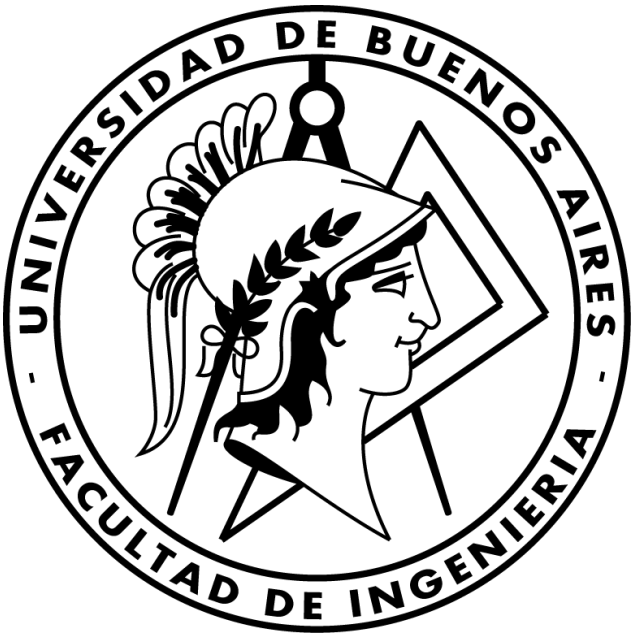


(TA047) - Ciencia de Datos

Trabajo Práctico 1 (2C2025)

Patricio Ibar - Padrón 109569

11 de septiembre de 2025

Índice

1. Introducción

En el presente informe se documentan los pasos seguidos para la realización del Trabajo Práctico 1 de la materia Ciencia de Datos (TA047) de la Facultad de Ingeniería de la Universidad de Buenos Aires. El objetivo del trabajo práctico es realizar un análisis exploratorio de conjunto de datos. El análisis incluye la limpieza y preparación de los datos, la realización de consultas y visualizaciones, y la interpretación de los resultados obtenidos.

Se trabajó utilizando Python y la librería pandas para la manipulación y análisis de datos. Además, se utilizaron otras librerías como matplotlib y seaborn para la visualización de datos.

Tanto el código fuente, junto con las visualizaciones, como el informe se encuentran disponibles en el siguiente repositorio de GitHub: `https://github.com/patricioibar/datos-tp1`

2. Limpieza de Datos

Luego de cargar los datos y previo a la realización de análisis exploratorios, es fundamental llevar a cabo un proceso de limpieza y preparación de los datos. En esta sección se describen los pasos realizados para la limpieza y preparación de los datos antes de su análisis. Se detallan las técnicas utilizadas para manejar valores faltantes, reducción del uso de memoria y la transformación de variables.

2.1. Carga de Datos

Los datos fueron cargados desde un archivo CSV utilizando la librería pandas de Python. Al cargar una tabla, se especificaron las columnas a utilizar para reducir el uso de memoria. También se especificaron los tipos de datos para cada columna para asegurar una correcta interpretación. Especificar el tipo de datos también optimiza el uso de memoria, ya que pandas asigna el tipo de datos más eficiente posible.

Los tipos de datos utilizados fueron:

- `uint32` para identificadores y cantidades.
- `float32` para precios y totales.
- `category` para columnas con un número limitado de valores únicos, como estados o método de pago.
- `datetime64` para fechas y horas.
- `string` para texto sin formato.
- `bool` para valores booleanos.

Todas estas acciones se realizan a través de la función `pd.read_csv()` de pandas, que permite especificar las columnas a cargar y sus tipos de datos. A continuación se muestra un ejemplo de cómo se realiza la carga de datos:

```python
# Carga de la tabla de ordenes para las consultas propuestas por el enunciado
orders = pd.read_csv(
    'data/orders.csv',
    usecols=[
        'order_id',
        'customer_id',
        'status',
        'payment_method',
        'billing_address',
        'discount_amount'
    ],
    dtype={
        'order_id': 'uint32',
        'customer_id': 'uint32',
        'status': 'category',
        'discount_amount': 'float32',
        'payment_method': 'category'
    },
)
```

2.2. Manejo de Valores Faltantes

Es común encontrar valores faltantes en los conjuntos de datos. Estos valores pueden ser el resultado de errores en la recolección de datos o simplemente porque la información no estaba disponible en el momento de la recolección. Por ejemplo, para las primeras consultas del enunciado, se identificaron valores faltantes en las columnas `status` y `billing_address`. La presencia de valores faltantes puede ocasionar problemas al ejecutar ciertas operaciones o análisis, por lo que es crucial abordarlos adecuadamente. Además, es importante manejar los valores faltantes de una forma acorde al análisis que se desea realizar. En algunos casos puede ser necesario eliminar filas o columnas con valores faltantes, mientras que en otros casos puede ser más apropiado darle un valor específico.

Por ejemplo, tanto en las columnas `status` como en `billing_address`, se decidió dar un valor por defecto a los valores faltantes. Para esto se utilizó la función `fillna()` de pandas, que permite reemplazar los valores faltantes con un valor específico. En este caso, se decidió reemplazar los valores faltantes con el valor `ÜNDEFINED"`.

En otros casos, por lo menos en este dataset, algunos valores faltantes pueden ser inferidos a partir de valores en otras columnas. Por ejemplo, en la tabla de ítems de orden, las columnas `quantity`, `unit_price` y `line_total` están relacionadas por la fórmula:

$$\texttt{line\_total} = \texttt{quantity} \times \texttt{unit\_price}$$

Si uno de estos valores falta, puede ser calculado a partir de los otros dos.

2.3. Normalización de Datos

En muchos casos pueden encontrarse diferentes valores en una misma columna que representan la misma información. Por ejemplo, en la columna `status` pueden encontrarse los valores `"delivered"`, `"DELIVERED"` y `"Delivered"`, que representan el mismo estado de una orden. Para evitar confusiones y facilitar el análisis, es importante normalizar estos valores a un formato consistente.

La estrategia que adopté fue convertir todos los valores a mayúsculas utilizando la función `Series.str.upper()` de pandas, además de quitar los posibles espacios que el valor pueda tener al comienzo y al final, utilizando la función `Series.str.strip()`.

Luego de realizar estas 'normalizaciones', si la columna es categórica, es importante reconvertirla al tipo de dato `category` para optimizar el uso de memoria.

3. Características del Dataset

El dataset provisto por la cátedra contiene información sobre órdenes de compra, clientes, categorías, productos y reseñas de lo que podría ser una tienda en línea.
Al hacer varias consultas y visualizar los resultados se puede deducir que los datos no son reales, sino que son simulados. Por ejemplo, los nombres y apellidos de los clientes son muy genéricos y repetitivos, los nombres de los productos parecen ser palabras aleatorias sin significado, las ciudades tienen nombres inventados, las direcciones tienen estados y códigos postales de Estados Unidos pero países diferentes, etc. También aparecen valores sin sentido para algunos motivos de movimiento de inventario, como por ejemplo 'Theft' o 'Damage', pueden aparecer con valores positivos o negativos, lo carece de sentido real.

Además de esto, al realizar las consultas se encuentra con que los resultados son considerablemente uniformes, lo que refuerza la idea de que los datos son simulados. Esto es una lástima, pues dificulta la realización de análisis más profundos y la obtención de conclusiones interesantes.

A continuación muestro algunos gráficos que ilustran estas características simuladas o uniformes para las tablas `orders`, `customers` e `inventory_logs`. Todas las consultas y gráficos de esta sección pueden encontrarse en el notebook `datos_uniformes.ipynb`.

3.1. Usuarios

Para empezar, podemos ver en la figura 1 la cantidad de usuarios por país. Se puede observar que en todos los paises hay una cantidad muy similar de usuarios, y aún más, hay una cantidad también similar de usuarios sin país determinado. Esto es poco realista, ya que en la vida real la distribución de usuarios por país suele ser muy desigual debido a cuestiones socioeconómicas o poblacionales.

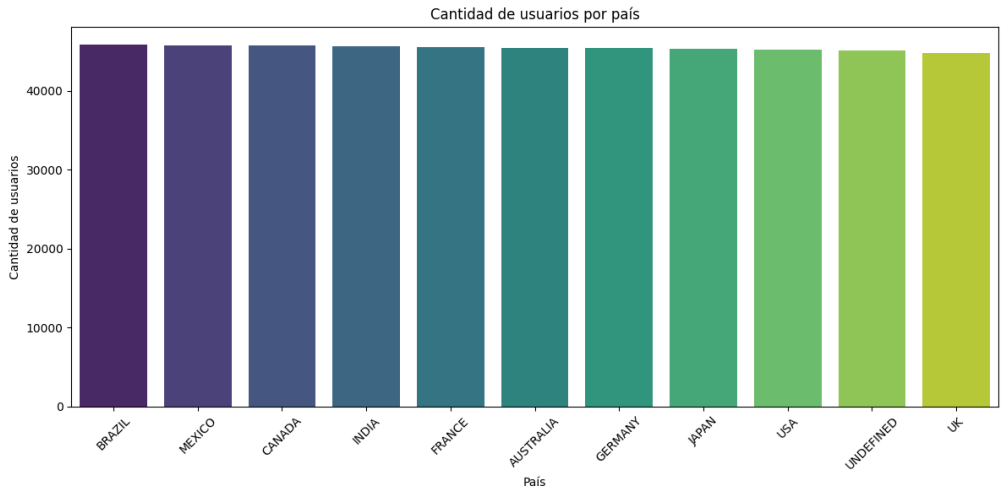


Figura 1: Cantidad de usuarios por país

En las siguientes dos figuras, podemos ver ilustrados los nuevos usuarios que se registran cada mes, diferenciados según el segmento al que pertenecen. En estos gráficos se puede observar que, si bien no hay

una cantidad exactamente igual de usuarios nuevos cada mes, la diferencia es muy pequeña. La mayor diferencia de nuevos usuarios está en el primer mes y el último mes registrado (ver figura 2), sin embargo la proporción de nuevos usuarios respecto al total de usuarios sigue siendo muy similar en todos los meses (ver figura 3).

Otra cosa que se puede observar en estos gráficos es que la mayoría de los usuarios pertenecen al segmento `Regular`, mientras que los segmentos `Premium` y `Budget` son tienen menor volumen. Aún así, es destacable la proporción casi idéntica en estos últimos dos segmentos.

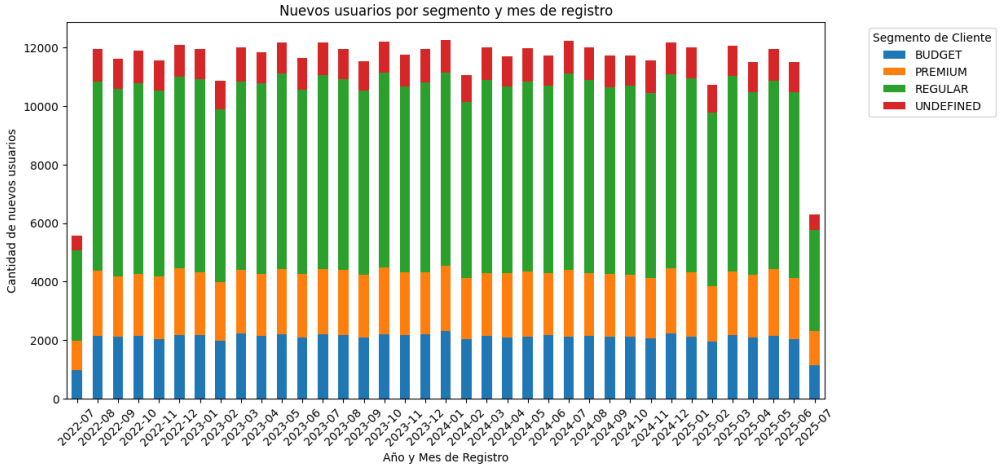


Figura 2: Cantidad de usuarios nuevos por segmento y mes de registro

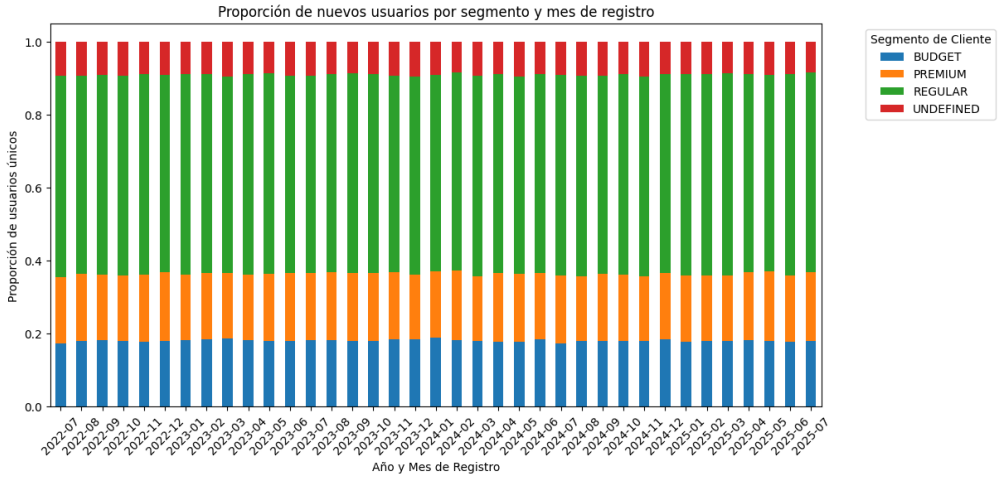


Figura 3: Proporción de nuevos usuarios por segmento y mes de registro

A continuación se muestran dos gráficos que ilustran la cantidad de usuarios por estado. Estos gráficos con mapas los generé utilizando la librería `geopandas`, y los mapas los descargué del sitio web naturalearthdata

En la figura 4 se puede observar que la mayoría de los estados tienen una cantidad similar de usuarios. En esta primera figura se visualiza una escala de 0 a 7000, y vemos que todos los estados tienen colores muy similares.

En la figura 5 se puede ver el mismo mapa con una escala más ajustada, aproximadamente de 6300 a

6700. En este gráfico sí se pueden notar colores más variados. Aún así hay que notar que la diferencia entre el estado con más usuarios y el estado con menos usuarios es de a lo sumo 400 usuarios, lo cual es una diferencia bastante baja tomando en consideración el total de usuarios por estado. Esto refuerza la idea de que los datos son simulados y no representan una distribución realista de usuarios por estado. Notar como, por ejemplo, el estado de Alaska tiene muchos más usuarios que Texas, lo cual es muy improbable en un escenario real.

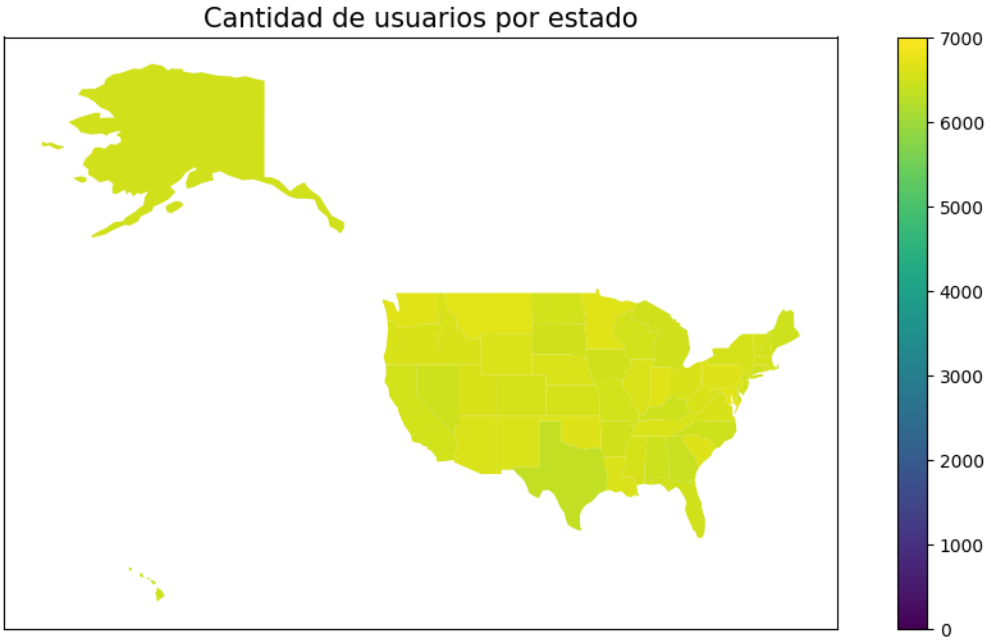


Figura 4: Cantidad de usuarios por estado

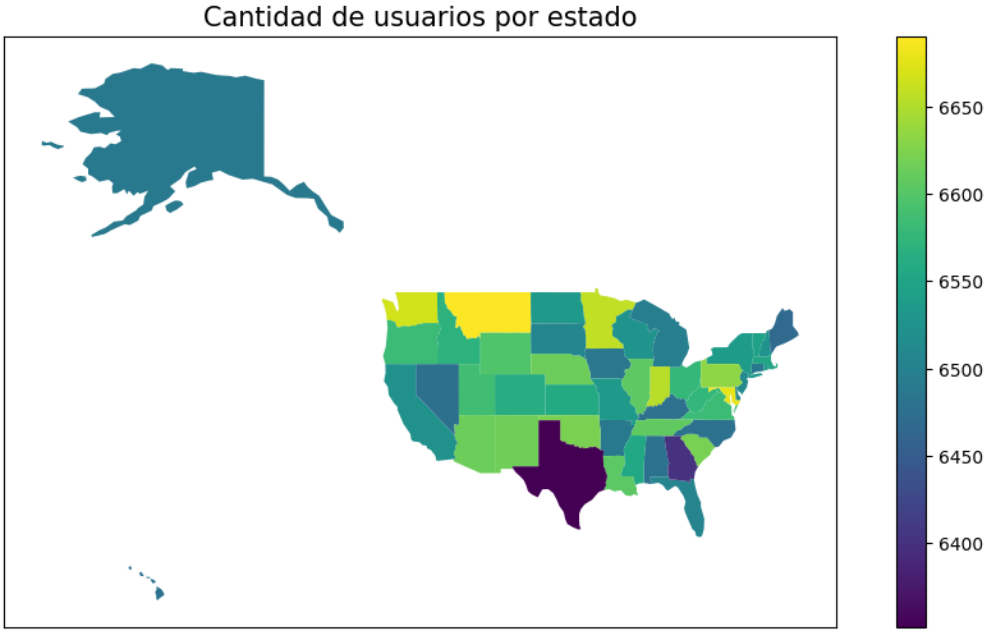


Figura 5: Cantidad de usuarios por estado (detalle)

3.2. Órdenes

En la figura 6 se pueden visualizar la cantidad de órdenes totales por estado. Nuevamente, se puede observar que la mayoría de los estados tienen una cantidad similar de órdenes, a excepción de los estados militares de Estados Unidos (AA, AE, AP) que tienen una cantidad considerablemente mayor de órdenes, casi exactamente el doble.

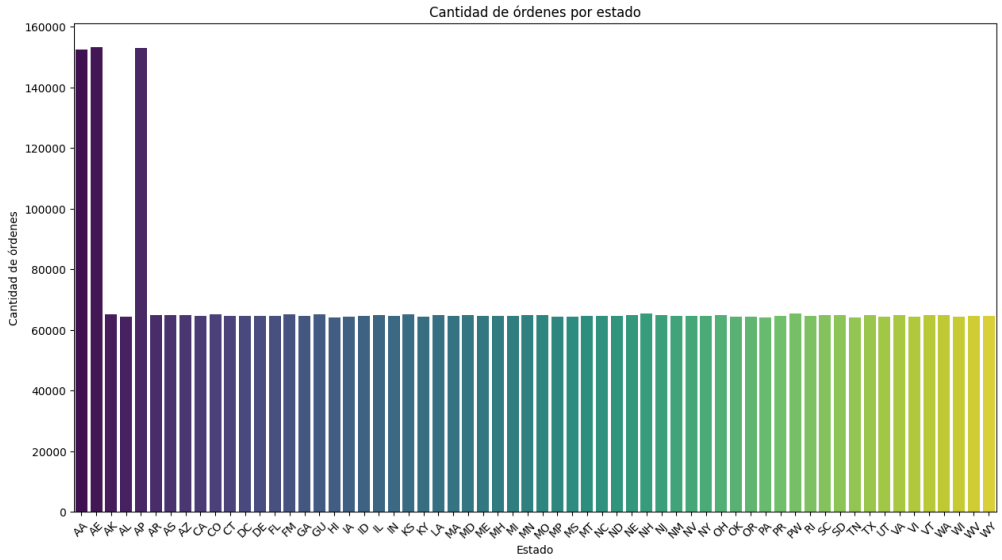


Figura 6: Cantidad de órdenes por estado

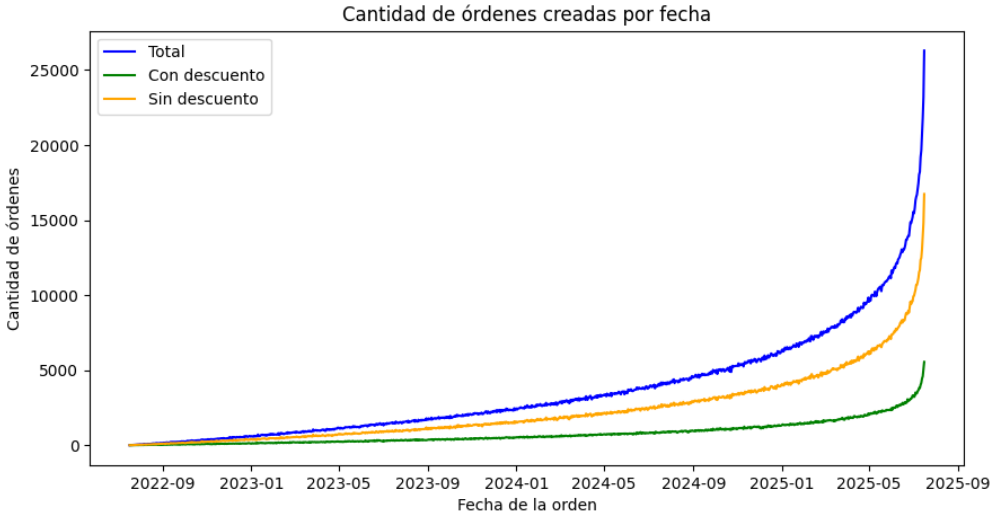


Figura 7: Cantidad de órdenes por fecha (agrupados con descuento, sin descuento y totales)

En la figura 7 se puede observar la cantidad de órdenes por fecha (i.e. la cantidad de filas en la tabla `orders` por fecha), diferenciados para órdenes con descuento, sin descuento y el total. Se puede observar que las tres curvas siguen una tendencia muy marcada, parecida a una curva exponencial, con desviaciones muy pequeñas.
Esto nos da una idea de cómo se pueden haber simulado los datos, podemos inferir que se utilizó algún tipo

de función matemática para generar estas cifras y que se utilizó un porcentaje fijo de órdenes con descuento respecto al total de órdenes.

En la figura 8 se puede observar la proporción de órdenes por estado en el tiempo. Cada color representa uno de los estados. Se puede observar nuevamente la tendencia muy marcada en todos los estados, con desviaciones muy pequeñas. Se puede apreciar muy bien cómo la cantidad de órdenes por mes es casi idéntica para todos los estados. La única excepción son las cantidades en los estados militares como se vió en la figura 6, que aún puede verse que tienen que siguen una tendencia similar a la de los demás estados.

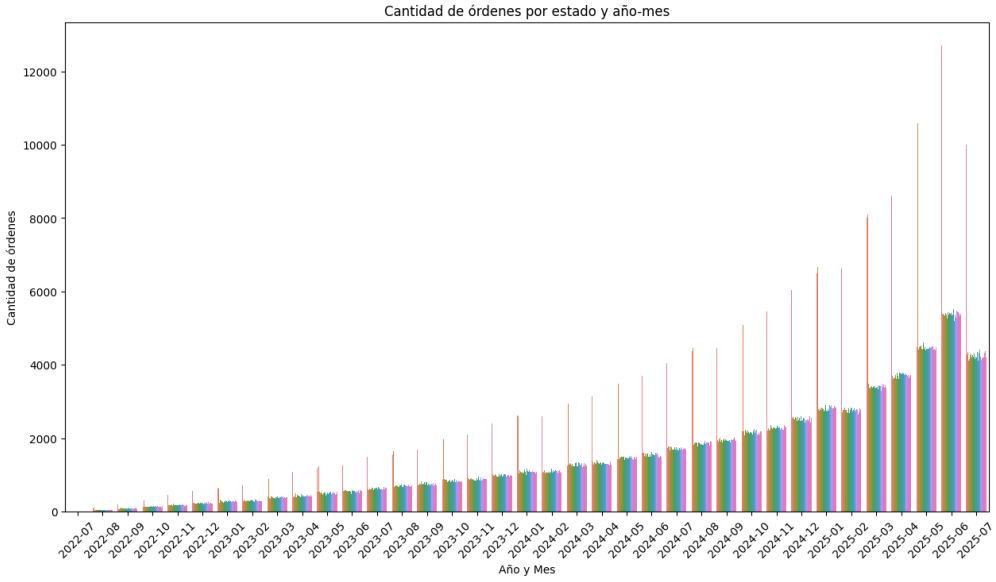


Figura 8: Proporción de órdenes por estado en el tiempo

La única anomalía en la tendencia que siguen los datos es en el último mes registrado. Esto puede deberse a que los datos fueron generados hasta una fecha específica (17/07/2025) y no se completó el mes, por lo que la cantidad de órdenes en ese mes es considerablemente menor.

3.3. Inventario

En la figura 9 se puede observar la cantidad de movimientos de inventario por tipo y día. Se puede observar que a lo largo del tiempo se registran cantidades de movimientos similares para cada tipo de movimiento. La única excepción es el tipo de movimiento 'Undefined', que tiene una cantidad considerablemente menor de movimientos registrados.

Notar cómo se forman franjas en las cuales se distribuyen los puntos, donde cada uno representa la cantidad de movimientos que hubo en un día. Si los datos fueran reales, se podría esperar ver que las franjas sigan un ciclo anual, con más movimientos en ciertas épocas del año (por ejemplo, en diciembre por las fiestas) y menos en otras. Sin embargo, en este caso las franjas no siguen un patrón anual, sino que parecen estar distribuidas de manera uniforme a lo largo del tiempo.

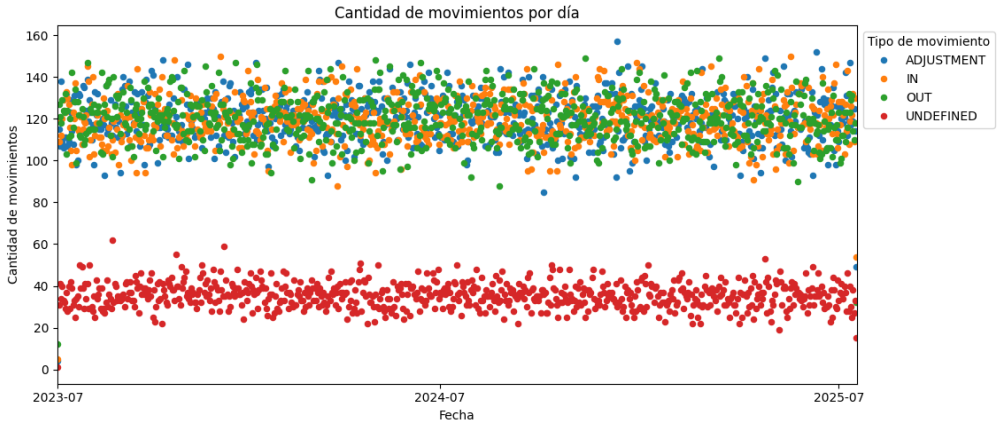


Figura 9: Cantidad de movimientos por tipo y día

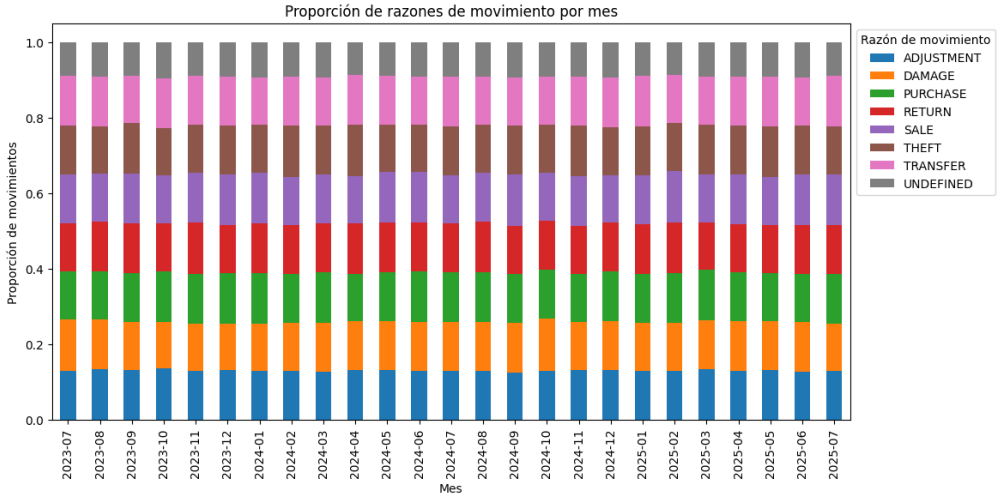


Figura 10: Proporción de razones de movimientos por mes

En la figura 10 nuevamente se puede observar una distribución uniforme en el tiempo, esta vez de las razones de movimientos de inventario. Se puede observar que hay una cantidad muy similar de todos los movimientos, y que la proporción entre ellos se mantiene casi constante a lo largo del tiempo. Por ejemplo podemos ver que la proporción entre movimientos de tipo 'Sale' y 'Theft' es casi exactamente 1:1 en todos los meses, lo cual sería preocupante en un escenario real.

Finalmente, se presentan dos gráficos que personalmente me gustan mucho.
En la figura 11 se puede observar el cambio acumulado en la cantidad de unidades del inventario (sin diferenciar entre productos). Es notable cómo la línea negra, que representa el cambio total en la cantidad, se mantiene casi constante a lo largo del tiempo, con pequeñas variaciones. Esto indica que la cantidad de unidades en el inventario no varía mucho a lo largo del tiempo. Se puede observar que en todos los meses las cantidades en los movimientos 'IN' se compensan con las de los movimientos 'OUT', mientras que las cantidades de los movimientos 'ADJUSTMENT' y 'UNDEFINED' casi no aparecen en el gráfico, lo que indica que las entradas en la tabla se cancelan entre sí en un mismo mes.

Por último, en la figura 12 se puede observar la distribución de los tipos de movimiento en el tiempo. Podemos observar que, aún con una elevada cantidad de `bins` en el histograma, la distribución de los tipos

de movimiento es prácticamente constante para los tipos de movimiento ‘IN’, ‘OUT’ y ‘ADJUSTMENT’. Además, podemos ver límites muy marcados y redondos en los valores que toma cada tipo de movimiento. También se puede observar que el tipo de movimiento ‘UNDEFINED’ no sigue una distribución uniforme, pero sigue la misma distribución que el total de movimientos. De esta manera podemos inferir que para el dataset se crearon los datos ‘IN’, ‘OUT’ y ‘ADJUSTMENT’ de manera uniforme, y luego los datos ‘UNDEFINED’ se crearon como un porcentaje fijo del total de movimientos.

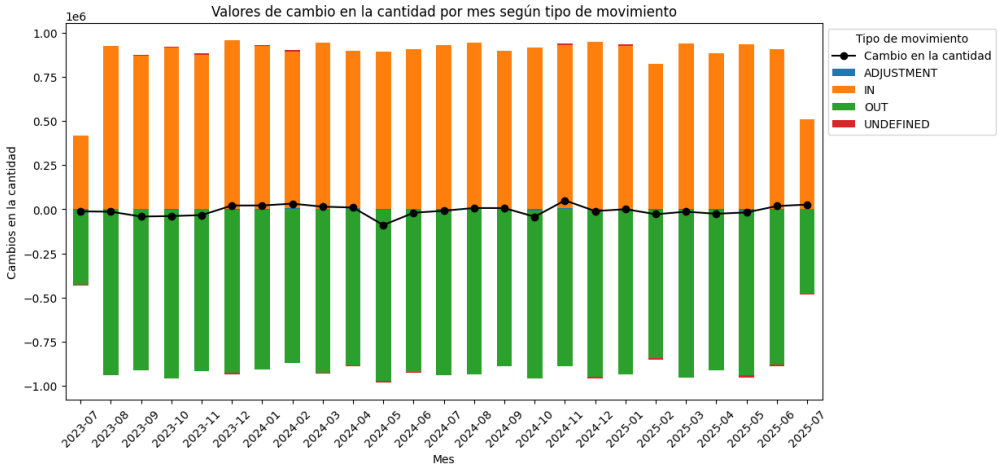


Figura 11: Valores de cambio en la cantidad por mes según tipo de movimiento

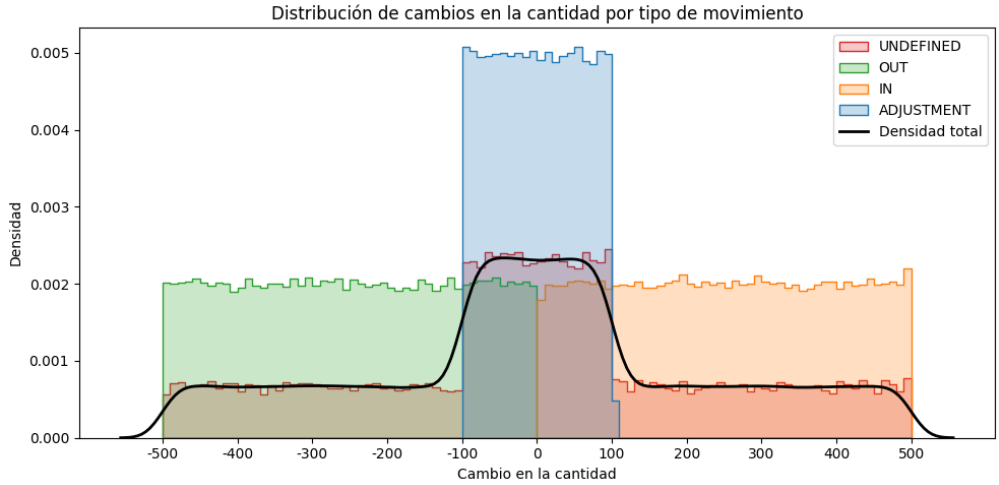


Figura 12: Distribución de valores de cambio en la cantidad según tipo de movimiento

4. Consultas y Visualizaciones

Todas las consultas siguen un flujo de trabajo similar:

1. Limpieza y preprocesamiento de los datos. Esto es selección de columnas necesarias, manejo de valores nulos, creación de nuevas columnas si es necesario, etc.
2. Filtrado de los datos según las condiciones de la consulta.
3. Realización de merges, groupby y operaciones entre columnas necesarias.
4. Visualización y análisis de los resultados obtenidos.

Esta forma de proceder, primero seleccionar columnas y filtrar filas, y luego realizar operaciones, es importante para optimizar el rendimiento de las consultas, ya que de esta manera se reduce la cantidad de datos que se deben procesar en los pasos posteriores.

4.1. Consultas Propuestas por el Enunciado

A continuación se presentan las consultas propuestas por el enunciado, junto con las consideraciones tomadas para su resolución y los resultados obtenidos. El código fuente y los resultados completos pueden encontrarse en el notebook `consultas_enunciado.ipynb`.

4.1.1. ¿Cuál es el estado que más descuentos tiene en total? ¿y en promedio?

Para esta consulta se tomaron algunas consideraciones:

- Se consideraron los 50 Estados de los Estados Unidos. No se consideraron territorios ni estados militares.
- El estado a considerar es el encontrado en la columna `billing_address` de la tabla `orders`.
- Se consideraron los .[es]tados con más descuentos.[a] aquellos que poseen la mayor cantidad de órdenes con descuentos aplicados. Se considera que una orden tiene un descuento si el monto del descuento no es nulo y es mayor a 0.
- La segunda parte de la consulta ("¿y en promedio?") se interpreta como el estado con mayor promedio en el valor de `discount_amount`.

Para poder realizar esta consulta, fue necesario extraer el estado y el código postal de la columna `billing_address` de la tabla `orders`.

Las direcciones parecían seguir un patrón que me facilitó extraer el estado y el código postal mediante una expresión regular. Para esto, primero se normalizaron los datos de la columna y luego se utilizó la siguiente expresión regular:

```
orders["billing_address"] = orders["billing_address"].str.upper()
orders.fillna({"billing_address":"UNDEFINED"}, inplace=True)
pattern = r'([A-Z]{2})\s(\d{5})'
orders[["state", "zip_code"]] = orders["billing_address"].str.extract(pattern)
```

La expresión regular extrae dos grupos: el primero corresponde al estado (dos letras mayúsculas) y el segundo al código postal (cinco dígitos). Para verificar que esta extracción fuera exitosa se realizaron las siguientes comprobaciones, para las cuales se obtuvieron resultados positivos (ver Anexo 5.1.1).

```
null_state_and_addr = orders["state"].isna() & orders["billing_address"].str.contains("UNDEFINED")

print("Todas las filas que tienen estado nulo, tienen direccion de facturacion indefinida?",
      "Si" if null_state_and_addr.sum() == orders["state"].isna().sum() else "No")
```

Finalmente, para responder la consulta, se creó un filtro que deja fuera los estados militares y territorios llamado `not_states_filter` (ver Anexo 5.1.1). Se filtraron las órdenes conservando solo aquellas que tenían un monto de descuento mayor a 0 y que cumplían con el filtro de estados, y se agruparon por estado, contando la cantidad de órdenes con descuento por estado.

```
orders_with_discount = orders.loc[orders["discount_amount"] > 0]
                             .loc[not_states_filter]

quantity_of_orders_with_discounts_by_state = orders_with_discount.groupby("state")["order_id"]
                                                                 .count().reset_index()
```

Luego, para visualizar mejor los resultados, se renombraron las columnas y se agregó una columna con el nombre del estado utilizando `map` (ver Anexo 5.1.1). Se visualizaron los 5 estados con más órdenes utilizando la función `nlargest` de pandas.

```
print("\nTop 5 estados con mas ordenes con descuentos:")
quantity_of_orders_with_discounts_by_state.nlargest(5, "Cantidad de Ordenes con Descuento")
```

Código de Estado	Cantidad de Órdenes con Descuento	Nombre del Estado
LA	13,950	Louisiana
MO	13,940	Missouri
IL	13,930	Illinois
KY	13,903	Kentucky
IA	13,873	Iowa

Cuadro 1: Top 5 estados con más órdenes con descuentos

Entonces la respuesta a "¿Cuál es el estado que más descuentos tiene en total?" es Louisiana.

Luego, para encontrar el estado con mayor promedio en el valor de `discount_amount`, se calculó el promedio del monto de descuento por estado y se utilizó nuevamente la función `nlargest` para obtener los 5 estados con mayor promedio.

```
states_avg_discount = orders_with_discount.groupby("state")["discount_amount"].mean().reset_index()
states_avg_discount.nlargest(5, "discount_amount")
```

Código de Estado	Promedio de Descuento ($)	Nombre del Estado
NC	50.67	North Carolina
GA	50.46	Georgia
OK	50.42	Oklahoma
CO	50.34	Colorado
MS	50.32	Mississippi

Cuadro 2: Top 5 estados con mayor promedio de monto de descuento

Entonces la respuesta a "¿y en promedio?" es North Carolina.

4.1.2. ¿Cuáles son los 5 códigos postales más comunes para las órdenes con estado 'Refunded'? ¿Y cuál es el nombre más frecuente entre los clientes de esas direcciones?

Para resolver esta consulta, se aprovechó la extracción del código postal realizada sobre la tabla `orders` en la consulta anterior. En primer lugar, se filtraron las órdenes con estado 'Refunded' y se contaron la cantidad de órdenes por código postal utilizando la función `value_counts`. Luego, se utilizó las función `nlargest` para obtener los 5 códigos postales más comunes entre las órdenes con estado 'Refunded'.

```
refunded_orders = orders[orders["status"].str.contains("REFUNDED")]

amount_refunded_orders_by_zipcode = refunded_orders["zip_code"].value_counts().reset_index()
top_refunded_zipcodes = amount_refunded_orders_by_zipcode.nlargest(5, "count")

print("\nTop 5 codigos postales mas comunes para ordenes con estado 'Refunded':")
print(top_refunded_zipcodes)
```

Código Postal	Cantidad de Órdenes 'Refunded'
31571	6
65247	5
38151	5
09045	5
14396	5

Cuadro 3: Top 5 códigos postales más comunes para órdenes con estado 'Refunded'

Como los puestos 2, 3, 4 y 5 están empatados con 5 órdenes cada uno, se agregó una consulta adicional para ver todos los códigos postales que tienen 5 órdenes 'Refunded' (ver Anexo 5.1.2).

Para obtener los nombres más frecuentes entre los clientes de esos códigos postales, se filtró la tabla `customers` utilizando el método `isin` para seleccionar únicamente las filas cuyo `postal_code` se encuentra en un conjunto específico de valores. Luego se contaron las apariciones de cada nombre utilizando nuevamente la función `value_counts`. Para mostrar los 5 nombres más frecuentes, se utilizó la función `nlargest`.

```
most_common_names = customers
    .loc[customers["postal_code"].isin(top_refunded_zipcodes["zip_code"])]["first_name"].value_counts()
most_common_names.nlargest(5)
```

Nombre	Cantidad de Apariciones
UNDEFINED	4
RICHARD	2
MICHAEL	2
ROBERT	1
CYNTHIA	1

Cuadro 4: Top 5 nombres más frecuentes entre los clientes de los códigos postales con más órdenes 'Refunded'

Como puede verse, el nombre más frecuente entre los clientes de esas direcciones es 'UNDEFINED', que aparece 4 veces. Esto significa que hay 4 órdenes registradas por clientes sin nombre registrado. Nuevamente, se realizó una consulta adicional para ver otros nombres que aparecen una sola vez (ver Anexo 5.1.2).

4.1.3. Para cada tipo de pago y segmento de cliente, devolver la suma y el promedio expresado como porcentaje, de clientes activos y de consentimiento de marketing.

Para resolver esta consulta era necesario tener columnas tanto en la tabla `customers` como en la tabla `orders`. Por lo tanto, se realizó un `merge` entre ambas tablas utilizando la columna `customer_id` como clave.

```
orders_customers = orders.merge(customers, on="customer_id").reset_index()
```

Luego, para evitar contar dos veces a un mismo cliente que hizo varias órdenes con el mismo método de pago, se eliminaron las filas duplicadas basándose en las columnas `customer_id` y `payment_method`.

```
orders_customers_unique = orders_customers.drop_duplicates(subset=["payment_method", "customer_id"])
```

Finalmente, se agrupó por `payment_method` y `customer_segment`, y se calcularon las funciones de agregación necesarias para obtener la cantidad total de clientes, la cantidad de clientes activos y la cantidad de clientes con consentimiento de marketing. Luego, se calcularon los promedios expresados como porcentaje.

```
orders_customers_grouped = orders_customers_unique
    .groupby(["payment_method", "customer_segment"]).agg({
        "customer_id": "count",
        "is_active": ["sum", "mean"],
        "marketing_consent": ["sum", "mean"]
    })
orders_customers_grouped[('is_active', 'mean')] *= 100
orders_customers_grouped[('marketing_consent', 'mean')] *= 100
```

Me pareció interesante graficar los resultados utilizando un `heatmap` de la librería `seaborn`, para lo cual fue necesario reestructurar el DataFrame utilizando la función `unstack` (ver Anexo 5.1.3). En el notebook también pueden encontrarse los resultados expuestos como tabla.

4.1.4. Para los productos que contienen en su descripción la palabra 'stuff', calcular el peso total de su inventario agrupado por marca

Para esta consulta, se asumió que la columna `weight_kg` representa el peso unitario del producto en el inventario.

Primero se filtraron los productos que contienen la palabra 'stuff' en su descripción, sin importar mayúsculas o minúsculas. Para esto se utilizó el método `str.contains` de pandas, con el parámetro `case=False` para ignorar mayúsculas y minúsculas. También se seleccionaron únicamente las columnas relevantes para la consulta: `brand`, `weight_kg` y `stock_quantity`.

```
stuff_products = products
    .loc[products["description"].str.contains("STUFF", case=False)][["brand", "weight_kg", "stock_quantity"]]
```

Luego, se calculó el peso total de cada producto multiplicando el peso unitario por la cantidad en stock, y se creó una nueva columna llamada `total_weight_kg`. También se eliminaron las columnas que ya no eran necesarias.

```
stuff_products["total_weight_kg"] = stuff_products["weight_kg"] * stuff_products["stock_quantity"]
stuff_products.drop(columns=["weight_kg", "stock_quantity"], inplace=True)
```

Finalmente, se agrupó por marca y se sumó el peso total de los productos de cada marca. Se utilizó la función `nlargest` para obtener las 5 marcas con mayor peso total en inventario.

```
stuff_products.groupby("brand")["total_weight_kg"].sum().nlargest(5)
```

Marca	Peso Total (kg)
UNDEFINED	11,157,882.66
3M	2,250,899.67
ADIDAS	1,923,907.88
NIKE	1,783,569.90
HASBRO	1,714,411.24

Cuadro 5: Top 5 marcas con mayor peso total de productos con 'stuff' en la descripción

4.2. Consultas Propias

A continuación se presentan consultas adicionales para cumplir con los requerimientos del trabajo práctico, junto con las consideraciones tomadas para su resolución y los resultados obtenidos. El código fuente y los resultados completos pueden encontrarse en el notebook `consultas_propias.ipynb`.

Las siguientes consultas fueron realizadas intentando cubrir las visualizaciones que se piden en la propuesta del trabajo práctico. Estas requieren la comparación de los siguientes tipos de variables:

1. Una continua con una línea de tiempo.
2. Una discreta con una continua.
3. Una discreta con una discreta.
4. Una continua con otra continua.

Además, se agregaron las siguientes visualizaciones adicionales:

5. Un heatmap.
6. Línea de regresión.
7. Boxplot.
8. Treemap.

4.2.1. Evolución del ingreso neto de dólares a lo largo del tiempo

Para calcular el ingreso neto de dólares, utilicé la tabla `orders` que contiene el tipo de cambio utilizado en cada orden.
Consideré el ingreso neto como la diferencia entre el total pagado por orden y el total de descuentos e impuestos aplicados. La fórmula que surge es la siguiente:

$$\texttt{Ingreso Neto} = \texttt{Total Pagado} - \texttt{Descuentos} - \texttt{Impuestos}$$

Para conseguir una evolución a lo largo del tiempo y poder graficarlo como una variable continua, agrupé los datos por fecha de creación de la orden, sumando las columnas de ingreso neto.

```
data = orders.loc[orders.currency == "USD"][["date","total_amount","discount_amount","tax_amount"]]
data["net_income"] = data["total_amount"] - data["discount_amount"] - data["tax_amount"]
data = data.groupby("date")["net_income"].sum().reset_index()
```

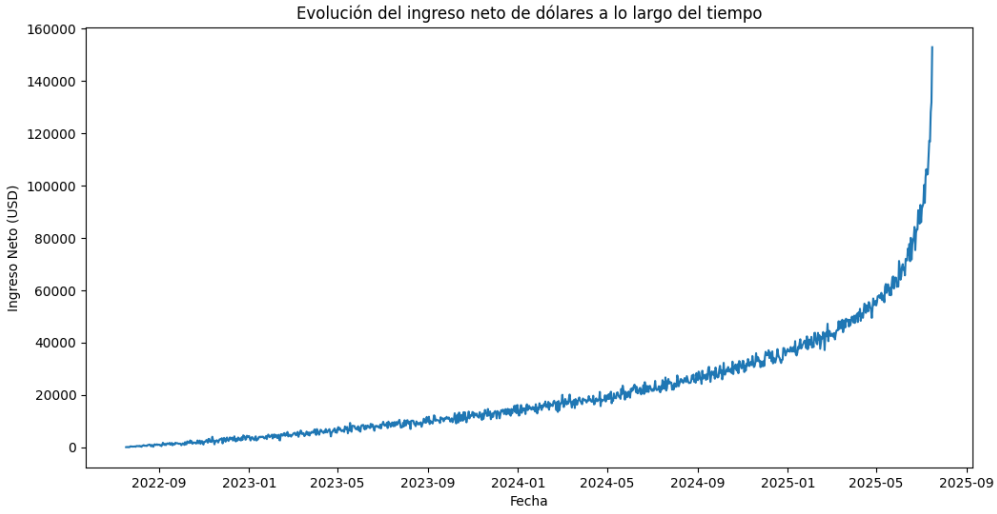


Figura 13: Evolución del ingreso neto de dólares a lo largo del tiempo.

Puede observarse en la figura 13 que el ingreso neto de dólares ha tenido una tendencia creciente muy marcada a lo largo del tiempo, similar a la encontrada en la figura 7. Esto nos da la idea que el ingreso neto aumenta a medida que aumenta la cantidad de órdenes.

4.2.2. Cantidad de dólares gastados por método de pago por clientes militares activos

Se considera que un cliente es 'militar' si su dirección tiene el estado 'AA', 'AE' o 'AP'. Además, se considera que un cliente es 'activo' si la columna `is_active` de la tabla `customers` es verdadera.

Primero filtré los clientes que tienen direcciones militares y que tienen la columna `is_active` en verdadero, y conservé únicamente los IDs de los clientes. Luego, filtré la tabla `orders` para quedarme con las órdenes realizadas por estos clientes y que estén en dólares, utilizando un llamado a `loc` y otro a `merge`. Finalmente, agrupé por método de pago y sumé el total gastado.

```
military_states = ['AA', 'AE', 'AP']
active_military_customers = customers
       .loc[(customers['is_active']) & (customers['state'].isin(military_states))]['customer_id']
       .reset_index()

active_military_orders = orders
       .loc[orders.currency == "USD"]
       .merge(active_military_customers, on='customer_id', how='inner')

payment_method_sum = active_military_orders.groupby('payment_method')['total_amount'].sum()
```

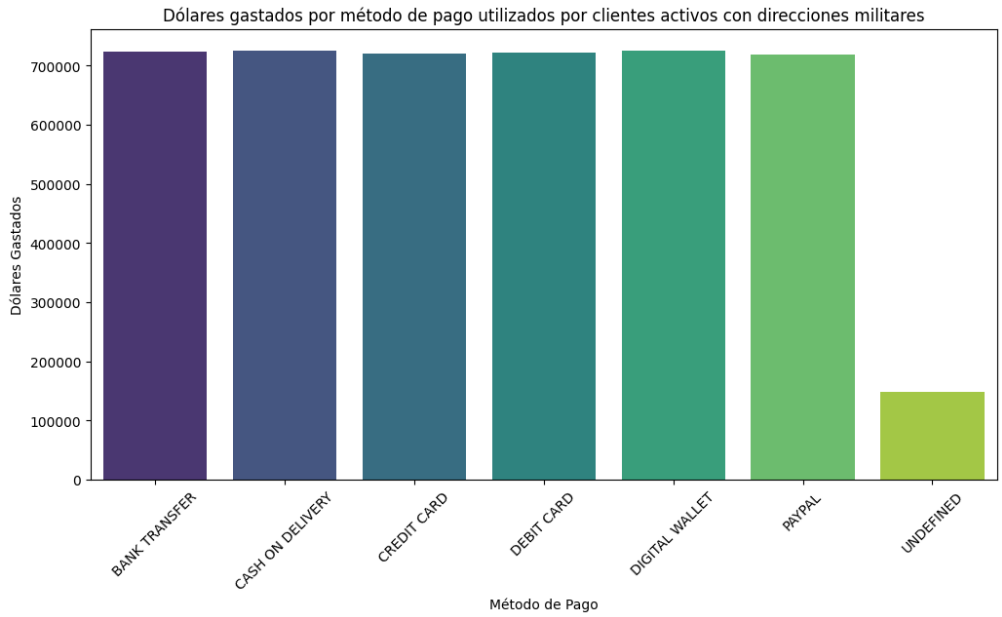


Figura 14: Cantidad de dólares gastados por método de pago por clientes activos con direcciones militares.

Podemos observar una cantidad similar de dólares gastados en cada método de pago, solamente notando una diferencia notable en el método de pago 'undefined', que tiene un gasto considerablemente menor.

4.2.3. Distribución de calificaciones según segmento de cliente.

Para esta consulta combiné las tablas `customers` y `reviews` utilizando un llamado a `merge`, ya que necesitaba la columna `customer_segment` de la tabla `customers` y la columna `rating` de la tabla `reviews`. Luego, para calcular la distribución de calificaciones debería contar las apariciones de cada calificación (de 1 a 5) por cada segmento de cliente y luego dividir por el total de reviews en ese segmento. Encontré que esto puede hacerse utilizando únicamente el método `value_counts` estableciendo como verdadero el parámetro `normalize`.

```
reviews = reviews.merge(customers[['customer_id', 'customer_segment']], on='customer_id', how='left')
reviews.fillna({'customer_segment': 'UNDEFINED'}, inplace=True)
data = reviews.groupby('customer_segment')['rating'].value_counts(normalize=True).unstack()
```

Decidí completar los valores faltantes en la columna `customer_segment` con el valor 'UNDEFINED', ya que no se puede inferir de ninguna manera el valor para esa columna, pero aún quería conservar todas las reviews en el análisis.

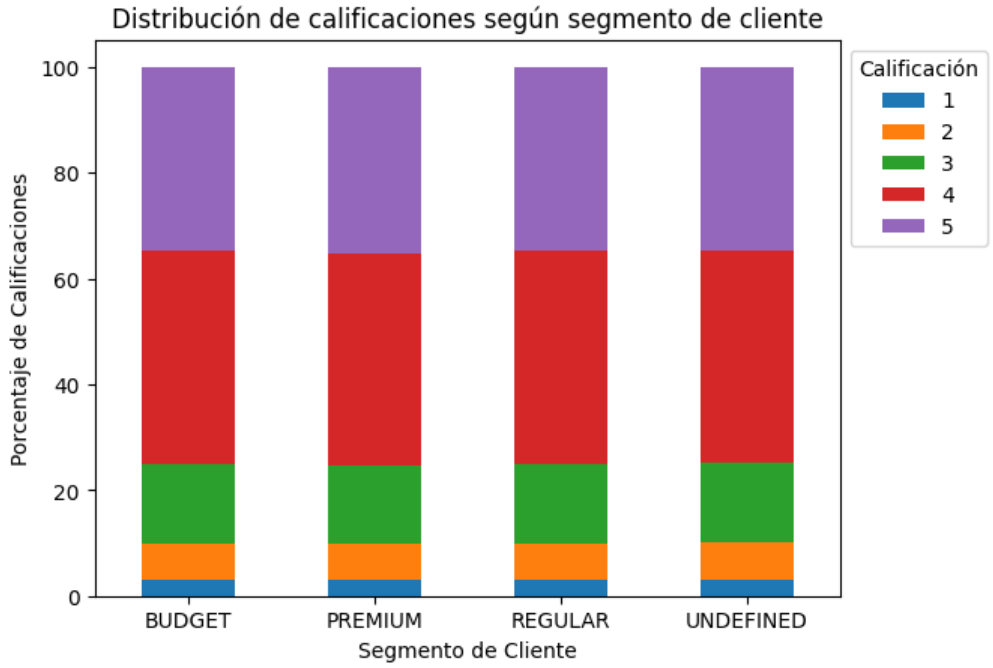


Figura 15: Distribución de calificaciones según segmento de cliente.

Pueden observarse nuevamente distribuciones muy uniformes, sin una diferencia notable entre los segmentos de clientes. Puede identificarse que la mayoría de calificaciones son de 4 y 5 estrellas, y que las calificaciones de 1 estrella son las menos frecuentes.

4.2.4. Relación entre stock disponible y ventas de unidades de zapatos

Realicé esta consulta con la idea de poder ver si existe alguna relación entre la cantidad de stock disponible y la cantidad de unidades vendidas de zapatos. Quería ver si a mayor stock disponible, mayor cantidad de ventas, o si por el contrario no existe una relación entre estas variables.

Para esto, consideré que el stock encontrado en la tabla de `products` es el stock actual del producto. Se compara contra las ventas históricas de unidades de ese producto, que se encuentran en la tabla `order_items`. No fue posible filtrar según tiempos de ventas, ya que esa información no se encuentra en la tabla `order_items`, y no es posible cruzar esta tabla con la tabla `orders` ya que no identificadores de órdenes en común.

Para realizar esta consulta primero identifiqué los IDs de las categoráis de zapatos, y los utilicé para buscar los IDs de los productos que pertenecen a estas categorías. Luego, utilicé estos IDs para filtrar la tabla `order_items` y quedarme con las ventas de unidades de zapatos. Agrupé por ID de producto y sumé la cantidad de unidades vendidas. Finalmente, realicé un `merge` con la tabla `products` para obtener el stock disponible de cada producto, y con la previemante generada `shoes_categories` para obtener los nombres de las categorías.

Me pareció curioso que al graficar esto en un scatter plot, la leyenda de colores no contenía únicamente las categorías de zapatos, sino que también aparecían categorías de otros tipos de productos. Esto se debía a que la columna `category_name` es de tipo `category` y aún luego de ser filtrada y mergeada, conservaba los valores originales. Para solucionar esto, convertí la columna a tipo `string`.

```
shoes_categories = categories[categories['parent_category'] == 'SHOES'][['category_id', 'category_name']]
shoes_products = products[products['category_id'].isin(shoes_categories['category_id'])]

shoes_order_items = items[items['product_id'].isin(shoes_products['product_id'])]
.groupby('product_id')['quantity'].sum().reset_index()

data = shoes_order_items.merge(
products[['product_id', 'stock_quantity', 'category_id']], on='product_id', how='left'
)
data = data.merge(
shoes_categories[['category_id', 'category_name']], on='category_id', how='left'
)
```

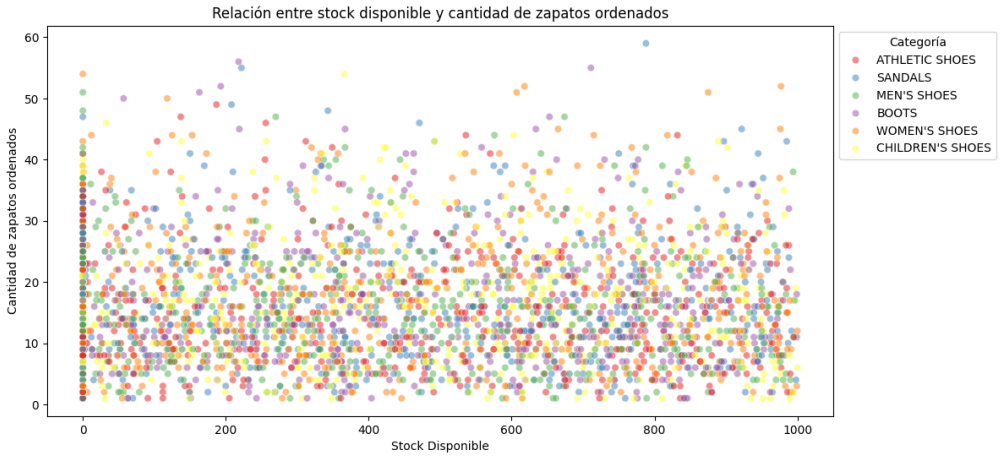


Figura 16: Relación entre stock disponible y ventas de unidades de zapatos.

Lamentablemente, no se observa una relación clara entre las variables. Tampoco se puede observar una diferencia clara entre el comportamiento de las distintas categorías de zapatos.
Puede observarse una concentración elevada en los puntos en el valor de x=0, que indica que hay muchos productos de zapatos que no tienen stock.

4.2.5. Rating promedio por categoría padre y segmento de cliente

Para esta consulta combiné las tablas `reviews`, `products`, `categories` utilizando llamados a `merge`, ya que necesitaba la columna `rating` de la tabla `reviews`, la columna `category_id` de la tabla `products`, la columna `parent_category` de la tabla `categories`. También necesitaba la columna `customer_segment` de la tabla `customers`, que ya se encontraba en la tabla `reviews` una consulta anterior.
Luego agrupé por categoría padre y segmento de cliente, y calculé el promedio de las calificaciones.

```
reviews = reviews.merge(products[['product_id', 'category_id']], on='product_id', how='left')
reviews = reviews.merge(categories[['category_id', 'parent_category']], on='category_id', how='left')
data = reviews.groupby(['parent_category', 'customer_segment'])['rating'].mean().unstack()
```

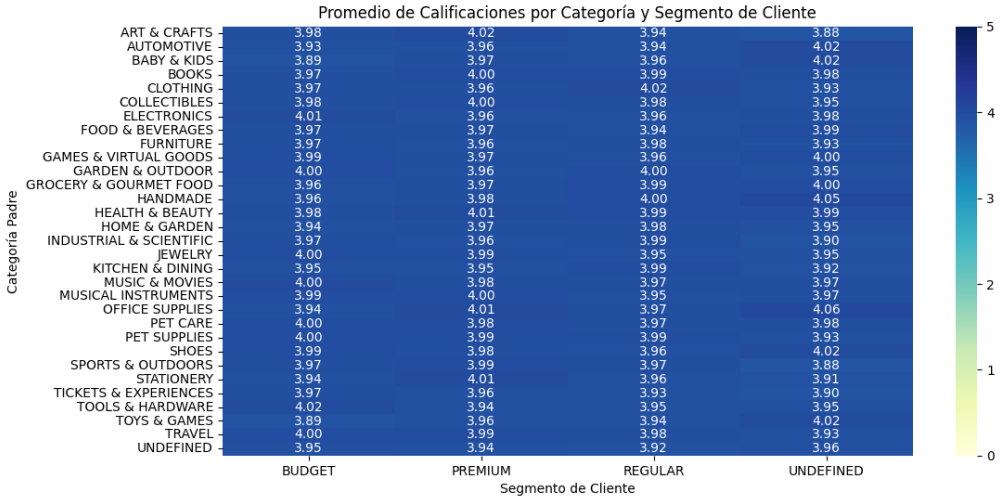


Figura 17: Rating promedio por categoría padre y segmento de cliente.

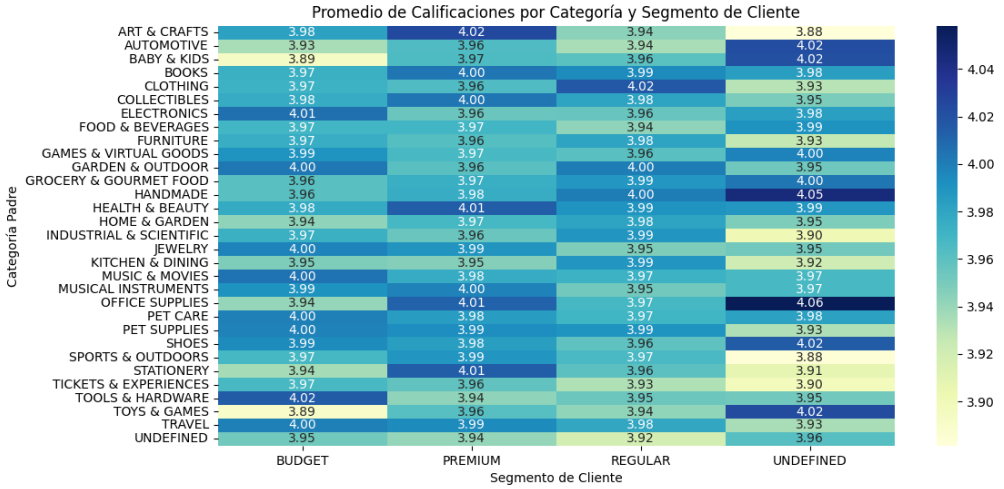


Figura 18: Rating promedio por categoría padre y segmento de cliente, con una escala de color más cercana.

Puede observarse en la figura 17 que todos los ratings promedio se encuentran muy cercanos a 4, obteniendo un heatmap de color casi uniforme.
En la figura 18 se ajusta la escala de colores para poder observar mejor las diferencias entre los ratings promedio. La escala va aproximadamente desde 3.85 hasta 4.10, lo cual es bastante pequeño. Aún con esta escala ajustada, se pueden observar que la mayoría de ratings se encuentran muy cercanos entre sí.

4.2.6. Relación entre precio de Smartphones activos y sus unidades vendidas

Originalmente, esta consulta se había planteado como la relación entre el precio de smartphones activos y su rating promedio. Sin embargo, como se observó en consultas anteriores, los ratings promedio de los productos se encuentran todos muy cercanos entre sí, por lo que no sería posible observar una relación clara entre estas variables. Por este motivo, se decidió cambiar la variable a comparar contra el precio por la cantidad de unidades vendidas.

Para realizar esta consulta, primero identifiqué el ID de la categoría 'SMARTPHONES', y lo utilicé para

buscar los IDs de los productos que pertenecen a esta categoría y que además están activos. Luego, utilicé estos IDs para filtrar la tabla `order_items` y agrupé por ID de producto para sumar la cantidad de unidades vendidas. Finalmente, realicé un `merge` con la tabla `products` para obtener el precio de cada producto.

```
smartphone_category_id = categories[
    categories['category_name'] == 'SMARTPHONES'
    ]['category_id'].iloc[0]

active_smartphones = products[
    (products['category_id'] == smartphone_category_id) & (products['is_active'])
    ][['product_id', 'price']]

data = items[items['product_id'].isin(active_smartphones['product_id'])]
          .groupby('product_id')['quantity'].sum().reset_index()

data = data.merge(active_smartphones[['product_id', 'price']], on='product_id', how='left')
```

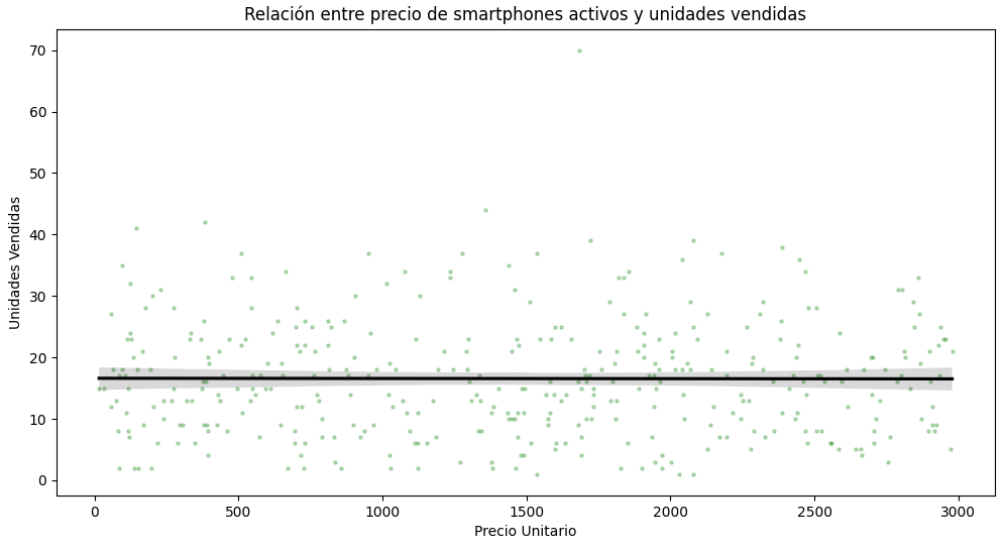


Figura 19: Relación entre precio de smartphones activos y unidades vendidas.

Una vez más, la distribución de los puntos es bastante uniforme, sin una relación clara entre las variables. No se observa que a mayor precio, mayor o menor cantidad de unidades vendidas. Más aún, puede observarse que la línea de regresión es prácticamente horizontal, indicando que no existe una relación lineal entre las variables sino que se distribuyen de manera uniforme.

4.2.7. Distribución del descuento unitario para las 5 categorías de producto más vendidas

Para hallar el descuento unitario de la venta de un producto, supuse que la columna `discount_amount` de la tabla `order_items` representa el descuento total aplicado a la venta de ese producto en esa orden. Por lo tanto, el descuento unitario puede hallarse dividiendo este valor por la cantidad de unidades vendidas de ese producto en esa orden.

Primero encontré las 5 categorías de productos más vendidas sumando la cantidad de unidades vendidas por categoría. Para esto, realicé un `merge` entre las tablas `order_items` y `products` para poder obtener el ID de la categoría de cada producto vendido. Aproveché este paso para quedarme únicamente con las columnas que voy a utilizar en esta consulta. Luego, agrupé por ID de categoría y sumé la cantidad de unidades

vendidas, quedándome con las 5 categorías con mayor cantidad de ventas utilizando `nlargest`.
Luego, filtré las filas de la tabla `order_items` para quedarme únicamente con los productos que pertenecen a estas categorías, para poder visuaizar la distribución de esos valores. Finalmente, calculé el descuento unitario y graficé su distribución utilizando violinplot.

```
items_categories = items.merge(
    products[['product_id', 'category_id']], on='product_id', how='left'
    )[['product_id', 'category_id', 'quantity', 'discount_amount']]

top_5_categories = items_categories.groupby('category_id')['quantity'].sum().nlargest(5)

items_categories = items_categories[items_categories['category_id'].isin(top_5_categories.index)]
items_categories['unit_discount'] = items_categories['discount_amount'] / items_categories['quantity']
```

Para quedarme con los nombres de las categorías en lugar de sus IDs, armé un diccionario a partir de la tabla `categories` y utilicé el método `map` para crear una nueva columna con los nombres.

```
categories_names = categories[['category_id', 'category_name']]
    .loc[categories['category_id'].isin(top_5_categories.index)].reset_index()

    category_name_map = {}
for _, row in categories_names.iterrows():
    category_name_map[row['category_id']] = row['category_name']

items_categories['category_id'] = items_categories['category_id'].map(category_name_map)
items_categories.rename(columns={'category_id': 'category_name'}, inplace=True)
```

También reemplacé los valores nulos de la columna `unit_discount` por 0, ya que supuse que estos valores se producen cuando no hubo descuento aplicado en la venta del producto.

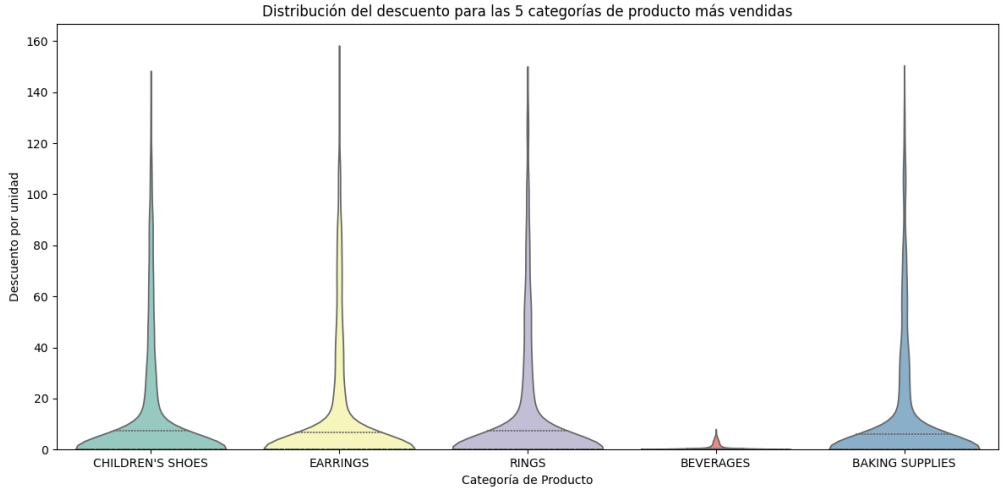


Figura 20: Distribución del descuento unitario para las 5 categorías de producto más vendidas.

Mostré los violin plots 'truncados' en la parte inferior ya que el descuento unitario no puede ser negativo. Notar cómo la distribución de las categorías es casi idéntica, aún para categorías de productos diferentes como 'CHILDREN'S SHOES', 'RINGS', y 'BAKING SUPPLIES'. La única excepción es la categoría 'BEVERAGES', que tiene una distribución un poco más concentrada en valores bajos de descuento unitario.

4.2.8. Unidades de productos robados por categoría y subcategoría de producto

Para esta consulta, consideré que un producto fue robado si la columna razón del registro de la tabla `inventory_logs` es igual a 'STOLEN', y el valor de la columna `quantity_change` es menor a 0, lo cual indicaría que el producto fue sustraído del inventario.

Primero filtré la tabla `inventory_logs` para quedarme únicamente con los registros que cumplen estas condiciones. Luego, mergeé con la tabla `products` para obtener el ID de la categoría de cada producto robado, y con la tabla `categories` para obtener los nombres de las categorías y subcategorías. Finalmente, agrupé por categoría y subcategoría, sumando la cantidad de unidades robadas.
Finalmenente, tuve que quedarme con los valores absolutos de la cantidad de unidades robadas, ya que al ser valores negativos no se podían graficar en un treemap.

```
stealed_orders = inventory[
    (inventory['reason'] == 'THEFT') & (inventory['quantity_change'] < 0)
    ][['quantity_change', 'product_id']]

stealed_orders = stealed_orders.merge(
    products[['product_id', 'category_id']], on='product_id'
    )
stealed_orders = stealed_orders.merge(
    categories[['category_id', 'category_name', 'parent_category']], on='category_id'
    )[['quantity_change', 'category_name', 'parent_category']]

stealed_orders = stealed_orders.groupby(
    ['parent_category', 'category_name']
)['quantity_change'].sum().reset_index()

stealed_orders['quantity_change'] = stealed_orders['quantity_change'].abs()
```

Pueden observarse en la figura 21 cantidades similares para todas las categorías padre y subcategorías, sin una diferencia notable entre ellas. Esto en un escenario real resultaría extraño, ya que se esperaría que algunas categorías de productos sean más propensos a ser robadas que otros.
La gran cantidad de categorías y subcategorías dificulta la visualización, pero puede observarse que todas tienen valores similares.
En el notebook puede encontrarse el código para graficar el treemap de la figura 21 utilizando la librería `plotly`, el cual genera una versión interactiva del gráfico en la que se pueden visualizar mejor las categorías de manera individual.

Adicionalmente, agregué un gráfico de tipo sunburst, que es similar al treemap pero muestra la jerarquía de categorías de manera radial. El código para generarlo y poder visuaizarlo de forma interactiva también puede encontrarse en el notebook
En este gráfico puede observarse de manera más clara que todas las categorías y categorías padre tienen valores similares.

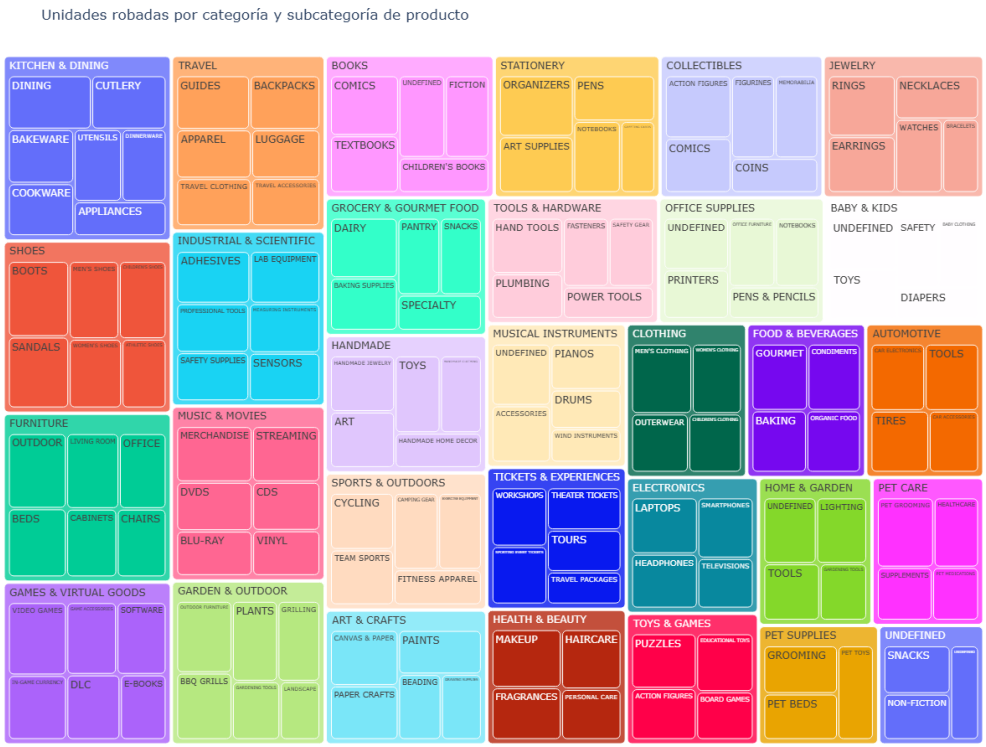


Figura 21: Unidades de productos robados por categoría y subcategoría de producto.

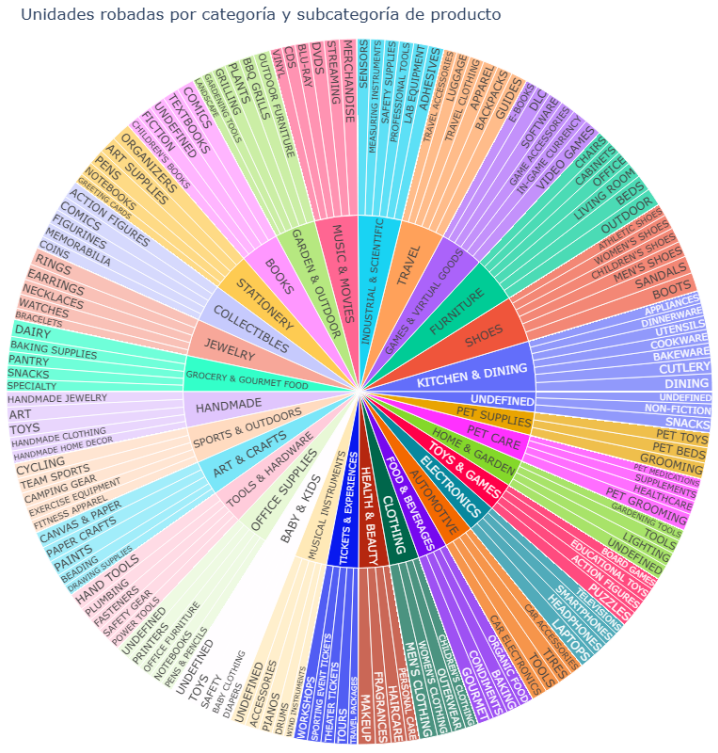


Figura 22: Unidades de productos robados por categoría y subcategoría de producto, visualizado como sunburst.

5. Anexo

5.1. Output de la Consultas Propuestas por el Enunciado

Estos outputs pueden encontrarse en el notebook `propuestas_del_enunciado.ipynb`.

5.1.1. Consulta 1

Output de la Validación

```
Filas totales en dataset orders: 4700000
Filas con estado no nulos: 4277862
Filas con estado nulo: 422138
Todas las filas que tienen estado nulo, tienen direccion de facturacion indefinida? Si
```

Filtro de Estados

```
not_states_filter = ~(
    orders["state"].str.contains("AA") # Military Americas
    | orders["state"].str.contains("AE") # Military Europe
    | orders["state"].str.contains("AP") # Military Pacific
    | orders["state"].str.contains("FM") # Federated States of Micronesia
    | orders["state"].str.contains("MH") # Marshall Islands
    | orders["state"].str.contains("MP") # Northern Mariana Islands
    | orders["state"].str.contains("PW") # Palau
    | orders["state"].str.contains("GU") # Guam
    | orders["state"].str.contains("VI") # U.S. Virgin Islands
    | orders["state"].str.contains("AS") # American Samoa
    | orders["state"].str.contains("PR") # Puerto Rico
    | orders["state"].str.contains("DC") # District of Columbia
    | orders["state"].isna()          # Nulos
)
print("Cantidad de estados: ", orders.loc[not_states_filter]["state"].unique().size)
print("Estados considerados: ", orders.loc[not_states_filter]["state"].unique())
```

```
Cantidad de estados: 50
Estados considerados: ['ND' 'NJ' 'MN' 'WI' 'OH' 'NV' 'MA' 'AZ' 'MO' 'VT' 'MI' 'NY' 'NM' 'PA'
 'WY' 'NE' 'WV' 'KY' 'WA' 'TX' 'OK' 'ME' 'KS' 'IN' 'FL' 'MD' 'MS' 'AL'
 'MT' 'ID' 'NC' 'AK' 'SD' 'NH' 'SC' 'CT' 'CA' 'CO' 'GA' 'IA' 'VA' 'OR'
 'DE' 'LA' 'UT' 'AR' 'IL' 'TN' 'RI' 'HI']
```

Formateo de los Resultados El archivo json utilizado con los nombres de estado puede encontrarse junto con el código fuente del proyecto.

```
import json
with open("state_names.json", "r") as f:
    state_names = json.load(f)

quantity_of_orders_with_discounts_by_state["Nombre del Estado"] =
    quantity_of_orders_with_discounts_by_state["state"].map(state_names)

quantity_of_orders_with_discounts_by_state.rename(
    columns={
        "order_id": "Cantidad de Ordenes con Descuento",
        "state": "Codigo de Estado"
        }, inplace=True
    )
```

5.1.2. Consulta 2

Códigos postales con 5 órdenes 'Refunded'

```
print("\nCodigos postales con 5 ordenes 'Refunded':")
print(amount_refunded_orders_by_zipcode.loc[amount_refunded_orders_by_zipcode["count"] == 5])
```

```
Codigos postales con 5 ordenes 'Refunded':
   zip_code count
1     65247     5
2     38151     5
3      9045     5
4     14396     5
5     73291     5
6     91623     5
```

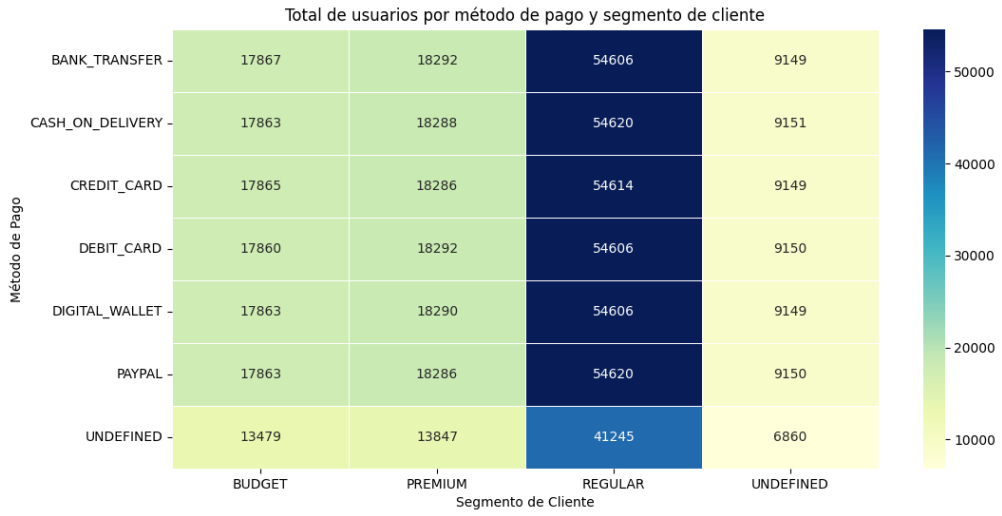
Otros nombres que aparecen en órdenes 'Refunded'

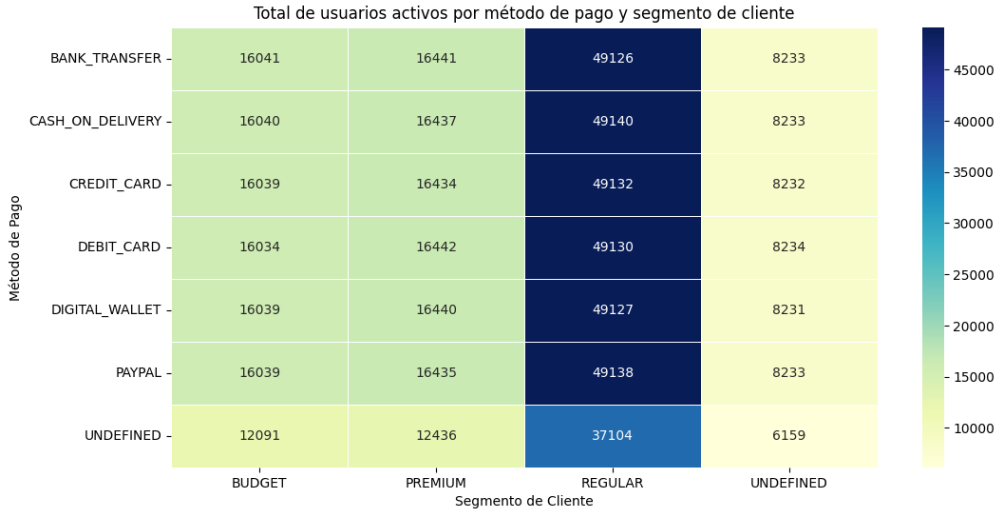
```
print("\nOtros nombres que aparecen una sola vez:")
print(most_common_names.loc[most_common_names == 1].head(7))
```

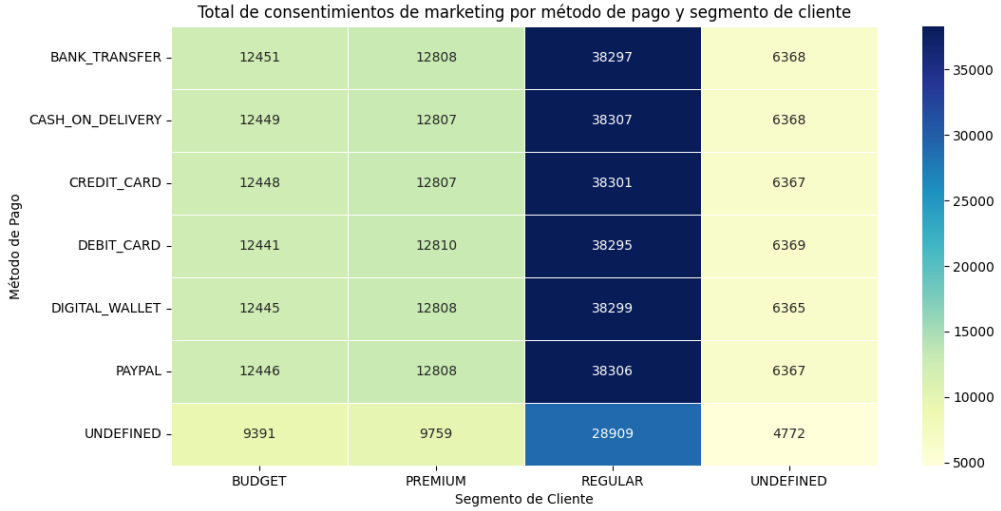
```
Otros nombres que aparecen una sola vez:
first_name
ROBERT    1
CYNTHIA   1
JAY       1
MARIA     1
KAREN     1
CARLA     1
WILLIAM   1
```

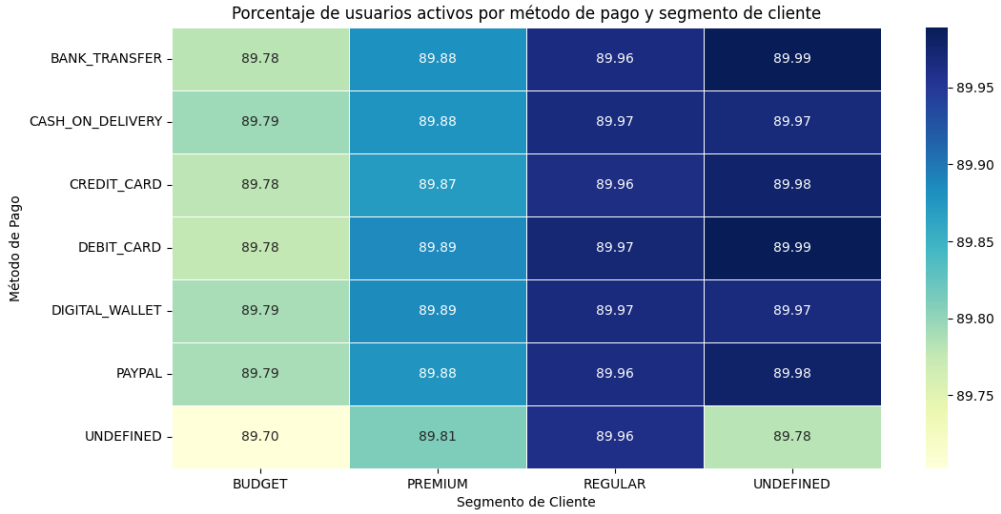
5.1.3. Consulta 3

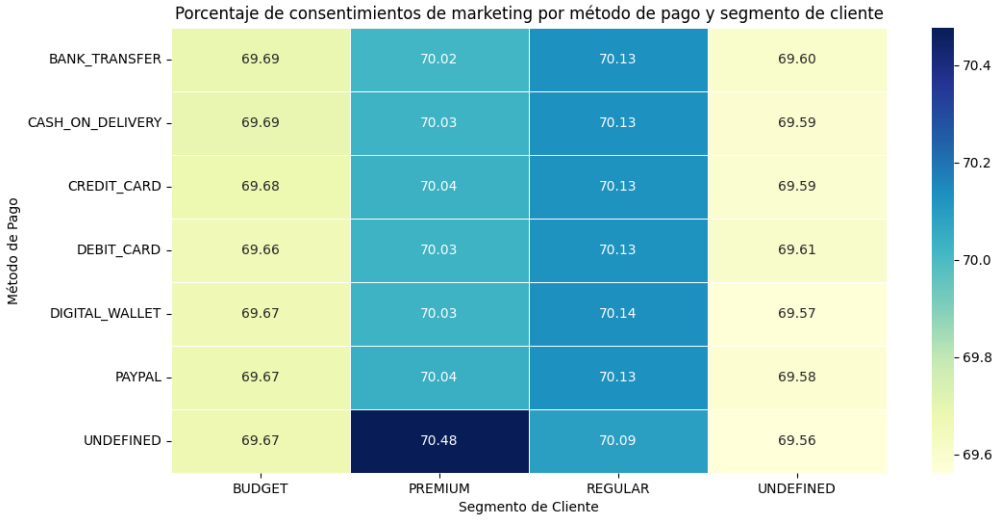
Gráficos Heatmap resultantes




Total de usuarios activos por método de pago y segmento de cliente
BANK_TRANSFER
CASH_ON_DELIVERY
CREDIT_CARD
DEBIT_CARD
DIGITAL_WALLET
PAYPAL
UNDEFINED
16041 16441 49126 8233
16040 16437 49140 8233
16039 16434 49132 8232
16034 16442 49130 8234
16039 16440 49127 8231
16039 16435 49138 8233
12091 12436 37104 6159
Método de Pago
BUDGET
PREMIUM
REGULAR
UNDEFINED
Segmento de Cliente
45000
40000
35000
30000
25000
20000
15000
10000


Total de consentimientos de marketing por método de pago y segmento de cliente
BANK_TRANSFER
CASH_ON_DELIVERY
CREDIT_CARD
DEBIT_CARD
DIGITAL_WALLET
PAYPAL
UNDEFINED
12451 12808 38297 6368
12449 12807 38307 6368
12448 12807 38301 6367
12441 12810 38295 6369
12445 12808 38299 6365
12446 12808 38306 6367
9391 9759 28909 4772
Método de Pago
BUDGET
PREMIUM
REGULAR
UNDEFINED
Segmento de Cliente
35000
30000
25000
20000
15000
10000
5000


Porcentaje de usuarios activos por método de pago y segmento de cliente
BANK_TRANSFER
CASH_ON_DELIVERY
CREDIT_CARD
DEBIT_CARD
DIGITAL_WALLET
PAYPAL
UNDEFINED
89.78 89.88 89.96 89.99
89.79 89.88 89.97 89.97
89.78 89.87 89.96 89.98
89.78 89.89 89.97 89.99
89.79 89.89 89.97 89.97
89.79 89.88 89.96 89.98
89.70 89.81 89.96 89.78
Método de Pago
BUDGET
PREMIUM
REGULAR
UNDEFINED
Segmento de Cliente
89.95
89.90
89.85
89.80
89.75


Porcentaje de consentimientos de marketing por método de pago y segmento de cliente
Método de Pago
BANK_TRANSFER
CASH_ON_DELIVERY
CREDIT_CARD
DEBIT_CARD
DIGITAL_WALLET
PAYPAL
UNDEFINED
69.69
70.02
70.13
69.60
69.69
70.03
70.13
69.59
69.68
70.04
70.13
69.59
69.66
70.03
70.13
69.61
69.67
70.03
70.14
69.57
69.67
70.04
70.13
69.58
69.67
70.48
70.09
69.56
BUDGET
PREMIUM
REGULAR
UNDEFINED
Segmento de Cliente
70.4
70.2
70.0
69.8
69.6