*
  - 6.3 *Variables Temporales*
  - 6.4 *Edad y Experiencia del Empleado*
7. *Diseño orientado a objetos y análisis dinámico de ventas*

# Introducción

Este informe documenta el proceso integral de análisis y optimización aplicado a la base de datos *sales\_company*.

El trabajo se estructuró en tres ejes:

- **Analítico:** identificar los productos y vendedores con mejor desempeño y evaluar la distribución de las ventas por cliente y categoría.
- **Operativo:** mejorar la velocidad de respuesta de las consultas de negocio mediante estrategias de indexación.
- **Enriquecimiento:** generar variables derivadas y automatizar alertas que añadan valor sin alterar la métrica objetivo.

## Metodología

### 1. Modelado SQL

- Definición de tablas y claves (sentencias DDL).
- Creación de triggers para alertas de alta rotación.
- Consultas de agregación y vistas intermedias utilizando funciones de ventana.
- 

### 2. Procesamiento en Python (Pandas)

- Limpieza y normalización de datos.
- Cálculo de precio neto y variables temporales.
- Detección de valores atípicos mediante el rango intercuartílico (IQR).

### 3. Optimización de rendimiento

- Diseño y prueba de índices individuales y compuestos.
- Medición comparativa con *EXPLAIN ANALYZE* y cronómetro para validar mejoras.

# Resultados y Análisis

## 3.1 Productos y Vendedores Top-5

**Pregunta 1.a** – ¿Cuáles fueron los 5 productos más vendidos y qué vendedor lideró las ventas de cada uno?

Los cinco productos más vendidos y sus vendedores principales fueron:

123 ProductID	AZ ProductName	123 SalesPersonID	AZ Vendedor	123 total_vendido	123 cantidad_total	123 porcentaje_venta
179	Yoghurt Tubes	9	Daphne King	10.285	199.724	5,15
161	Longos - Chicken Wings	10	Jean Vang	10.785	199.659	5,4
47	Thyme - Lemon; Fresh	21	Devon Brewer	11.050	198.567	5,56
280	Onion Powder	21	Devon Brewer	10.570	198.163	5,33
103	Cream Of Tartar	11	Sonya Dickson	10.348	198.126	5,22

## Análisis

1. ¿Hay algún vendedor que aparece más de una vez como el que más vendió un producto?

Sí. El vendedor **Devon Brewer** (SalesPersonID 21) encabeza las ventas de **dos** de los cinco productos más vendidos (ProductID 47 y 280).

Esto lo convierte en el único vendedor que lidera las ventas de más de un producto dentro del top-5.

2. ¿Alguno de estos vendedores representa más del 10 % de las ventas de su producto?

No. Las participaciones oscilan entre **5,15 % y 5,56 %**, por lo que **ninguno supera el 10 %** del total vendido de su respectivo producto.

En consecuencia, las ventas de cada producto están relativamente distribuidas entre varios vendedores; ningún vendedor concentra una porción dominante.

## Conclusión

- El vendedor **Devon Brewer** demuestra un rendimiento sobresaliente al liderar dos productos distintos, aunque su participación individual no llega a ser dominante (>10 %).

- La baja concentración (5%) sugiere que las ventas de estos productos dependen de **un equipo de vendedores** más que de un único vendedor.

### 3.2 Clientes Únicos y Penetración

**Pregunta 1.b** – Entre los 5 productos más vendidos. Son los siguientes:

123 ProductID	A-Z ProductName	123 clientes_unicos	123 total	123 porcentaje_clientes
161	Longos - Chicken Wings	14.252	98.759	14,43
103	Cream Of Tartar	14.246	98.759	14,43
47	Thyme - Lemon; Fresh	14.101	98.759	14,28
179	Yoghurt Tubes	14.066	98.759	14,24
280	Onion Powder	14.058	98.759	14,23

### Análisis

#### 1. ¿Cuántos clientes únicos y qué proporción?

Cada uno de los cinco productos fue comprado por aproximadamente el 14 % de la base de clientes; en números absolutos, entre 14.058 y 14.252 compradores distintos.

#### 2. ¿Amplia adopción o concentración en pocos clientes?

- Un porcentaje cercano al 15 % no es marginal (<5 %), pero tampoco masivo (>25 %).
- Implica que **miles de clientes diferentes** participaron, por lo que el volumen de ventas **no depende de un puñado de “grandes compradores”**.
- En otras palabras: **adopción moderada y bastante equilibrada**.

#### 3. Comparación entre productos

- Las diferencias entre el que más penetra (14,43 %) y el que menos (14,23 %) son ínfimas (0,2 p.p.).
- Ningún producto sobresale como estrella ni queda rezagado; todos comparten un patrón de demanda muy parecido.

#### 4. Dependencia de un segmento específico

- Dado que la penetración es homogénea y moderada en la base total, **no se observa dependencia marcada de un nicho concreto**.
- Para confirmar segmentaciones más finas (ej. región, canal, tipo de cliente) habría que cruzar con atributos de cliente, pero con la métrica global no se evidencia un sesgo fuerte.

### 3.3 Categorías de los Productos Top-5

123 ProductID	A-Z ProductName	A-Z CategoryName	123 unidades_producto	123 total_categoria	123 porcentaje_categoria
179	Yoghurt Tubes	Seafood	199.724	6.996.142	2,85
161	Longos - Chicken Wings	Snails	199.659	7.199.358	2,77
280	Onion Powder	Beverages	198.163	7.393.693	2,68
47	Thyme - Lemon; Fresh	Poultry	198.567	9.159.792	2,17
103	Cream Of Tartar	Meat	198.126	9.719.274	2,04

- Cada uno de los cinco productos más vendidos proviene de **una categoría distinta** (*Seafood, Snails, Beverages, Poultry, Meat*).
- Por lo tanto, el ranking global de unidades no está concentrado en una sola línea de productos.

#### ¿Qué proporción representan dentro de su categoría?

- Ningún producto supera el **3 %** de las unidades totales de su categoría; los valores fluctúan entre el **2,04 % y 2,85 %**.
- Esto revela que cada categoría es **muy diversa** y que las ventas se reparten entre numerosos SKUs.

#### Comparación de relevancia mediante funciones de ventana

- Al dividir las unidades del producto por ese total se obtuvo *porcentaje\_categoria*, que mide su **peso relativo**.
- Aunque son los líderes de la empresa, **ninguno domina su propia categoría**, lo que sugiere que el catálogo interno es amplio y competitivo.

#### Conclusión

- Los top-5 destacan a nivel compañía, pero **no tiran de su categoría**: todavía hay margen para acciones que eleven su cuota al 4-5 % sin canibalizar otros artículos.
- También pueden utilizarse como **productos gancho** para promocionar referencias menos populares dentro de la misma línea, ya que atraen clientela sin ser hegemónicos.

### 3.4 Top-10 Productos Globales

¿Cuáles son los 10 productos con mayor cantidad de unidades vendidas en todo el catálogo y cuál es su posición dentro de su propia categoría?

123 ProductID	A-Z ProductName	A-Z CategoryName	123 unidades_vendidas	123 rank_categoria
179	Yoghurt Tubes	Seafood	199.724	1
161	Longos - Chicken Wings	Snails	199.659	1
47	Thyme - Lemon; Fresh	Poultry	198.567	1
280	Onion Powder	Beverages	198.163	1
103	Cream Of Tartar	Meat	198.126	1
324	Apricots - Dried	Snails	198.032	2
39	Dried Figs	Produce	198.032	1
319	Towels - Paper / Kraft	Meat	198.005	2
425	Wine - Redchard Merritt	Dairy	197.969	1
184	Hersey Shakes	Poultry	197.942	2

- **7 de los 10** artículos (70 %) son N° 1 en su línea; dominan tanto el ranking global como el interno.
- **3 productos** (Apricots – Dried, Towels – Paper/Kraft y Hersey Shakes) aparecen segundos; aunque figuran en el Top-10 absoluto, en su categoría hay otro SKU que vende ligeramente más.

#### Concentración de ventas

- **Seafood, Beverages, Produce y Dairy** tienen un único SKU en el Top-10 → la mayor parte de la visibilidad recae en un solo producto.
- **Snails, Meat y Poultry** colocan dos SKUs cada una → la demanda está **concentrada en un puñado de artículos muy parejos**, dejando el resto de la categoría en la larga cola.

#### Observaciones sobre concentración de las ventas.

- **Liderazgo global = liderazgo de nicho** para 70 % de los casos; sin embargo, tres categorías muestran **duelos muy ajustados** por el primer puesto.

- Las diferencias internas menores al 1 % sugieren riesgo de **canibalización** (clientes intercambian entre los dos SKUs líderes) y oportunidad de gestionar precio/promoción de forma coordinada.
- En categorías con solo un representante en el Top-10, el desempeño depende de **un único SKU**; conviene impulsar alternativas para reducir riesgo y diversificar ingresos.

## **Monitoreo Automático de Ventas – Trigger**

*Crea un trigger que registre en una tabla de monitoreo cada vez que un producto supere las 200.000 unidades vendidas acumuladas.*

*El trigger debe activarse después de insertar una nueva venta y registrar en la tabla el ID del producto, su nombre, la nueva cantidad total de unidades vendidas, y la fecha en que se superó el umbral.*

*Para cumplir con la consigna, se creó un **trigger llamado log\_productos** que se ejecuta después de cada inserción en la tabla **sales**. Su objetivo es **detectar automáticamente si un producto supera las 200.000 unidades vendidas acumuladas**.*

*El trigger calcula la suma total de unidades vendidas para el **ProductID** recién insertado. Si dicha cantidad supera el umbral y **aún no existe un registro previo en la tabla monitoreo\_productos**, se inserta un nuevo registro con:*

- ID del producto,
- Nombre,
- Total vendido acumulado,
- Fecha y hora en que se detectó el evento.

*Este mecanismo permite:*

- Tener **un registro histórico automático de productos con alto volumen de ventas**
- Evitar duplicados gracias a la condición **NOT EXISTS**,



- *Facilitar futuras estrategias de negocio como promociones, alertas o auditorías sobre productos destacados.*

monitoreo_productos X						
Propiedades Datos Diagrama						
monitoreo_productos Enter a SQL expression to filter results (use Ctrl+Space)						
Grilla	123 ID	123 productId	A-Z ProductName	123 TotalVendido	FechaRegistro	
1	1	103	Cream Of Tartar	498.126	2025-06-03 17:14:43	

## Optimización de Consultas

*Selecciona dos consultas del avance 1 y crea los índices que consideres más adecuados para optimizar su ejecución.*

*Prueba con índices individuales y compuestos, según la lógica de cada consulta. Luego, vuelve a ejecutar ambas consultas y compara los tiempos de ejecución antes y después de aplicar los índices. Finalmente, describe brevemente el impacto que tuvieron los índices en el rendimiento y en qué tipo de columnas resultan más efectivos para este tipo de operaciones.*

*Para optimizar la consulta del ejercicio 1A (ventas agrupadas por producto y vendedor), se creó un índice compuesto que incluye:*

*CREATE INDEX idx\_sales\_product\_vendedor\_qty ON sales(ProductID, SalesPersonID, Quantity);*

### Justificación técnica:

- *ProductID y SalesPersonID se usan en el GROUP BY.*
- *Quantity se utiliza en la función de agregación SUM, por lo que incluirla permite que el motor acceda al dato directamente desde el índice (index covering), sin leer la tabla completa.*

### Impacto en el rendimiento:

- *Antes del índice: 9.751 segundos*
- *Después del índice: 5.6 segundos*

- **Mejora:** ~43%

Esto demuestra que los índices compuestos no solo aceleran el agrupamiento y filtrado, sino que también pueden reducir el acceso a disco al cubrir todas las columnas necesarias en una consulta.

## **Optimización 2: Índice sobre *sales(ProductID, CustomerID)***

Para la consulta del **Ejercicio 1B**, que calcula la cantidad de **clientes únicos por producto**, se creó el siguiente índice compuesto:

***CREATE INDEX idx\_sales\_product\_cliente ON sales(ProductID, CustomerID);***

### **Justificación técnica:**

- *ProductID* se usa en el **WHERE** y **GROUP BY**.
- *CustomerID* se utiliza en un **COUNT(DISTINCT CustomerID)**.
- Este índice facilita el agrupamiento por producto y la identificación eficiente de clientes únicos, al reducir el escaneo de filas redundantes.

### **Impacto en el rendimiento:**

- **Antes del índice:** 10.0 segundos
- **Después del índice:** 5.7 segundos
- **Mejora:** ~43%

Esto evidencia cómo los índices compuestos diseñados sobre columnas clave de filtrado y agregación mejoran drásticamente el tiempo de ejecución de reportes analíticos.

## **Feature Engineering y Limpieza de Datos**

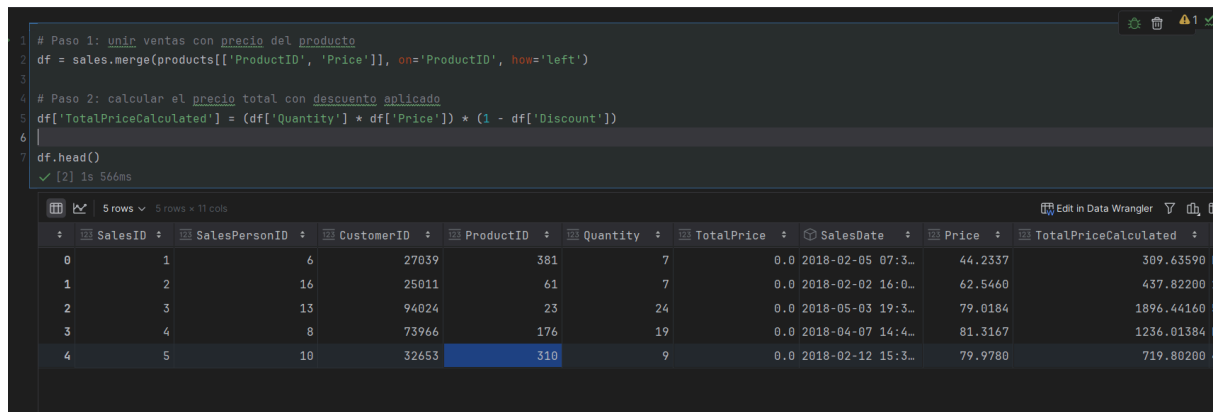
### **6.1 Cálculo de *TotalPriceCalculated***

El campo *TotalPrice* en la tabla *sales* no tiene valores válidos. Utilizando la información de precios de la tabla *products*, calcula el valor real de la venta para cada registro y almacena en una nueva columna.

Utilizo la siguiente fórmula:

$$\text{TotalPriceCalculated} = (\text{Quantity} \times \text{UnitPrice}) \times (1 - \text{Discount})$$

Se creó una nueva columna llamada *TotalPriceCalculated* que refleja el resultado de este cálculo. En la siguiente imagen se muestran los primeros registros con el cálculo ya aplicado:



The screenshot shows a Jupyter Notebook interface. The top part contains two code cells. The first cell merges a 'sales' DataFrame with a 'products' DataFrame on 'ProductID'. The second cell calculates 'TotalPriceCalculated' based on 'Quantity', 'Price', and 'Discount'. Below the code, the 'df.head()' output is displayed as a table with 11 columns: SalesID, SalesPersonID, CustomerID, ProductID, Quantity, TotalPrice, SalesDate, Price, and TotalPriceCalculated. The table shows 5 rows of data.

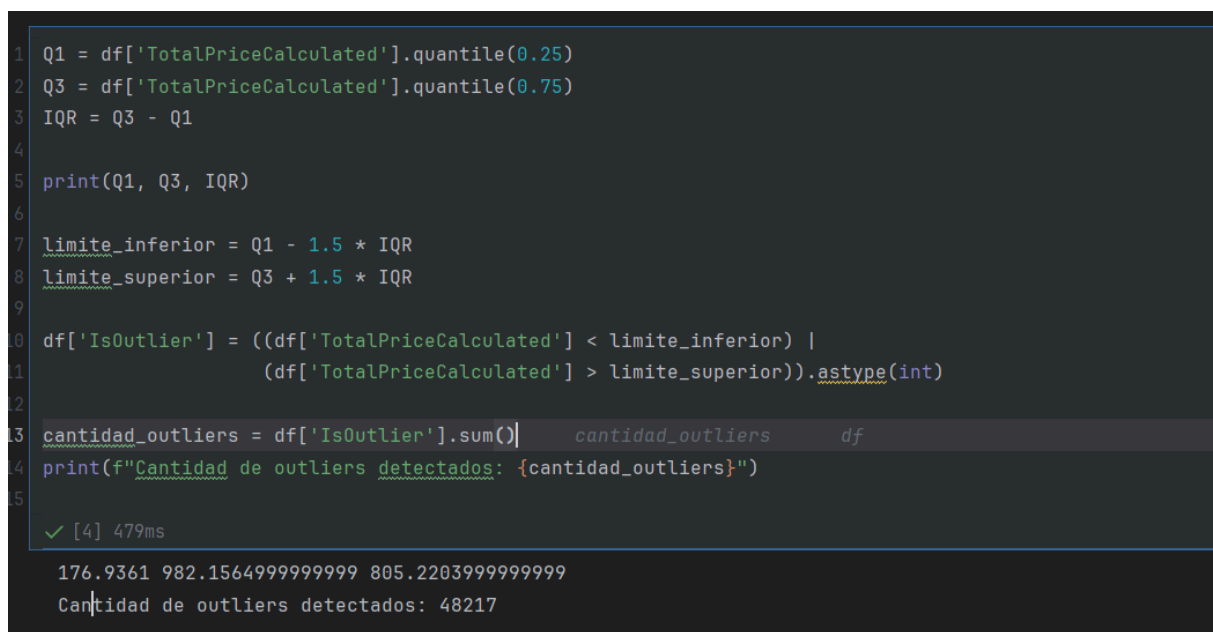
SalesID	SalesPersonID	CustomerID	ProductID	Quantity	TotalPrice	SalesDate	Price	TotalPriceCalculated
0	1	6	27039	381	7	0.0 2018-02-05 07:3...	44.2337	309.63590
1	2	16	25011	61	7	0.0 2018-02-02 16:0...	62.5460	437.82200
2	3	13	94024	23	24	0.0 2018-05-03 19:3...	79.0184	1896.44160
3	4	8	73966	176	19	0.0 2018-04-07 14:4...	81.3167	1236.01384
4	5	10	32653	310	9	0.0 2018-02-12 15:3...	79.9780	719.80200

Como se observa, ahora cada fila tiene un valor coherente en base a la cantidad vendida, el precio del producto y el descuento aplicado.

## 6.2 Detección de Outliers

Utilizando el criterio del rango intercuartílico (IQR). Luego, crea una nueva columna llamada *IsOutlier* que tenga el valor 1 si el registro es un outlier y 0 en caso contrario.

**¿Cuántos outliers se detectaron?**



The screenshot shows a Jupyter Notebook with a single code cell. The code calculates the IQR for 'TotalPriceCalculated', determines lower and upper limits (1.5 \* IQR away from Q1 and Q3), and creates an 'IsOutlier' column. It then prints the count of outliers. The output shows 48217 outliers.

```
1 Q1 = df['TotalPriceCalculated'].quantile(0.25)
2 Q3 = df['TotalPriceCalculated'].quantile(0.75)
3 IQR = Q3 - Q1
4
5 print(Q1, Q3, IQR)
6
7 limite_inferior = Q1 - 1.5 * IQR
8 limite_superior = Q3 + 1.5 * IQR
9
10 df['IsOutlier'] = ((df['TotalPriceCalculated'] < limite_inferior) |
11                  (df['TotalPriceCalculated'] > limite_superior)).astype(int)
12
13 cantidad_outliers = df['IsOutlier'].sum()
14 print(f"Cantidad de outliers detectados: {cantidad_outliers}")
15
```

✓ [4] 479ms

176.9361 982.1564999999999 805.2203999999999  
Cantidad de outliers detectados: 48217

```

cantidad_outliers = df['IsOutlier'].sum()
print(f"Cantidad de outliers detectados: {cantidad_outliers}")

df.head()

```

✓ [5] 431ms

176.9361 982.1564999999999 805.2203999999999  
 Cantidad de outliers detectados: 48217

Quantity	Discount	TotalPrice	SalesDate	TransactionNumber	Price	TotalPriceCalculated	IsOutlier
7	0.0	0.0	2018-02-05 07:38:25.430	FQL4S94E4ME1EZFTG42G	44.2337	309.63590	0
7	0.0	0.0	2018-02-02 16:03:31.150	12UGLX40DJ1A5DTFBH88	62.5460	437.82200	0
24	0.0	0.0	2018-05-03 19:31:56.880	5DT8RCPL87KI5E0R07B0	79.0184	1896.44160	0
19	0.2	0.0	2018-04-07 14:43:55.420	R3DR9MLD5NR76V017ULE	81.3167	1236.01384	0
9	0.0	0.0	2018-02-12 15:37:03.940	4BGS0Z5QMAZ8NDAFHHP3	79.9780	719.80200	0

Para este ejercicio, se analizaron los valores atípicos (outliers) en la columna **TotalPriceCalculated**, correspondiente al valor real de cada venta.

Se utilizó el criterio del rango intercuartílico (IQR) para identificar aquellos valores que se alejan significativamente del comportamiento normal de los datos.

Los pasos realizados fueron:

### 1. Cálculo del IQR:

- Q1 (primer cuartil): 176.9361
- Q3 (tercer cuartil): 982.1565
- $IQR = Q3 - Q1 = 805.2204$

### 2. Límites para detectar outliers:

- Límite inferior =  $Q1 - 1.5 \times IQR = -1030.8945$
- Límite superior =  $Q3 + 1.5 \times IQR = 2189.9871$

3. Todo valor fuera de ese rango se consideró un outlier.

4. Creación de la columna **IsOutlier**:

Se generó una nueva columna binaria que toma el valor **1** si el registro es un outlier y **0** en caso contrario.

5. Resultado:

Se detectaron 48.217 registros considerados outliers según este criterio.

## 6.3 Variables Temporales

A partir de la columna SalesDate, crea una nueva columna que contenga únicamente la hora de la venta.

Luego, identifica en qué hora del día se concentran más ventas totales (TotalPriceCalculated).

**¿La empresa vende más durante los días de semana o en el fin de semana?** Utiliza la columna SalesDate para identificar el día de la semana de cada venta, clasifica los registros como Entre semana o Fin de semana, y compara el total de ventas (TotalPriceCalculated) entre ambos grupos.

```
> 1 # Ensure that SalesDate is in datetime format
2 df['SalesDate'] = pd.to_datetime(df['SalesDate'])
3
4 # Create a new column with the hour (excluding date)
5 df['SaleHour'] = df['SalesDate'].dt.hour
6
7 # Group by hour and sum TotalPriceCalculated
8 sales_by_hour = df.groupby('SaleHour')['TotalPriceCalculated'].sum().reset_index()
9
10 # Get the hour with the highest sales total
11 peak_hour_sales = sales_by_hour.loc[sales_by_hour['TotalPriceCalculated'].idxmax()]
12 print(peak_hour_sales)
13
14 # Extract day of the week (0 = Monday, 6 = Sunday)
15 df['DayOfWeek'] = df['SalesDate'].dt.dayofweek
16
17 # Classify as Weekday or Weekend
18 df['DayType'] = df['DayOfWeek'].apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')
19
20 # Sum total sales by day type
21 sales_by_daytype = df.groupby('DayType')['TotalPriceCalculated'].sum().reset_index()
22 print(sales_by_daytype)
```

✓ [6] 7s 360ms

```
SaleHour      1.600000e+01
TotalPriceCalculated  1.790144e+08
Name: 16, dtype: float64

   TipoDia  TotalPriceCalculated
0  Entre semana      3.123405e+09
1  Fin de semana      1.192863e+09
```

Se analizó el comportamiento de las ventas según el momento del día y el tipo de día (semana vs. fin de semana).

1. **Extracción de la hora de venta:**

A partir de la columna *SalesDate*, se creó una nueva columna llamada *SaleHour* que contiene solamente la **hora en la que se realizó cada venta**.

2. **Hora de mayor volumen de ventas:**

Se agruparon los registros por hora y se sumaron los valores de la columna *TotalPriceCalculated*.

La **hora del día con mayor monto total de ventas fue las 16:00 hs**, con un total acumulado de **179.014.400 unidades monetarias**.

3. **Clasificación por tipo de día:**

Se creó una columna *TipoDia* para clasificar los registros como:

- **Entre semana** (de lunes a viernes)
- **Fin de semana** (sábado y domingo)

4. **Comparación de ventas según el tipo de día:**

<b>Tipo de Día</b>	<b>Total de Ventas (TotalPriceCalculated)</b>
Entre semana	3.123.405.000
Fin de semana	1.192.863.000

5. Como se observa, **la mayor parte de las ventas se concentran entre semana**, con más del doble de ingresos comparado con el fin de semana.

## 6.4 Edad y Experiencia del Empleado

Como parte del proceso de *feature engineering*, en el mismo *df* que vienes trabajando, calcula dos nuevas columnas en el dataset de ventas:

La edad del empleado al momento de su contratación y años de experiencia al momento de realizar cada venta.

Utiliza las columnas *BirthDate*, *HireDate* (de la tabla *employees*) y *SalesDate* (de la tabla *sales*). Asegúrate de trabajar con fechas en formato adecuado.

Como parte del proceso de feature engineering, se incorporaron dos nuevas columnas al dataset de ventas, utilizando las columnas *BirthDate* y *HireDate* de la tabla *employees*, y *SalesDate* de la tabla *sales*.

```
1 # Me aseguro el formato datetime
2 employees['BirthDate'] = pd.to_datetime(employees['BirthDate'])
3 employees['HireDate'] = pd.to_datetime(employees['HireDate'])
4 df['SalesDate'] = pd.to_datetime(df['SalesDate']) # por las dudas
5
6 # Merge usando los nombres correctos
7 df = df.merge(
8     employees[['EmployeeID', 'BirthDate', 'HireDate']],
9     left_on='SalesPersonID',
10    right_on='EmployeeID',
11    how='left'
12 )
13
14 # Calcular edad al momento de contratación
15 df['AgeAtHire'] = (df['HireDate'] - df['BirthDate']).dt.days
16
17 # Calcular años de experiencia al momento de la venta
18 df['YearsExperience'] = (df['SalesDate'] - df['HireDate']).dt.days
```

Los pasos realizados fueron:

### 1. Conversión de fechas:

Se aseguraron los formatos de las fechas utilizando la función `pd.to_datetime()` para poder operar con diferencias temporales.

### 2. Unión de tablas:

Se realizó un *merge* entre el dataset de ventas (*df*) y la tabla de empleados (*employees*), utilizando *SalesPersonID* (de *df*) y *EmployeeID* (de *employees*) como claves.

3. **Cálculo de edad al momento de la contratación (*AgeAtHire*):**

Se obtuvo restando la fecha de nacimiento (*BirthDate*) de la fecha de contratación (*HireDate*), y dividiendo los días entre 365 para obtener los años completos.

4. **Cálculo de años de experiencia al momento de cada venta (*YearsExperience*):**

Se calculó como la diferencia entre la fecha de la venta (*SalesDate*) y la fecha de contratación (*HireDate*), nuevamente expresado en años completos.

Estas nuevas columnas enriquecen el análisis permitiendo estudiar, por ejemplo, si los empleados más experimentados generan mayores ventas, o si la edad al momento de ingreso tiene algún impacto.

## ***Justificación de las transformaciones realizadas***

Durante el proceso de feature engineering, se aplicaron distintas transformaciones con el objetivo de enriquecer el análisis sin modificar la variable objetivo, que en este caso es *TotalPriceCalculated*.

Las transformaciones realizadas incluyen:

- **Conversión de columnas de fecha (*SalesDate*, *BirthDate*, *HireDate*) al formato *datetime*:**

Esta transformación fue necesaria para poder calcular diferencias de tiempo, como edad o experiencia, y para extraer componentes útiles como la hora o el día de la semana.

- **Cálculo de variables derivadas (*AgeAtHire*, *YearsExperience*, *SaleHour*, *DayType*):**

Estas nuevas columnas aportan información adicional que puede ser útil para entender los patrones detrás de las ventas, sin alterar la variable objetivo.

- **Clasificación binaria de outliers (*IsOutlier*):**

Esta transformación ayuda a detectar valores extremos, pero no modifica los datos originales ni afecta el valor de *TotalPriceCalculated*.



En todos los casos, la variable objetivo **TotalPriceCalculated** se mantuvo sin modificaciones, tal como exige el enunciado, para preservar su integridad y asegurar un análisis confiable y consistente.

## **Parte 4 – Diseño orientado a objetos y análisis dinámico de ventas**

### **Objetivo**

Diseñar una solución flexible y extensible que permita realizar distintos tipos de análisis sobre datos de ventas, encapsulando la lógica en clases reutilizables, aplicando patrones de diseño como **Strategy** y **Factory Method**, y midiendo el rendimiento de cada enfoque con herramientas de **profiling**.

### **Estructura implementada**

#### **1. Interfaz común:**

Se definió la clase abstracta **SalesAnalysisStrategy** con un método obligatorio **analyze(df)** que todas las estrategias deben implementar.

#### **2. Estrategias concretas:**

Se desarrollaron las siguientes clases que heredan de la interfaz:

- **Top5ConsecutiveDaysBrute**: usa un bucle **for** para encontrar los 5 días consecutivos con mayor venta (fuerza bruta).
- **Top5ConsecutiveDaysVectorized**: usa **pandas.rolling()** para hacer el mismo análisis de forma optimizada.
- **MaxSingleDayStrategy**: analiza cuál fue el día con mayor venta total (ejemplo de otro tipo de análisis).

### 3. **Factory Method:**

Se creó la clase `AnalysisFactory` con un método `get_strategy(nombre)` que devuelve la estrategia solicitada sin necesidad de modificar el código central.

### 4. **Extensibilidad:**

Para agregar un nuevo análisis, basta con:

- Crear una nueva clase que implemente `analyze(df)`
- Registrarla en el diccionario interno de `AnalysisFactory`

5. No es necesario modificar las estrategias existentes.

### 6. **Perfilamiento:**

Se implementó la función `profile_strategy()` que mide el tiempo promedio de ejecución de cada estrategia usando `timeit`.

## **Comparación de estrategias**

Se aplicaron dos enfoques para identificar el período de 5 días consecutivos con mayores ventas:

- **Fuerza bruta:** recorre cada ventana de 5 días con un bucle `for`.
- **Optimizado (vectorizado):** utiliza `pandas.rolling()` para calcular la suma móvil.

Ambos métodos encontraron el mismo resultado:


El período con mayores ventas fue del **28/03/2018 al 01/04/2018**, con un total de **166,6 millones**.

### ***Rendimiento comparado:***

<b><i>Estrategia</i></b>	<b><i>Tiempo promedio</i></b>
<i>top5_brute</i>	<i>2.60 segundos</i>
<i>top5_vectorized</i>	<i>2.75 segundos</i>

*En este caso, la estrategia vectorizada **no fue más rápida**, posiblemente por la distribución de fechas o por overhead de operaciones internas en Pandas. Sin embargo, en datasets más grandes o con fechas agrupadas, suele mostrar mejoras significativas.*

### ***Repositorio del proyecto***

 *Podés revisar el código completo y la documentación del proyecto en GitHub:*

 **Enlace directo:** <https://github.com/carlosmdq44/integradorHenry>