Trabajo Practico N.º 1
Pandas

Bautista Boeri
110898

Índice

1. Introducción general

En el presente informe se realiza un análisis exhaustivo de los datos proporcionados, con el objetivo de explorar patrones, relaciones y posibles irregularidades en el conjunto de información. Para ello, se presentan diversas consultas estructuradas acompañadas de sus respectivas hipótesis, resultados y visualizaciones gráficas. Este enfoque permite interpretar de manera sistemática los datos, facilitando la comprensión del dominio y la identificación de posibles anomalías que requieran atención. Se deja a continuación acceso al notebook utilizado: Notebook de análisis en Google Colab.

2. Carga de Datos

Antes de comenzar con las consultas, cargamos todos los datasets a partir de los archivos guardados en el drive en formato pkl para una carga más veloz. Además, importamos todas las librerías necesarias para el análisis y visualización de los datos.

Código

```
# Instalar gdown (si no está instalado)
!pip install gdown

import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
import plotly.express as px
import plotly.graph_objects as go

# Descargar todos los archivos de la carpeta pública
carpeta_drive = "https://drive.google.com/drive/folders/1UXAAJ-XgEe5F89U03eNdQ5NBcwzP78A2"
!gdown --folder "{carpeta_drive}"

# Detectar automáticamente la carpeta creada
contenido = os.listdir("/content")
carpeta_descargada = [c for c in contenido if os.path.isdir(os.path.join("/content", c))][0]

ruta_guardado = "/content/datasets_pkl"

# Listar archivos .pkl en esa carpeta
archivos_pkl = [f for f in os.listdir(ruta_guardado) if f.endswith(".pkl")]

# Crear diccionario {nombre_archivo: DataFrame}
dfs = {f: pd.read_pickle(os.path.join(ruta_guardado, f)) for f in archivos_pkl}

print(f" Se cargaron {len(dfs)} DataFrames")
```

```
print("Archivos cargados:", list(dfs.keys()))
```

3. Ordenes y descuentos

3.1. Descuentos por Estado

Introducción

Se busca analizar la cantidad total de descuentos que hay por estado. También, se ve el promedio de estos descuentos.

Hipótesis

Se le adjudica el descuento al estado detallado en la dirección de envió. Ademas, se tienen en cuenta las direcciones las cuales contengan dos letras en mayúscula para poder identificar el estado. Se espera poder ver que estados tienen mas descuentos y también su promedio. Para el promedio se utiliza las ordenes con descuento mayor a 0 para hacer un promedio de los que si tienen descuento.

Código

```
df = dfs['orders.pkl'].copy()

df = df[['shipping_address', 'discount_amount']]


df["estado"] = df["shipping_address"].str.extract(r"\b([A-Z]{2})\
    b")
df['discount_amount'] = df['discount_amount'].fillna(0)
df = df[df['discount_amount']>0]

resumen = df.groupby("estado").agg(
    total=("discount_amount", "count"),
    descuento_promedio=("discount_amount", "mean")
)
```

```
plt.figure(figsize=(12,6))
plt.bar(resumen.index, resumen['total'])
plt.xticks(rotation=90)
plt.ylabel("Total descuentos acumulados")
plt.title("Total de descuentos por estado (ordenado)")
plt.show()
```

```
plt.figure(figsize=(12,6))
plt.bar(resumen.index, resumen['descuento_promedio'])
plt.xticks(rotation=90)
plt.ylabel("Total descuentos promedio")
plt.title("Total de descuentos promedio por estado ")
plt.show()
```

Visualizaciones

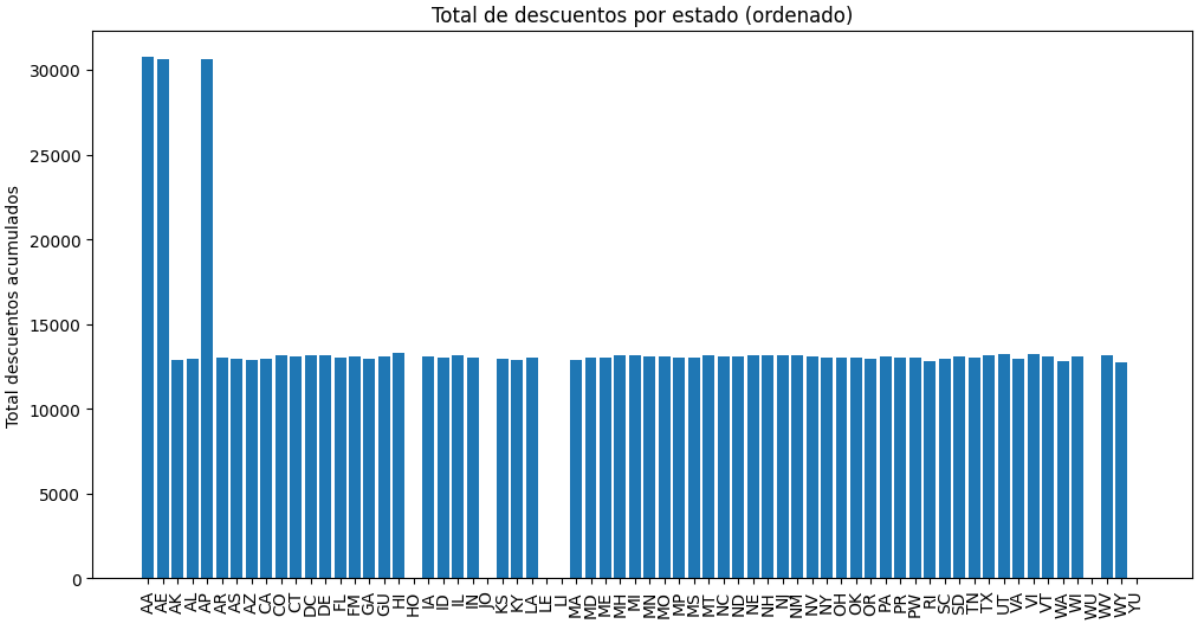


Figura 1: Total de descuentos por estado

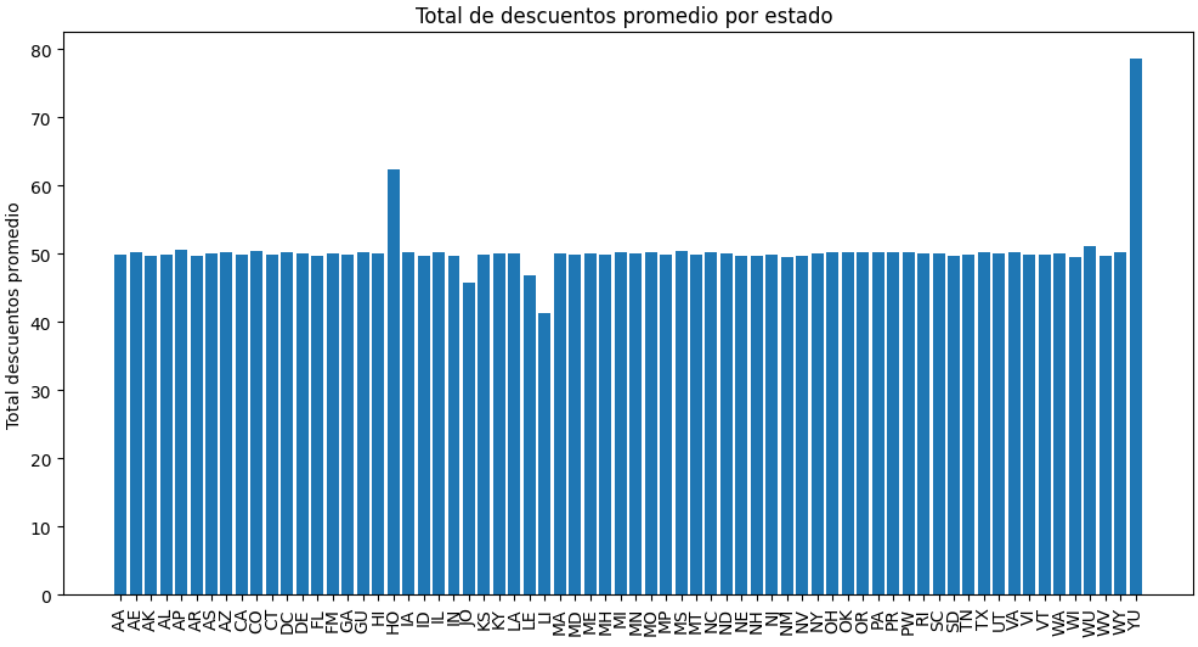


Figura 2: Descuento promedio por estado

Conclusión

Se observa que algunos estados concentran mayores montos totales de descuentos y algunos muy pocos. Pero en lineas generales, todos los estados tienen la misma cantidad de descuentos y promedio de descuentos mostrando un comportamiento bastante homogéneo con respecto a los descuento. Notemos que en los estados donde parece que no hay casi

descuentos, en estos el promedio varia mucho mas con respecto al promedio de los estados en general. Esto tiene sentido pues hay menos descuentos y puede variar en mayor grado el promedio.

3.2. Distribucion de descuentos en base al método de pago

Introducción

Se analiza la ditribucion de los descuentos acumulados en base a cada método de pago.

Hipótesis

Es importante tener en cuenta que solo se tienen en consideracion para la distribución a las ordenes que tienen descuento, es decir, que se les hizo un descuento mayor a 0. Se hizo esto para ver la distribución de los descuentos en mas detalle. Se espera poder ver si hay mas descuentos bajos que altos o hay una distribución masomenos igual sin importar la cantidad.

Código

```
df_orders = dfs['orders.pkl'].copy()
df_orders = df_orders[['order_id', 'payment_method', '
    discount_amount']]
df_orders = df_orders[df_orders['discount_amount']>0]


df_orders = df_orders[['order_id', 'payment_method', '
    discount_amount']]

df_orders.loc[:, 'payment_method'] = df_orders['payment_method'].
    str.strip().str.lower().str.capitalize()

plt.figure(figsize=(12,6))
```

```
plt.figure(figsize=(12,6))

sb.violinplot(
    data=df_orders,
    x='payment_method',
    y='discount_amount',
)

plt.xticks(rotation=45)
plt.xlabel("Método de Pago")
plt.ylabel("Monto de Descuento")
plt.title("Distribución de Descuentos por Método de Pago")
plt.tight_layout()
plt.show()
```

Resultados y visualizaciones

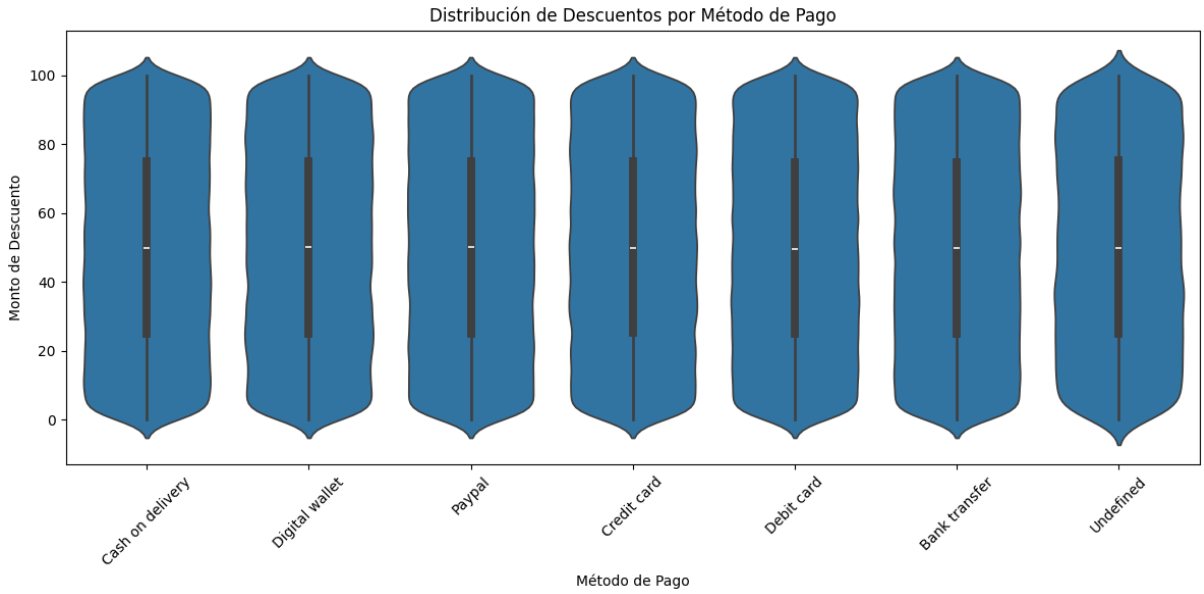


Figura 3: Distribución de los descuentos por cada método de pago

Conclusión

Se puede ver que la distribución de los descuentos es muy parecida entre todos los métodos de pago y dentro de cada método de pago en si también.

3.3. Ordenes por Mes en Categorías Padres

Introducción

Se analiza las órdenes por mes agrupadas por categoría padre. Solo se analizará el top 15 de las categorías padre.

Hipótesis

Se espera observar patrones de estacionalidad en las ventas dependiendo de la categoría principal del producto. Ademas, cualquier orden sin fecha de creacion, sera descartada.

Código

```
df_categorias = dfs['categories.pkl'].copy()
df_products= dfs['products.pkl'].copy()
df_orders = dfs['orders.pkl'].copy()
df_order_items = dfs['order_items.pkl'].copy()

df_orders['created_at'] = pd.to_datetime(df_orders['created_at'],
    format='mixed', errors='coerce')
df_orders['mes'] = df_orders['created_at'].dt.month

```

```
df_merge_categoria = df_categorias.merge(df_products, on='
    category_id')
df_merge_order = df_orders.merge(df_order_items, on='order_id')

df_merge_order_categoria = df_merge_order.merge(
    df_merge_categoria, on='product_id')
df_merge_order
```

Resultados y visualizaciones

```
0 rows x 24 columns
```

Conclusión

El análisis quedó incompleto por problemas de consistencia en los `order_id`, sin embargo, la metodología planteada permitiría detectar estacionalidad en categorías principales si se corrigieran las claves.

3.4. Ordenes de items por Mes y Estado

Introducción

Se analiza las órdenes por mes agrupadas por categoría padre. Solo se analizará el top 15 estados con mas ordenes de items.

Hipótesis

Se tiene en cuenta cualquier orden de ítems que tenga una fecha correcta y el estado detallado en la dirección de envío con el mismo formato que en la primera consulta. Ademas, se tendrán en cuenta solo los años en los que haya datos de todos los meses para hacer una comparación justa entre los meses.

Código

```
df_orders = dfs['orders.pkl'].copy()
df_orders = df_orders[['order_id', 'created_at', '
    shipping_address']]

df_orders['created_at'] = pd.to_datetime(df_orders['created_at'],
     errors='coerce')
df_orders = df_orders.dropna(subset=['created_at'])

df_orders["estado"] = df_orders["shipping_address"].str.extract(r
    "\b([A-Z]{2})\b")
df_orders['year'] = df_orders['created_at'].dt.year
df_orders['month'] = df_orders['created_at'].dt.month

años_completos = (
```

```
    df_orders.groupby('year')['month'].nunique()
    .loc[lambda s: s == 12]
    .index
)

df_fair = df_orders[df_orders['year'].isin(años_completos)]

df_grouped = df_fair.groupby(['month', 'estado']).size().unstack(
    fill_value=0)

top_15_states = df_grouped.sum().sort_values(ascending=False).
    head(15).index
df_top15 = df_grouped[top_15_states]

df_top15.index = df_top15.index.map(lambda x: pd.Timestamp(2000,
    x, 1).strftime('%b'))
```

```
plt.figure(figsize=(12, 8))
sb.heatmap(df_top15,
            annot=True,
            fmt='d',
            cmap="Reds",
            linewidths=0.5,
            linecolor='white',
            cbar_kws={'label': 'Cantidad de órdenes de items'})
```

Resultados y visualizaciones

Figura 4: Cantidad de ordenes por estado/mes

Conclusión

Se puede notar como las ordenes de items van aumentando a lo largo del año y este también es un patrón que pasa en todos los estados. Igualmente, podemos observar como en 3 estados(AP, AE y AA) tienen una cantidad sustancial mas de ordenes de ítems comparado con el resto.

3.5. Análisis jerárquico de ordenes de items y sus categorías

Introducción

Se hace un análisis de las ordenes de items en base a las categorías y sus categorías padre.

Hipótesis

No se tendrán en cuenta ordenes de items que no tengan un id de la categoría para no generar datos falsos. Si se tendrán en cuenta productos que no tengan una categoría padre o categoría determinada pues siguen siendo datos que pueden ser interés. En los datos proporcionados hay productos con distintos id pero con el mismo nombre, estos son tomados como dos productos distintos pues no sabemos bien por que están así separados.

Código

```
df_order_items = dfs['order_items.pkl'].copy()
df_products = dfs['products.pkl'].copy()
df_categorias = dfs['categories.pkl'].copy()

df_categorias.loc[:,'parent_category'] = df_categorias['
    parent_category'].str.strip().str.lower().str.capitalize()
df_categorias.loc[:,'category_name'] = df_categorias['
    category_name'].str.strip().str.lower().str.capitalize()

df_products.fillna({'category_id':'Undefined'}, inplace=True)
df_categorias.fillna({'category_id':'Undefined'}, inplace=True)
df_categorias.fillna({'parent_category':'Undefined'}, inplace=
    True)

df_merged = df_order_items.merge(df_products, on='product_id')

df_merged = df_merged[['category_id','order_id']].merge(
    df_categorias[['category_id','parent_category','category_name
        ']], on='category_id'
)

df_grouped = df_merged.groupby(['parent_category','category_id','
    category_name']).size()
df_grouped = df_grouped.reset_index().rename(columns={0:'Ordenes
    totales'})
```

```
fig = px.treemap(
    df_grouped,
    path=['parent_category', 'category_name'],
    values='Ordenes totales',
    color='Ordenes totales',
    hover_data={'Ordenes totales': True},
    color_continuous_scale='Reds'
)

fig.update_layout(
    title="Treemap jerárquico: Categoria padre -> Categoria",
    margin=dict(t=50, l=25, r=25, b=25)
)

fig.show()
```

Resultados y visualizaciones

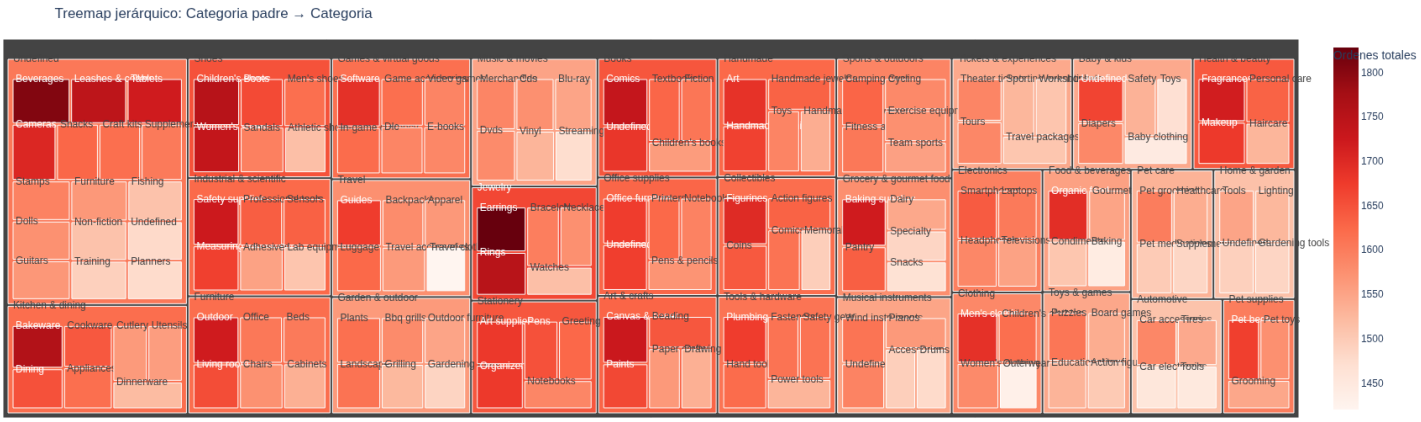


Figura 5: Cantidad de órdenes por estado y mes

Conclusión

Se puede observar mas en detalle el desglose de las ordenes por categorías y su categoría padre, ademas podemos notar como las ordenes de categorías padre no definida son las que mas suma en conjunto. Por lo que hay que tener cuidado con el uso de estos datos pues un gran porcentaje de las ordenes son de productos que no hay información sobre su categoría padre.

3.6. Ordenes en base a su estado y el medio de pago

Introducción

Se hace un análisis de la cantidad de ordenes que hay en base al método de pago y el estado de la orden.

Hipótesis

Se tienen en cuenta las compras con el método de pago/estado indefinido para ver si mantienen un comportamiento parecido. Se busca ver si hay variaciones al estado en los distintos metodos de pago.

Código

```
df_orders=dfs['orders.pkl'].copy()

df_orders = df_orders[['order_id', 'payment_method', 'status']]

df_orders.loc[:, 'payment_method'] = df_orders['payment_method'].
    str.strip().str.lower().str.capitalize()
df_orders.loc[:, 'status'] = df_orders['status'].str.strip().str.
    lower().str.capitalize()

```

```
df_orders.fillna('Undefined', inplace=True)

df_orders_grouped = df_orders.pivot_table(
    index="payment_method",
    columns="status",
    values="order_id",
    aggfunc="count",
    fill_value=0
)
```

```
df_orders_grouped.plot(kind="bar", stacked=True, figsize=(10,6))

plt.title("Órdenes por método de pago y status")
plt.ylabel("Cantidad de órdenes")
plt.xlabel("Método de pago")
plt.xticks(rotation=45)
plt.legend(title="Status")
plt.show()
```

Resultados y visualizaciones

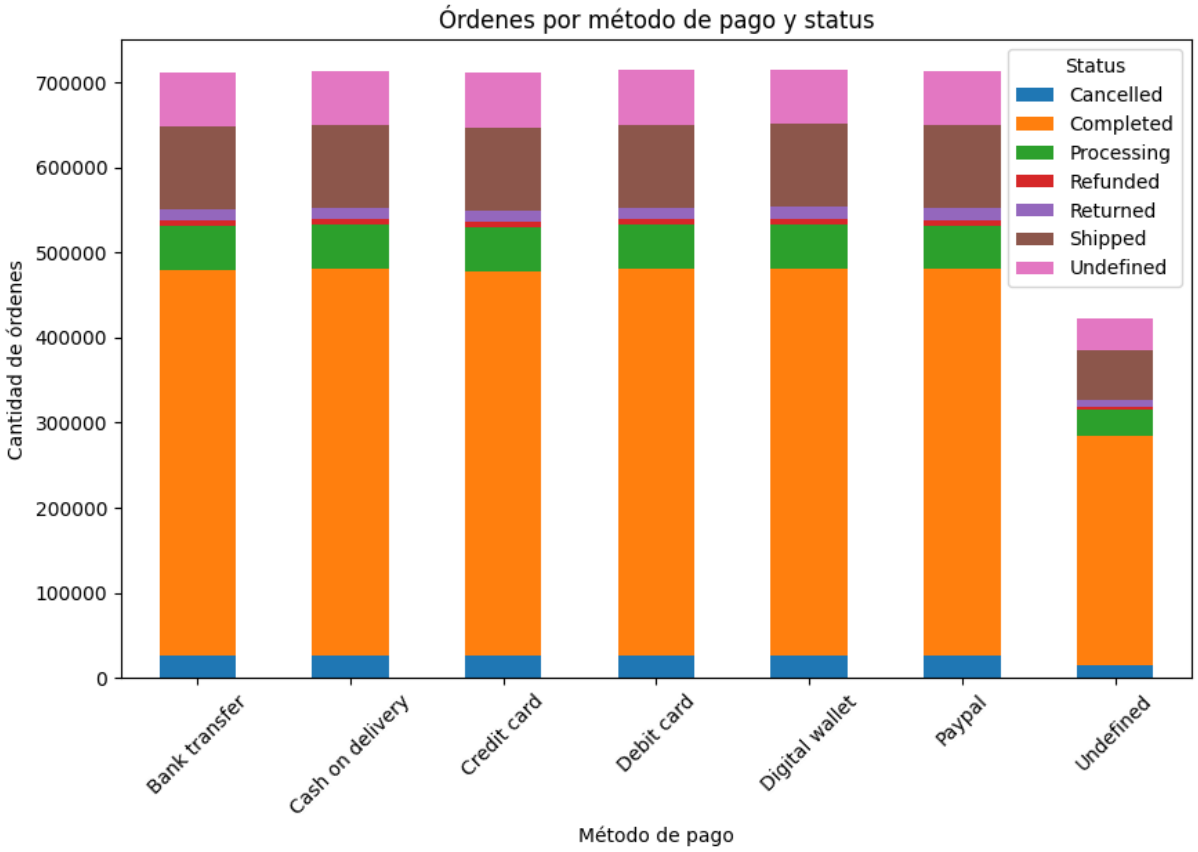


Figura 6: Cantidad de ordenes en base al método de pago y el estado de la orden

Conclusión

Podemos notar como no hay variaciones notorias al estado de la orden en los distintos métodos de pagos. Ademas tambien podemos observar como la cantidad de ordenes es casi igual en todos los metodos de pagos menos los indefinidos.

4. Clientes y pago

4.1. Órdenes Devueltas y Códigos Postales

Introducción

Se buscan los 5 códigos postales con mas devoluciones de ordenes. Ademas, se busca el cliente con mas devoluciones en las direcciones de estos códigos postales.

Hipótesis

Se decidió que los códigos postales a tener en cuenta son los de la sección de la dirección de envió siguiendo un formato determinado. Se utiliza esta la dirección de envió por simplicidad. Se espera poder ver si hay alguna clara diferencia

Código

```
df = dfs["orders.pkl"].copy()
df_clientes = dfs["customers.pkl"].copy()

refund_df = df[df["status"] == "Refunded"].copy()

refund_df["codigo_postal"] = (
    refund_df["shipping_address"]
    .str.split(",").str[1]
    .str.strip()
    .str.split().str[-1]
)

top_5_codigos = refund_df["codigo_postal"].value_counts().head(5)

refund_top5 = refund_df[refund_df["codigo_postal"].isin(
   top_5_codigos.index)]

clientes_frecuentes = refund_top5["customer_id"].mode()[0]

nombre_cliente = df_clientes.loc[
    df_clientes["customer_id"] == clientes_frecuentes, "
       first_name"
].values[0]

print("Top 5 códigos postales con más reembolsos:")
print(top_5_codigos)
```

```
print(f"\nEl cliente más frecuente en esas zonas es: {nombre_cliente}")
```

Resultados y visualizaciones

```
codigo_postal
70696    6
76449    4
59883    4
71463    4
90226    4
Name: count, dtype: int64

El nombre más frecuente entre clientes de esas direcciones es: Ashley
```

Conclusión

Podemos notar que no hay tantas devoluciones en algun codigo postal en particular por lo que no hay un comportamiento particular respecto a las devoluciones

4.2. Métodos de Pago y Segmento de Cliente

Introduccion

Se analiza que para cada tipo de pago y segmento de cliente, se analiza la suma y el promedio expresado como porcentaje, de clientes activos y de consentimiento de marketing.

Hipótesis

Se tienen en cuenta los nan y los indefinidos en los distintos tipos de datos como el segmento del cliente y el tipo de pago pues hay combinaciones de datos valiosos aunque no incluyan datos precisos por completo

Código

```
df_orders = dfs['orders.pkl'].copy()
df_customers = dfs['customers.pkl'].copy()

df_orders = df_orders[['customer_id', 'payment_method']]
df_customers = df_customers[['customer_id', 'customer_segment',
                             'marketing_consent','is_active']]

df_orders['payment_method'] = df_orders['payment_method'].fillna('no definido')
df_customers['customer_segment'] = df_customers['customer_segment'].fillna('no definido')
```

```
10
11 df_customers['customer_segment'] = df_customers['customer_segment
   '].replace('undefined', 'No definido')
12 df_orders['payment_method'] = df_orders['payment_method'].replace
   ('undefined', 'No definido')
13
14 df_orders['payment_method'] = df_orders['payment_method'].str.
   strip().str.lower().str.capitalize()
15 df_customers['customer_segment'] = df_customers['customer_segment
   '].str.strip().str.lower().str.capitalize()
16
17 df = df_orders.merge(df_customers, on='customer_id')
18
19 resumen = df.groupby(['payment_method', 'customer_segment']).agg(
20     total_clientes_activos=('is_active', 'sum'),
21     total_clientes_marketing=('marketing_consent', 'sum'),
22     cantidad_total_clientes=('customer_id', 'count')
23 ).reset_index()
24
25 resumen['porcentaje_clientes_activos'] = (resumen['
   total_clientes_activos'] / resumen['cantidad_total_clientes'] *
    100).round(2)
26 resumen['porcentaje_clientes_marketing'] = (resumen['
   total_clientes_marketing'] / resumen['cantidad_total_clientes']
    * 100).round(2)
27
28 print(resumen.to_string())
```

Resultados y visualizaciones

	Tipo de pago	Segmento de cliente	Clientes activos (total)	Clientes con consentimiento marketing (total)	Total de clientes	Clientes activos (%)	Clientes con consentimiento marketing (%)
0	Transferencia bancaria	Budget	114885	89153	127753	89.930000	69.790000
1	Transferencia bancaria	No definido	58775	45603	65325	89.970000	69.810000
2	Transferencia bancaria	Premium	116360	90546	129544	89.820000	69.900000
3	Transferencia bancaria	Regular	350102	273106	389287	89.930000	70.160000
4	Efectivo en entrega	Budget	114670	88842	127574	89.890000	69.640000
5	Efectivo en entrega	No definido	58901	45678	65498	89.930000	69.740000
6	Efectivo en entrega	Premium	117736	91760	130726	90.060000	70.190000
7	Efectivo en entrega	Regular	350185	273204	389270	89.960000	70.180000
8	Tarjeta de crédito	Budget	113910	88255	126745	89.870000	69.630000
9	Tarjeta de crédito	No definido	58729	45383	65306	89.930000	69.490000
10	Tarjeta de crédito	Premium	116702	91178	129896	89.840000	70.190000
11	Tarjeta de crédito	Regular	350004	273322	389018	89.970000	70.260000
12	Debit card	Budget	113869	88403	126824	89.790000	69.710000
13	Debit card	No definido	58792	45409	65333	89.990000	69.500000
14	Debit card	Premium	117625	91479	130993	89.790000	69.840000
15	Debit card	Regular	351591	273540	390574	90.020000	70.040000
16	Digital wallet	Budget	114641	89114	127566	89.870000	69.860000
17	Digital wallet	No definido	58542	45196	65053	89.990000	69.480000
18	Digital wallet	Premium	117553	91429	130895	89.810000	69.850000
19	Digital wallet	Regular	351015	274082	390561	89.870000	70.180000
20	No definido	Budget	68152	52689	75745	89.980000	69.560000
21	No definido	No definido	34851	27029	38798	89.830000	69.670000
22	No definido	Premium	69600	54181	77434	89.880000	69.970000
23	No definido	Regular	207565	161836	230845	89.920000	70.110000
24	Paypal	Budget	114113	88744	127116	89.770000	69.810000
25	Paypal	No definido	58814	45809	65423	89.900000	70.020000
26	Paypal	Premium	117467	91581	130655	89.910000	70.090000
27	Paypal	Regular	351110	274019	390243	89.970000	70.220000

Conclusión

Podemos notar que no hay mucha variación en la cantidad de clientes en base al método de pago. Ademas se puede observar como el consentimiento de marketing y los clientes activos no varían fuertemente ni por el segmento de cliente ni por el tipo de pago.

Por lo que podemos intuir que ni el método de pago ni el segmento de cliente afecta en sí el cliente está activo o si da su consentimiento para el marketing.

4.3. Distribución de tiempo de compras de clientes

Introducción

Se analiza la cantidad de días que hay entre las compras en promedio de los clientes que hicieron ordenes.

Hipótesis

Es importante tener en cuenta que no se tienen en cuenta datos que no tengan información sobre el cliente o sobre la fecha pues es el análisis necesita que estos datos sean certeros. Se espera poder diferenciar los clientes los cuales hacen pedidos mas seguido de los que no. También se distinguen a los clientes que solo hicieron un pedido, estos no se tienen en cuenta para ver la distribución temporal.

Código

```
df_orders = dfs['orders.pkl'].copy()
df_orders = df_orders[['order_id','order_date','customer_id']].
    dropna()
df_orders['order_date'] = pd.to_datetime(df_orders['order_date'],
    errors='coerce')

df_orders = df_orders.sort_values(['customer_id', 'order_date'])
df_orders['days_between'] = df_orders.groupby('customer_id')['
    order_date'].diff().dt.days

df_summary = df_orders.groupby('customer_id').agg({
    'order_id': 'count',
    'days_between': 'mean'
}).round(1).reset_index()

df_summary.columns = ['customer_id', 'order_count', '
    avg_days_between']

def classify_detailed(row):
    order_count = row['order_count']
    avg_days = row['avg_days_between']

    if order_count == 1:
        return 'Una sola compra'
    elif pd.isna(avg_days):
        return 'Sin datos válidos'
    elif avg_days == 0:
        return 'Mismo día (0 días)'
    elif avg_days <= 10:
        return 'Muy frecuente (1-10 días)'
```

```
    elif avg_days <= 20:
        return 'Frecuente (11-20 días)'
    elif avg_days <= 30:
        return 'Regular (21-30 días)'
    elif avg_days <= 40:
        return 'Ocasional (31-40 días)'
    elif avg_days <= 50:
        return 'Ocasional (41-50 días)'
    elif avg_days <= 60:
        return 'Poco frecuente (51-60 días)'
    return 'Muy esporádico (>60 días)'

df_summary['segment'] = df_summary.apply(classify_detailed, axis=1)

results = df_summary['segment'].value_counts()
for segment, count in results.items():
    pct = (count / len(df_summary)) * 100
    print(f"{segment}: {count:,} ({pct:.1f}%)")

recurring = df_summary[df_summary['order_count'] > 1]['avg_days_between']
print(f"\nRecurrentes - Promedio: {recurring.mean():.1f}, Mediana: {recurring.median():.1f}")
```

```
plt.figure(figsize=(10,5))
sb.kdeplot(df_summary["avg_days_between"], fill=True, color="steelblue", alpha=0.6)
plt.title("Distribución de días promedio entre compras", fontsize=14)
plt.xlabel("Promedio de días entre compras")
plt.ylabel("Densidad de clientes")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

Resultados y visualizaciones

```
Muy frecuente (1-10 días): 37,997 (38.0%)
Frecuente (11-20 días): 37,387 (37.4%)
Regular (21-30 días): 21,273 (21.3%)
Ocasional (31-40 días): 1,994 (2.0%)
Mismo día (0 días): 1,235 (1.2%)
Ocasional (41-50 días): 104 (0.1%)
Poco frecuente (51-60 días): 2 (0.0%)

Recurrentes - Promedio: 13.3, Mediana: 12.9
```

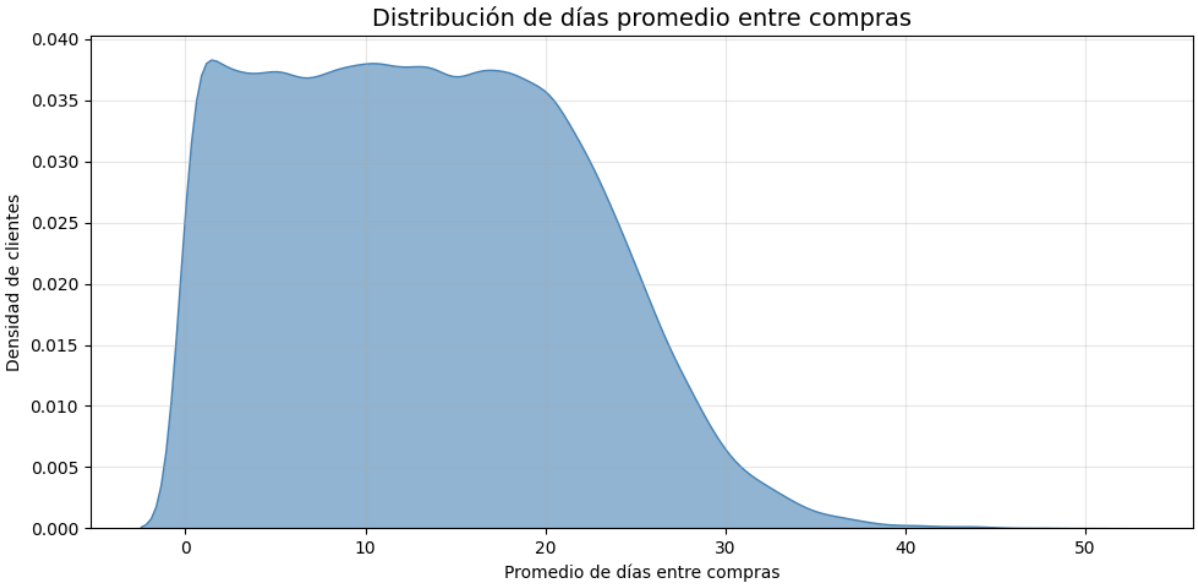


Figura 7: Distribución de días promedio entre compras

Conclusión

Podemos notar que la mayoría de los clientes se toman varios días en promedio para hacer otros días. Ademas podemos notar que hay muy pocos clientes que tardan mucho tiempo en pedir de nuevo.

5. Productos y reviews

5.1. Productos con "Stuff" en la Descripción

Introducción

Se analizará los productos que contengan la palabra stuff en su descripción. Después se agrupa por marca y se calculara su peso total en su inventario. Además, se verá cuáles son las 5 marcas con más peso en el inventario.

Hipótesis

Se descartan los datos que tengan la marca como indefinida o que no tenga información sobre ella pues queremos hacer un análisis de las marcas. Ademas, cualquier producto de; que no hay información sobre su peso o su stock, tendrá un peso total igual a 0 para no usar datos que no tenemos la información necesaria para datos certeros.

Código

```
1 df_products = dfs['products.pkl'].copy()
2
3 df_stuff = df_products[df_products['description'].str.contains('
    stuff', case=False, na=False)].copy()
4
```

```
df_stuff['weight_kg'] = df_stuff['weight_kg'].fillna(0)
df_stuff['stock_quantity'] = df_stuff['stock_quantity'].fillna(0)
df_stuff['brand'] = df_stuff['brand'].fillna("Desconocida")
df_stuff['brand'] = df_stuff['brand'].replace(["undefined", "
    Undefined"], "Desconocida")

df_stuff = df_stuff[df_stuff['brand'] != "Desconocida"].copy()
df_stuff['peso_total'] = df_stuff['weight_kg'] * df_stuff['
    stock_quantity']

top_5_pesadas = df_stuff.groupby('brand')['peso_total'].sum().
    nlargest(5)

print(top_5_pesadas)
```

Resultados y visualizaciones

```
      Marca  Peso total
0        3M  2037214.18
1    Adidas  1783357.68
2   Wayfair  1508354.30
3    Hasbro  1420969.47
4      Nike  1408352.56
```

Conclusión

Las marcas identificadas no parecen tener un comportamiento en específico. Nada más podemos notar que dos de las cinco marcas son de ropa.

5.2. Distribución de ratings

Introducción

Se hace un análisis simple de la distribución del rating de las reviews.

Hipótesis

No se tienen en cuenta reviews que no tenga un rating. Se espera ver si la calidad de lo que se vende es bien recibida por los clientes.

Código

```
df_reviews = df_reviews[['product_id','rating']]
df_reviews['rating'] = pd.to_numeric(df_reviews['rating'], errors
    ='coerce')
df_reviews = df_reviews.dropna(subset=['rating'])

print(f"Total reviews: {len(df_reviews):,}")
```

```
print(f"Rating promedio: {df_reviews['rating'].mean():.2f}")
```

```
plt.figure(figsize=(8, 6))
plt.hist(df_reviews['rating'],
         bins=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5],
         edgecolor='black',
         alpha=0.7)
plt.xlabel('Rating)')
plt.ylabel('Número de Reviews')
plt.title('Distribución de Ratings')
plt.xticks([1, 2, 3, 4, 5])
plt.show()
```

Resultados y visualizaciones

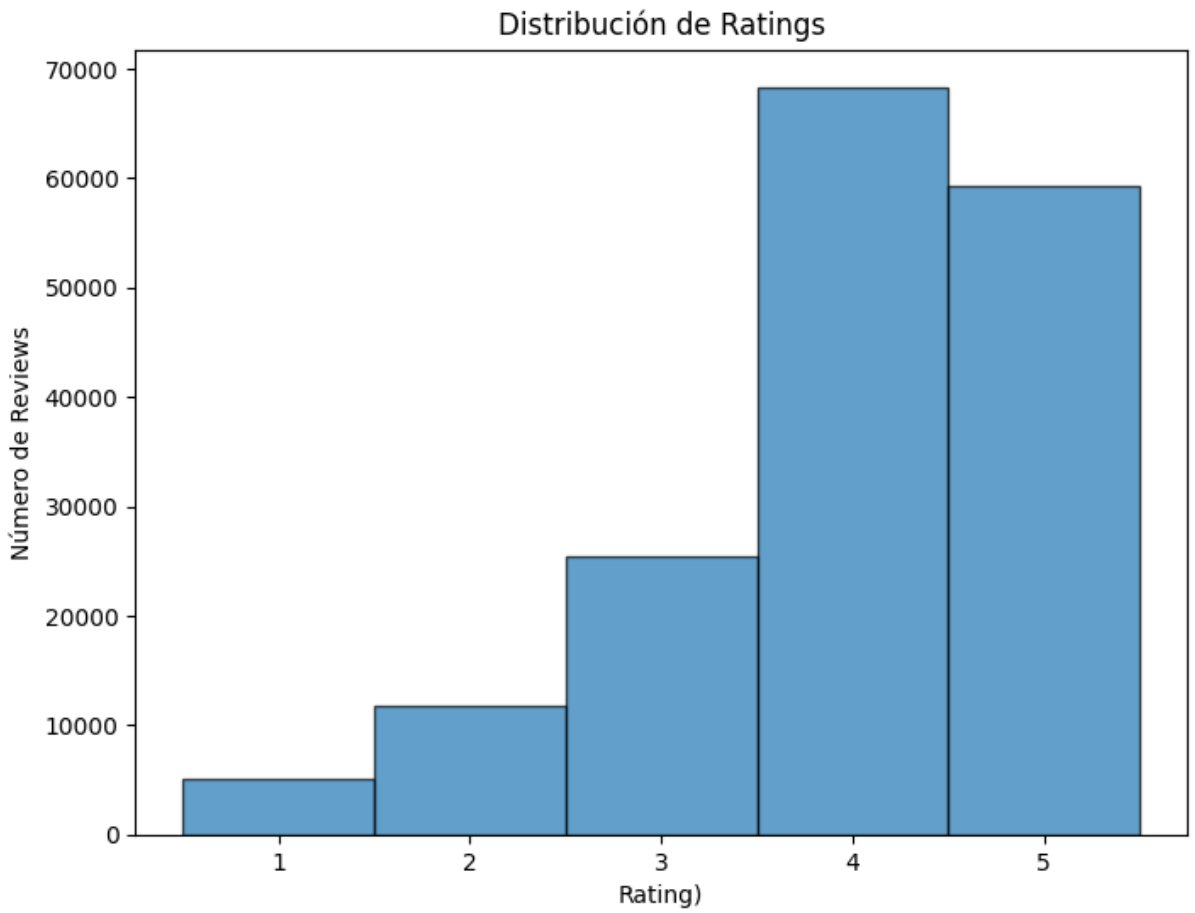


Figura 8: Cantidad de órdenes por estado y mes

```
Total reviews: 170,000
Rating promedio: 3.97
```

Conclusión

Podemos notar que en general hay un buen rating y teniendo un buen promedio. Ademas hay pocas reviews con bajo rating.

5.3. Reviews a lo largo del tiempo

Introducción

Se analiza el promedio de las reviews a lo largo del tiempo. El promedio se toma en cuenta en base a cada semana.

Hipótesis

Cualquier review sin fecha de creación es descartada en el análisis pues la fecha es clave para esto. Es importante aclarar que el promedio es por cada semana para que haya un promedio mas similar por cada semana pues puede haber mas variaciones en un dia que en una semana. Se espera poder observar si las reviews muestran un comportamiento continuo y sin mucho movimiento para asi poder analizar si los productos que se venden son de una calidad parecida todo el tiempo.

Código

```
df_reviews=dfs['reviews.pkl'].copy()

df_reviews['created_at'] = pd.to_datetime(df_reviews['created_at'],errors='coerce')

df_reviews = df_reviews.dropna(subset=['created_at'])

df_reviews['review_week'] = df_reviews['created_at'].dt.to_period('W').apply(lambda r: r.start_time)

df_reviews=df_reviews[['review_id','review_week','rating']]

df_reviews=df_reviews.groupby(['review_week']).agg({'rating':'mean'}).reset_index()
```

```
plt.figure(figsize=(12,5))
sb.lineplot(data=df_reviews, x='review_week', y='rating', marker='o')
plt.xticks(rotation=45)
plt.title("Promedio semanal de rating")
plt.ylabel("Rating promedio")
plt.xlabel("Semana")
plt.grid(True)

# Descomentar para verlo mas general
#plt.ylim(0, 5)

```

```
12 plt.show()
```

Resultados y visualizaciones

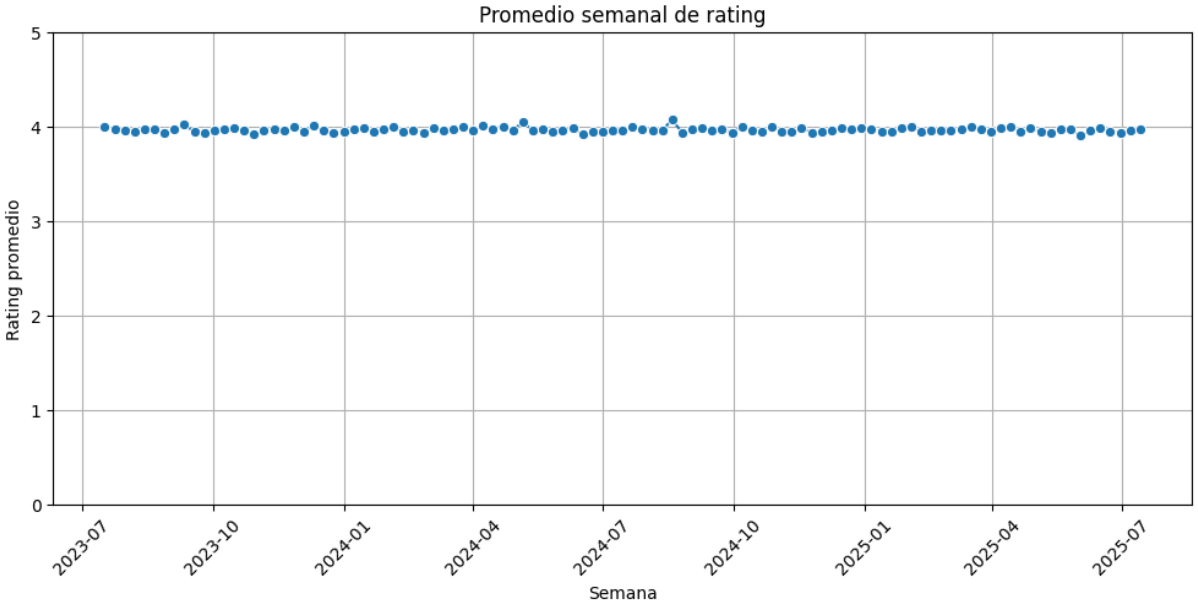


Figura 9: Rating promedio a traves del tiempo

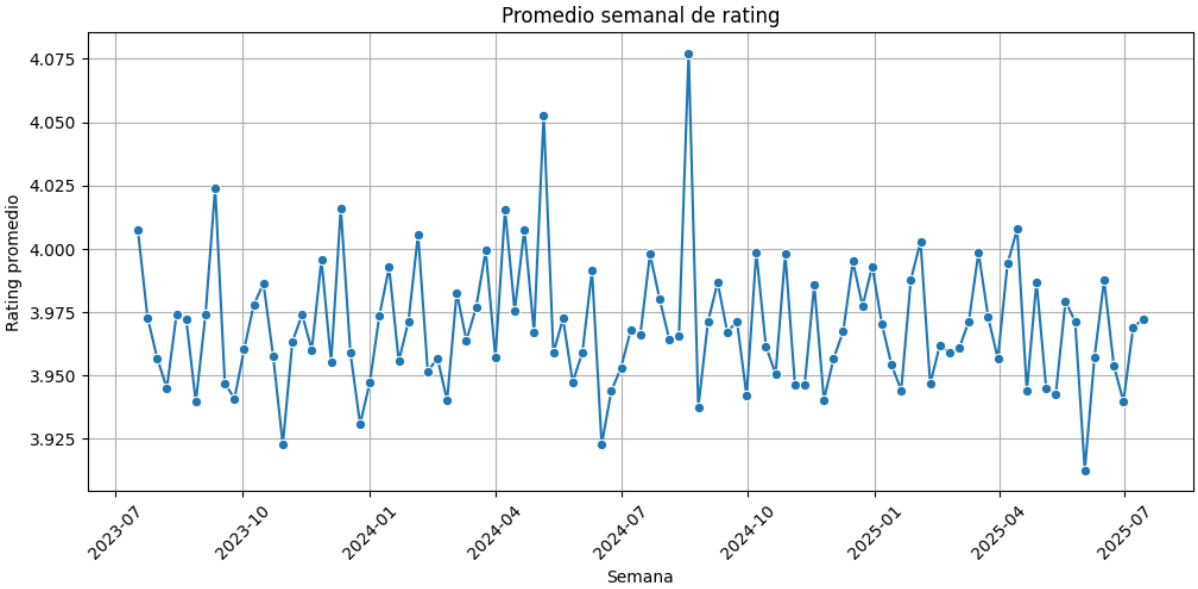


Figura 10: Rating promedio a traves del tiempo mas detallado

Conclusión

Se puede observar como la variación del promedio semanal es bastante bajo demostrando que las reviews no varían mucho en base al tiempo. Se mostraron dos gráficos para mostrar en mas detalle pues en el segundo gráfico parece que varia mucho pero no es asi

5.4. Palabras mas utilizadas en las reviews

Introducción

Se hace un análisis simple de las palabras mas utilizadas en la descripcion y el titulo de las reviews

Hipótesis

Se espera poder ver cuales son las palabras mas utilizadas en las descripciones y los titulo de las reviews. Se descartan palabras vacías que se utilizan en oraciones comunes pero que no tienen un significado. Esto se hace pues estas palabras van a ser muy utilizadas y no van a decir mucho de los datos.

Código

```
df_reviews = dfs['reviews.pkl'].copy()

texto_completo = ' '.join(df_reviews['title'].fillna('').astype(
    str))
#descomentar la proxima linea para ver la nube con los
    comentarios en vez de los titulos
#texto_completo = ' '.join(df_reviews['comment'].fillna('').
    astype(str))

wordcloud = WordCloud(
    width=1200,
    height=600,
    background_color='white',
    max_words=100,
    colormap='viridis',
    stopwords=set(['the', 'and', 'or', 'but', 'in', 'on', 'at', '
        to', 'for', 'of', 'with', 'by','a','it'])
).generate(texto_completo.lower())

plt.figure(figsize=(15, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Palabras más frecuentes en las Reviews', fontsize=16,
    pad=20)
plt.show()
```

Resultados y visualizaciones

Figura 11: Wordcloud de los titulos de la reviews

Figura 12: Wordcloud de los comentarios de la reviews

Conclusión

Podemos ver como undefined esta muy grande, esto muestra que hay muchos datos indefinidos respecto a los títulos de las reviews. En las descripciones podemos identificar palabras como new, la cual tiene sentido en una review de un producto. También tenemos a mr, la cual no dice mucho sobre nuestro dominio.

6. Inventario y rentabilidad

6.1. Robos y roturas en el inventario

Introducción

Se hace un análisis de los robos y las roturas del inventario de los distintos productos. Ademas se analiza si se rompen o roban mas productos en base a su categoría.

Hipótesis

Se tienen en cuenta los robos/roturas de productos que no tengan categoría definida ya que puede seguir siendo información valiosa a tener en cuenta para solucionar estos problemas. Se espera poder ver que categorías sufren mas de estos problemas. Notemos que este análisis se hará sobre la cantidad de incidentes, no de la cantidad de productos involucrada ya que estamos comparando distintos productos y la cantidad puede variar mucho.

Código

```
df_logs = dfs['inventory_logs.pkl'].copy()
df_products = dfs['products.pkl'].copy()
df_categorias = dfs['categories.pkl'].copy()

df_logs['movement_type'] = df_logs['movement_type'].str.upper().
    str.strip()
df_logs['reason'] = df_logs['reason'].fillna('undefined').str.
    lower().str.strip()
df_logs['quantity_change'] = pd.to_numeric(df_logs['
    quantity_change'], errors='coerce')

df_categorias['parent_category'] = df_categorias['parent_category
    '].fillna('Undefined').str.strip().str.lower().str.capitalize()
df_categorias['category_name'] =df_categorias['category_name'].
    fillna('Undefined').str.strip().str.lower().str.capitalize()

theft_damage = df_logs[df_logs['reason'].isin(['theft', 'damage'
    ])].copy()
print(f"Total incidentes de theft/damage: {len(theft_damage):,}")

df_theft_analysis = (
    theft_damage
    .merge(df_products[['product_id', 'category_id']], on='
        product_id')
    .merge(df_categorias[['category_id', 'category_name', '
        parent_category']], on='category_id')
)

print("\n=== INCIDENTES POR CATEGORÍA PADRE TOP 10 ===")

```

```python
parent_summary = df_theft_analysis.groupby(['parent_category', 'reason']).agg({
    'quantity_change': 'count',  # Solo incidentes
    'product_id': 'nunique'
}).round(2)

parent_summary.columns = ['incidents', 'products_affected']
parent_summary = parent_summary.reset_index()

parent_pivot = parent_summary.pivot_table(
    index='parent_category',
    columns='reason',
    values='incidents',
    fill_value=0
).sort_values(['theft', 'damage'], ascending=False)

print(parent_pivot.head(10))

# Resumen final
theft_incidents = parent_summary[parent_summary['reason'] == 'theft']['incidents'].sum()
damage_incidents = parent_summary[parent_summary['reason'] == 'damage']['incidents'].sum()
theft_categories = parent_summary[parent_summary['reason'] == 'theft']['parent_category'].nunique()
damage_categories = parent_summary[parent_summary['reason'] == 'damage']['parent_category'].nunique()

print(f"\n=== RESUMEN ===")
print(f"Total incidentes de robo: {theft_incidents}")
print(f"Total incidentes de daño: {damage_incidents}")
print(f"Total incidentes: {theft_incidents + damage_incidents}")
print(f"Categorías padre afectadas por robos: {theft_categories}")
print(f"Categorías padre afectadas por daños: {damage_categories}")
```

Resultados y visualizaciones

```
Total incidentes de theft/damage: 103,921

=== INCIDENTES POR CATEGORÍA PADRE TOP 10 ===
reason                   damage   theft
parent_category
Undefined                4548.0  4459.0
Furniture                2006.0  1984.0
Kitchen & dining         1719.0  1673.0
Grocery & gourmet food   1497.0  1557.0
Tools & hardware         1392.0  1553.0
Games & virtual goods    1511.0  1536.0
```

```
Shoes                        1470.0   1533.0
Industrial & scientific      1531.0   1518.0
Music & movies               1502.0   1488.0
Pet care                     1460.0   1487.0

=== RESUMEN ===
Total incidentes de robo: 44147
Total incidentes de daño: 44071
Total incidentes: 88218
Categorías padre afectadas por robos: 31
Categorías padre afectadas por daños: 31
```

Conclusión

Podemos notar que los productos que mas roturas/daños tiene son productos que no tienen una categoría padre. Después el resto de las categorías varían bastante. En lineas generales cada categoría se le rompen y le roban maso menos la misma cantidad veces.

6.2. Profit por categoría padre

Introducción

Se analiza el profit promedio de todos los productos dentro de una categoría padre

Hipótesis

No se tienen en cuenta datos que no hay información del precio, el costo o su categoria asociada para no generar promedios falsos. Se espera ver si algunas categorías generan mas rendimientos que otras

Código

```
df_products = dfs['products.pkl'].copy()
df_categorias = dfs['categories.pkl'].copy()

df_products=df_products[['product_id','category_id','price','cost
    ']]

df_products=df_products.dropna()

df_categorias['parent_category'] = df_categorias['parent_category
    '].fillna('Undefined').str.strip().str.lower().str.capitalize()

df_merge=df_products.merge(df_categorias, left_on='category_id',
    right_on='category_id')

df_merge['profit_multiplier'] = df_merge['price'] / df_merge['
    cost']

```

```
df_merge=df_merge[['product_id','parent_category','
    profit_multiplier']]

df_merge=df_merge.groupby(['parent_category']).agg({'
    profit_multiplier':'mean'}).reset_index()
```

```
df_merge.sort_values('profit_multiplier').plot(x='parent_category
    ', y='profit_multiplier', kind='bar', figsize=(12, 6), legend=
    False)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Resultados y visualizaciones

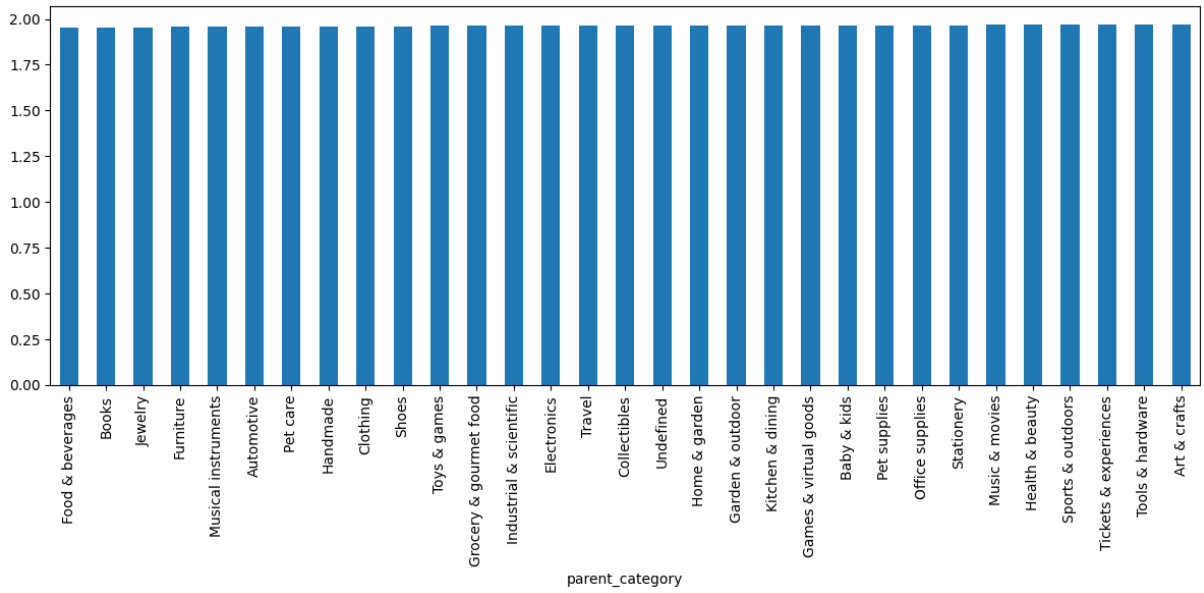


Figura 13: Distribución de días promedio entre compras

Conclusión

Podemos notar que no hay una gran diferencia del multiplicador de ganancias en base al costo y la ganancia del promedio de cada categoría.

6.3. Dividiendo los ingresos en los costos y las ganancias

Introducción

Se hace un análisis para poder separar las ganancias y los costos en base a cada categoría para poder que genera mas rendimientos esta generando en este lapso de tiempo.

Hipótesis

Se espera ver que categoría es la que mas rendimientos esta generando en el ultimo tiempo. Se tienen en cuenta las ventas que tengan los datos de los costos, cantidad vendidas e ingresos totales para que sea mas claro el análisis.

Código

```
df_order_items = dfs['order_items.pkl'].copy()
df_products = dfs['products.pkl'].copy()
df_categories = dfs['categories.pkl'].copy()
df_merged.dropna(subset=['line_total','cost','quantity'],inplace=
    True)

df_merged = df_order_items.merge(
    df_products[['product_id', 'category_id', 'cost']],
    on='product_id'
).merge(
    df_categories[['category_id', 'category_name', '
        parent_category']],
    on='category_id'
)

df_merged['revenue'] = df_merged['line_total']
df_merged['total_cost'] = df_merged['cost'] * df_merged['quantity
    ']
df_merged['profit'] = df_merged['revenue'] - df_merged['
    total_cost']

df_merged['parent_category']=df_merged['parent_category'].str.
    strip().str.lower().str.capitalize()
df_merged['parent_category'].fillna('Undefined', inplace=True)

parent_summary = df_merged.groupby('parent_category').agg({
    'revenue': 'sum',
    'total_cost': 'sum',
    'profit': 'sum'
}).reset_index()
```

```
top_parents = parent_summary['parent_category'].tolist()

labels = []
sources = []
targets = []
values = []

# Nodos: Ingresos -> Categorías Padre -> Costos/Ganancias
labels.extend(['INGRESOS'])
labels.extend([f'{parent}' for parent in top_parents])
labels.extend(['COSTOS TOTALES', 'GANANCIA NETA'])

```

```
# Flujos: Ingresos -> TODAS las Categorías Padre
for i, parent in enumerate(top_parents):
    revenue = parent_summary[parent_summary['parent_category'] ==
        parent]['revenue'].iloc[0]
    sources.append(0)  # Desde INGRESOS
    targets.append(i + 1)  # Hacia categoría padre
    values.append(revenue)

# Flujos: TODAS las Categorías Padre -> Costos
total_costs_idx = len(labels) - 2
for i, parent in enumerate(top_parents):
    cost = parent_summary[parent_summary['parent_category'] ==
       parent]['total_cost'].iloc[0]
    sources.append(i + 1)  # Desde categoría padre
    targets.append(total_costs_idx)  # Hacia COSTOS
    values.append(cost)

# Flujos: TODAS las Categorías Padre -> Ganancias
profit_idx = len(labels) - 1
for i, parent in enumerate(top_parents):
    profit = parent_summary[parent_summary['parent_category'] ==
       parent]['profit'].iloc[0]
    if profit > 0:  # Solo mostrar ganancias positivas
        sources.append(i + 1)  # Desde categoría padre
        targets.append(profit_idx)  # Hacia GANANCIA
        values.append(profit)

category_colors = (px.colors.qualitative.Set1 + px.colors.
   qualitative.Set2 +
                   px.colors.qualitative.Set3 + px.colors.
                      qualitative.Pastel1)[:len(top_parents)]

node_colors = (
    ['#1f77b4'] +  # Ingresos (azul)
    category_colors +  # Categorías con colores distintos
    ['#ff4444', '#00aa00']  # Costos (rojo), Ganancia (verde)
)

link_colors = []
for s, t in zip(sources, targets):
    if t == total_costs_idx:  # Hacia costos
        link_colors.append('rgba(255,68,68,0.4)')
    elif t == profit_idx:  # Hacia ganancia
        link_colors.append('rgba(0,170,0,0.4)')
    else:  # Hacia categorías
        link_colors.append('rgba(31,119,180,0.4)')

# Crear Sankey
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=20,
```

```
        thickness=25,
        line=dict(color="black", width=0.8),
        label=labels,
        color=node_colors
    ),
    link=dict(
        source=sources,
        target=targets,
        value=values,
        color=link_colors
    )
)])

fig.update_layout(
    title_text="Flujo de Dinero: Ingresos -> Todas las Categorías
        -> Costos/Ganancias",
    font_size=10,
    width=1400,
    height=900
)

fig.show()
```

Resultados y visualizaciones

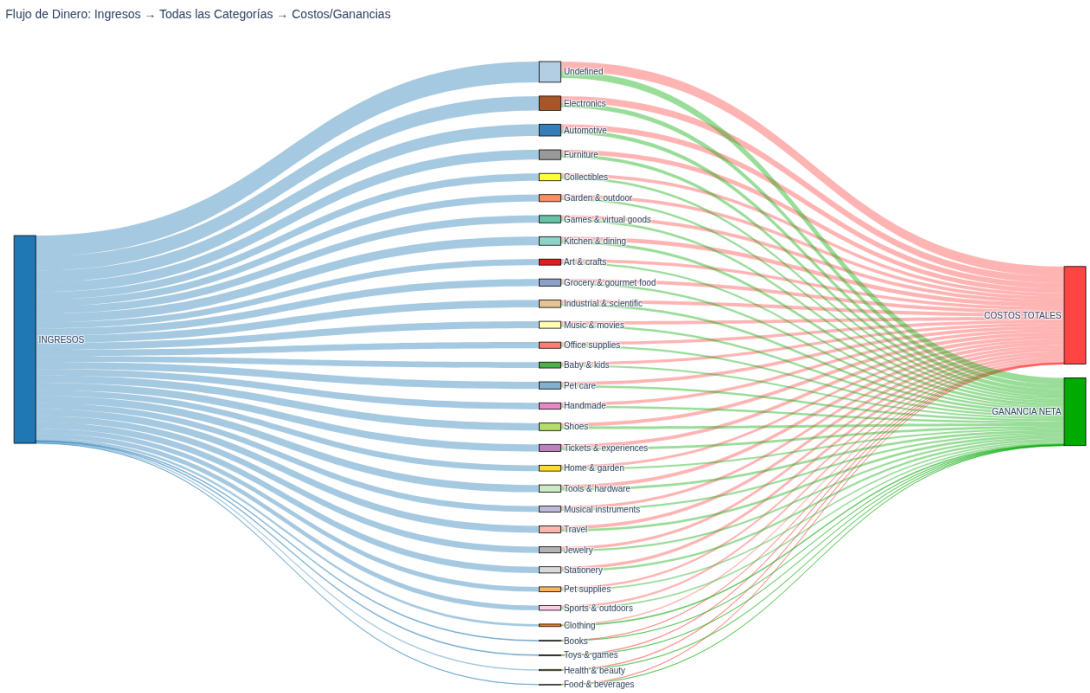


Figura 14: Separacion de las ganancias y costos de los ingresos

Conclusión

Podemos notar que hay categorias que mueven mas dinero que otras, solo la de indefinidos mueve mucho mas que el resto. Aca podemos ver que de todo lo que ingresa, se va mas dinero al de los costos que a los ingresos. Igualmente podemos ver que hay ganancias en el negocio.

7. Conclusiones generales

En conclusión, el análisis de los datos revela un comportamiento general consistente, sin variaciones destacables. No obstante, se identificó un desfase en los identificadores de las órdenes entre los conjuntos de datos de órdenes y de ordenes de items, lo que limitó la posibilidad de establecer relaciones directas entre productos y clientes. Esta inconsistencia impide realizar inferencias más profundas sin introducir suposiciones que podrían ser incorrectas. También hay mucho datos que contienen datos indefinidos o no contiene datos, lo cual afecta a una análisis con mas información.

Trabajo practico 1: Pandas Bautista Boeri 110898

```
# Instalar gdown (si no está instalado)
!pip install gdown

import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
import plotly.express as px
import plotly.graph_objects as go

# Descargar todos los archivos de la carpeta pública
carpeta_drive = "https://drive.google.com/drive/folders/1UXAAJ-XgEe5F89U03eNdQ5NBcwzP78A2"
!gdown --folder "{carpeta_drive}"

# Detectar automáticamente la carpeta creada
contenido = os.listdir("/content")
carpeta_descargada = [c for c in contenido if os.path.isdir(os.path.join("/content", c))][0]

ruta_guardado = "/content/datasets_pkl"

# Listar archivos .pkl en esa carpeta
archivos_pkl = [f for f in os.listdir(ruta_guardado) if f.endswith(".pkl")]

# Crear diccionario {nombre_archivo: DataFrame}
dfs = {f: pd.read_pickle(os.path.join(ruta_guardado, f)) for f in archivos_pkl}

print(f"✅ Se cargaron {len(dfs)} DataFrames")
print("Archivos cargados:", list(dfs.keys()))
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.12/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from gdown) (4.13.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from gdown) (3.19.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.12/dist-packages (from gdown) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from gdown) (4.67.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->gdown) (
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests[sock
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gd
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gd
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.12/dist-packages (from requests[socks]
Retrieving folder contents
Processing file 1x8hlH5GyqD89YfAdKZ3cYvXmXBojWc4J categories.pkl
Processing file 1Gf21XJoj7HbnLcUMslUIyWefE0p1-tkX customers.pkl
Processing file 1njEPh1dPRiJWNjlAXQc-SrpG87sLCl_N inventory_logs.pkl
Processing file 1lM6RVI5aoEuyL3gUkWrQTL1r5MEWczmw order_items.pkl
Processing file 16O0Ohly99MnkXchEaANXe0_mMnT9b2tr orders.pkl
Processing file 1YV6x7BHM23Ykcm2rKEiSuCxd4gbZmS1J products.pkl
Processing file 19xSuuXS1-hwi_MtNIunCaVoQQqxSdgFK reviews.pkl
Retrieving folder contents completed
Building directory structure
Building directory structure completed
Downloading...
From: https://drive.google.com/uc?id=1x8hlH5GyqD89YfAdKZ3cYvXmXBojWc4J
To: /content/datasets_pkl/categories.pkl
100% 11.9k/11.9k [00:00<00:00, 32.9MB/s]
Downloading...
From (original): https://drive.google.com/uc?id=1Gf21XJoj7HbnLcUMslUIyWefE0p1-tkX
From (redirected): https://drive.google.com/uc?id=1Gf21XJoj7HbnLcUMslUIyWefE0p1-tkX&confirm=t&uuid=49d76775-0a7e-4577-
To: /content/datasets_pkl/customers.pkl
100% 107M/107M [00:00<00:00, 149MB/s]
Downloading...
From: https://drive.google.com/uc?id=1njEPh1dPRiJWNjlAXQc-SrpG87sLCl_N
To: /content/datasets_pkl/inventory_logs.pkl
100% 37.0M/37.0M [00:00<00:00, 182MB/s]
Downloading...
From: https://drive.google.com/uc?id=1lM6RVI5aoEuyL3gUkWrQTL1r5MEWczmw
To: /content/datasets_pkl/order_items.pkl
100% 18.3M/18.3M [00:00<00:00, 174MB/s]
Downloading...
From (original): https://drive.google.com/uc?id=16O0Ohly99MnkXchEaANXe0_mMnT9b2tr
From (redirected): https://drive.google.com/uc?id=16O0Ohly99MnkXchEaANXe0_mMnT9b2tr&confirm=t&uuid=11fa7df2-281c-4d60-
To: /content/datasets_pkl/orders.pkl
100% 1.16G/1.16G [00:33<00:00, 35.0MB/s]
Downloading...
```

```
From (original): https://drive.google.com/uc?id=1YV6x7BHM23Ykcm2rKEiSuCxd4gbZmS1J
From (redirected): https://drive.google.com/uc?id=1YV6x7BHM23Ykcm2rKEiSuCxd4gbZmS1J&confirm=t&uuid=7f40e48c-968d-4dce-
To: /content/datasets_pkl/products.pkl
100% 213M/213M [00:05<00:00, 35.6MB/s]
Downloading...
From: https://drive.google.com/uc?id=19xSuuXS1-hwi_MtNIunCaVoQQgxSdgFK
To: /content/datasets_pkl/reviews.pkl
100% 105M/105M [00:04<00:00, 23.5MB/s]
Download completed
✅ Se cargaron 7 DataFrames
Archivos cargados: ['order_items.pkl', 'categories.pkl', 'reviews.pkl', 'products.pkl', 'orders.pkl', 'customers.pkl',
```

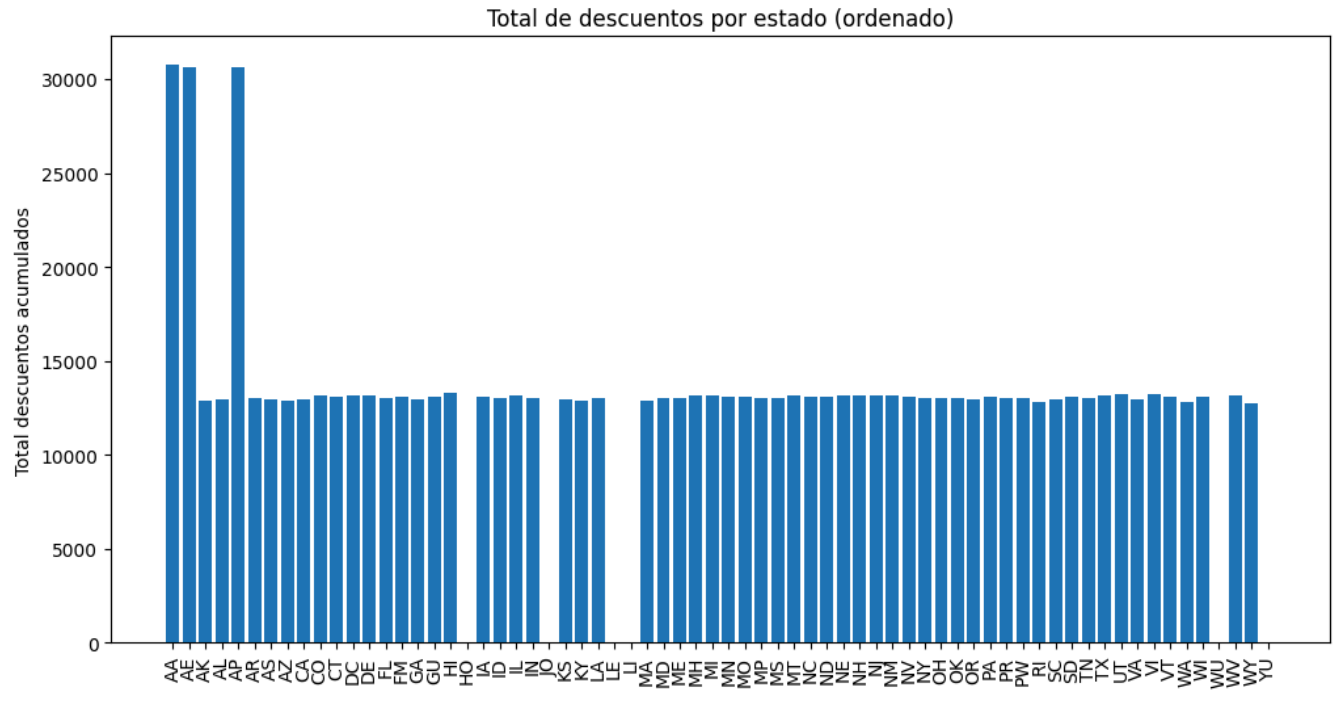
Ordenes y descuentos

Descuentos por Estado

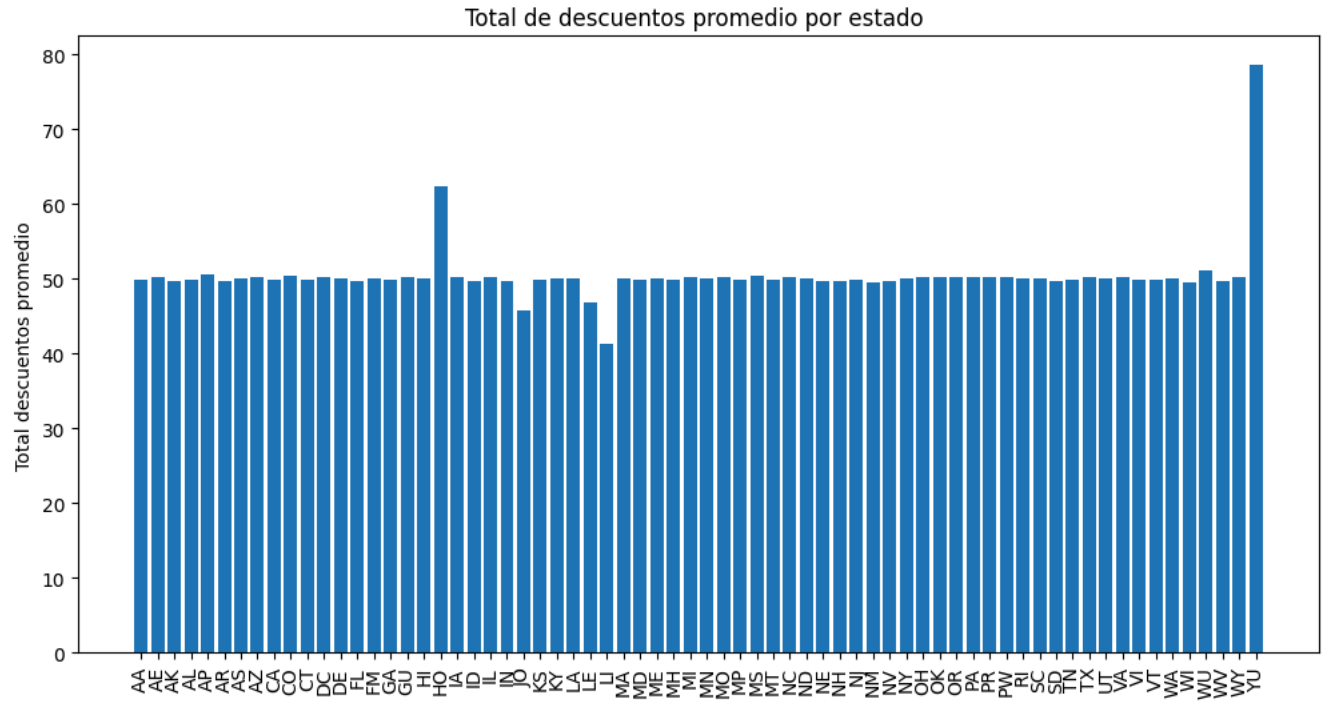
Analisis de cantidad total de descuento y del promedio de estos en base a cada estado. Cualquiera direccion que no contenga el estado en el formato AA (2 letras en mayusculas) no sera tomada en cuenta en el analisis

```
df = dfs['orders.pkl'].copy()

df = df[['shipping_address', 'discount_amount']]

df["estado"] = df["shipping_address"].str.extract(r"\b([A-Z]{2})\b")
df['discount_amount'] = df['discount_amount'].fillna(0)

df = df[df['discount_amount']>0]

resumen = df.groupby("estado").agg(
    total=("discount_amount", "count"),
    descuento_promedio=("discount_amount", "mean")
)
```

```
plt.figure(figsize=(12,6))
plt.bar(resumen.index, resumen['total'])
plt.xticks(rotation=90)
plt.ylabel("Total descuentos acumulados")
plt.title("Total de descuentos por estado (ordenado)")
plt.show()
```



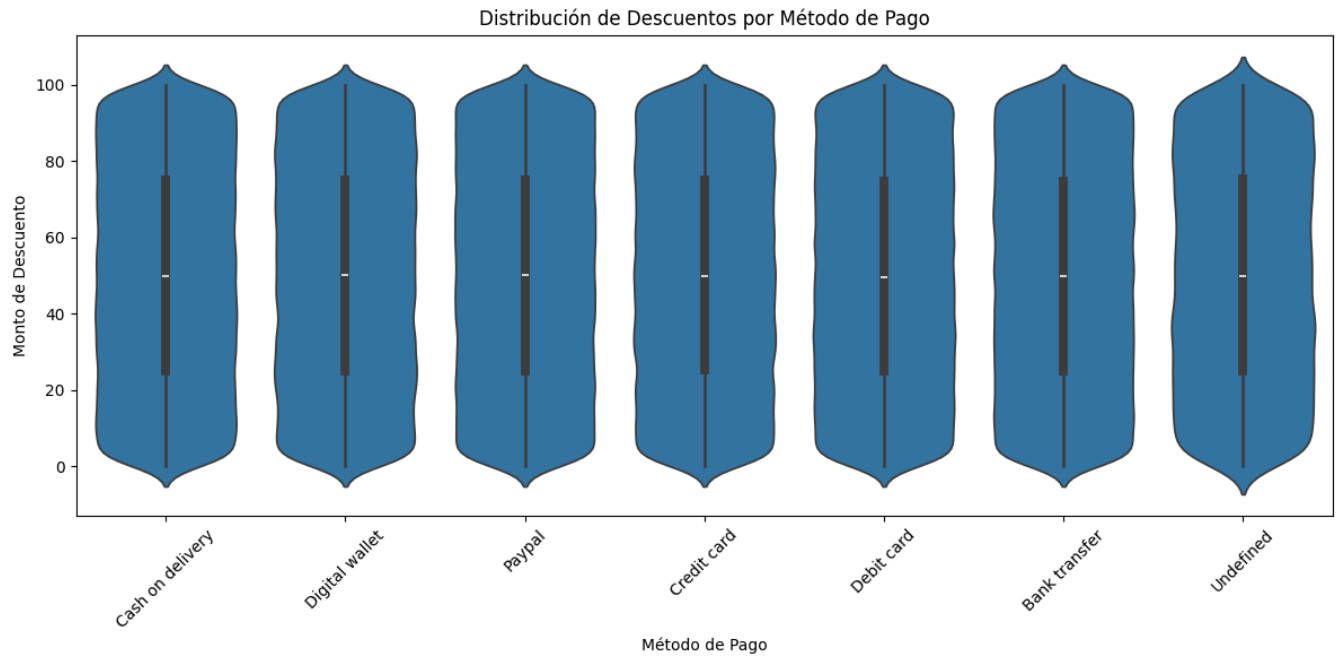
```
plt.figure(figsize=(12,6))
plt.bar(resumen.index, resumen['descuento_promedio'])
plt.xticks(rotation=90)
plt.ylabel("Total descuentos promedio")
plt.title("Total de descuentos promedio por estado ")
plt.show()
```



Distribucion de descuentos en base al metodo de pago

Comparacion de la cantidad descuento en base al metodo de pago. Solo son tenidas en cuenta las ordenes con descuento

```
df_orders = dfs['orders.pkl'].copy()
df_orders = df_orders[['order_id', 'payment_method', 'discount_amount']]
df_orders = df_orders[df_orders['discount_amount']>0]


df_orders = df_orders[['order_id', 'payment_method', 'discount_amount']]

df_orders.loc[:, 'payment_method'] = df_orders['payment_method'].str.strip().str.lower().str.capitalize()

plt.figure(figsize=(12,6))

sb.violinplot(
    data=df_orders,
    x='payment_method',
    y='discount_amount',
)

plt.xticks(rotation=45)
plt.xlabel("Método de Pago")
plt.ylabel("Monto de Descuento")
plt.title("Distribución de Descuentos por Método de Pago")
plt.tight_layout()
plt.show()
```



Ordenes por mes en categorias padres

Comparacion de cantidad de ventas por mes en las top 15 categorias padres de productos. No es util, order id es distinto en order items comparado a la info de ordenes

```
df_categorias = dfs['categories.pkl'].copy()
df_products= dfs['products.pkl'].copy()
df_orders = dfs['orders.pkl'].copy()
df_order_items = dfs['order_items.pkl'].copy()

df_orders['created_at'] = pd.to_datetime(df_orders['created_at'], format='mixed', errors='coerce')
df_orders['mes'] = df_orders['created_at'].dt.month


df_merge_categoria = df_categorias.merge(df_products, left_on='category_id', right_on='category_id')


df_merge_order=df_orders.merge(df_order_items, left_on='order_id', right_on='order_id')

df_merge_order_categoria=df_merge_order.merge(df_merge_categoria, left_on='product_id', right_on='product_id')

df_merge_order
```

	Unnamed: 0_x	order_id	customer_id	order_date	status	payment_method	shipping_address	billing_address	discount_amount_

0 rows × 24 columns

Ordenes de items por mes y estado

Ver que mes anio hay registrado

```
df_orders = dfs['orders.pkl'].copy()

df_orders['created_at'] = pd.to_datetime(df_orders['created_at'], errors='coerce')
```

```
df_orders['mes_anio'] = df_orders['created_at'].dt.to_period('M').dt.to_timestamp()

df_orders = df_orders[['order_id', 'mes_anio', 'shipping_address']]

todos_meses = sorted(df_orders['mes_anio'].dropna().unique())

todos_meses_str = [m.strftime('%b %Y') for m in todos_meses]

print("Meses disponibles:", todos_meses_str)
```

```
Meses disponibles: ['Jul 2022', 'Aug 2022', 'Sep 2022', 'Oct 2022', 'Nov 2022', 'Dec 2022', 'Jan 2023', 'Feb 2023', 'Ma
```

Comparacion de ventas totales por mes y estado. Se tienen en cuenta solo los anios que hay datos al 100%. Se asume que en ningun momento de los meses se dejo de guardar la informacion.

```
df_orders = dfs['orders.pkl'].copy()
df_orders = df_orders[['order_id', 'created_at', 'shipping_address']]

df_orders['created_at'] = pd.to_datetime(df_orders['created_at'], errors='coerce')
df_orders = df_orders.dropna(subset=['created_at'])

df_orders["estado"] = df_orders["shipping_address"].str.extract(r"\b([A-Z]{2})\b")
df_orders['year'] = df_orders['created_at'].dt.year
df_orders['month'] = df_orders['created_at'].dt.month

años_completos = (
    df_orders.groupby('year')['month'].nunique()
    .loc[lambda s: s == 12]
    .index
)

df_fair = df_orders[df_orders['year'].isin(años_completos)]

df_grouped = df_fair.groupby(['month', 'estado']).size().unstack(fill_value=0)

top_15_states = df_grouped.sum().sort_values(ascending=False).head(15).index
df_top15 = df_grouped[top_15_states]

df_top15.index = df_top15.index.map(lambda x: pd.Timestamp(2000, x, 1).strftime('%b'))
```

```
plt.figure(figsize=(12, 8))
sb.heatmap(df_top15,
            annot=True,
            fmt='d',
            cmap="Reds",
            linewidths=0.5,
            linecolor='white',
            cbar_kws={'label': 'Número de órdenes'})
```

```
<Axes: xlabel='estado', ylabel='month'>
```

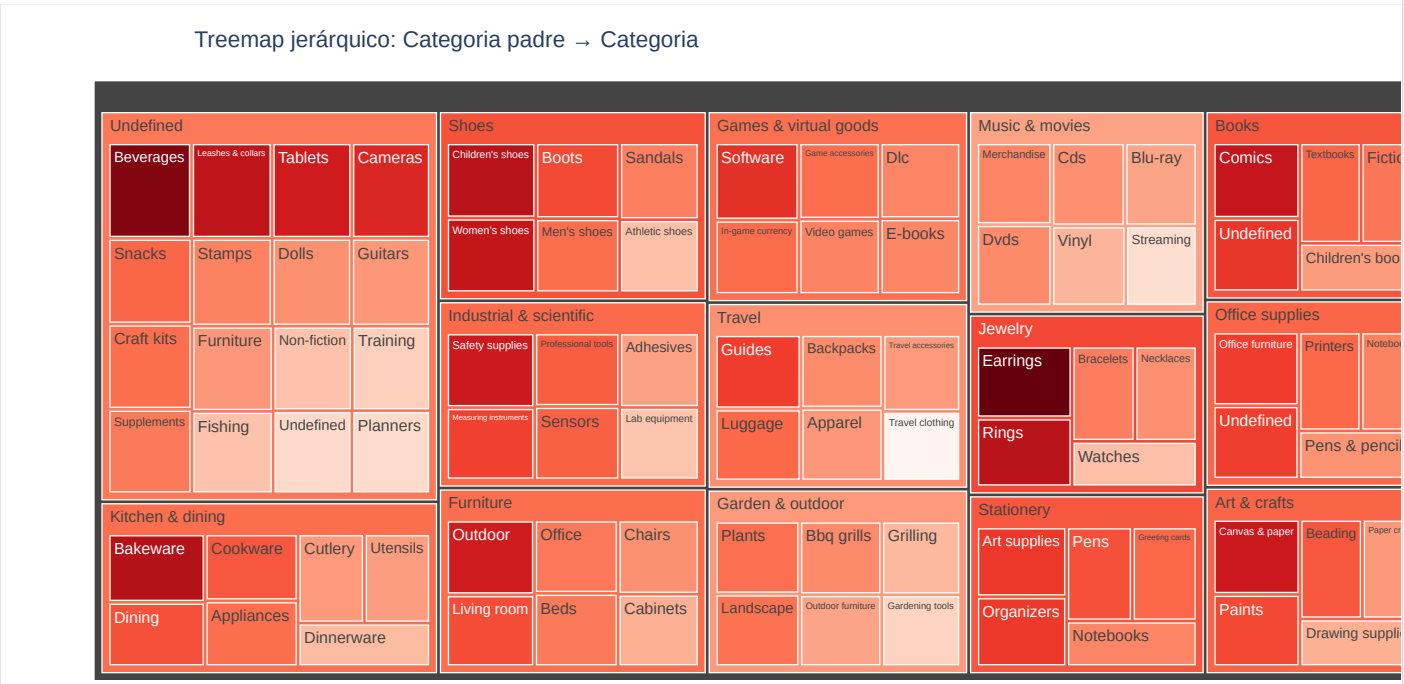
Analisis jerarquico de ordenes de items y sus categorias

Desglose de orders por categorias padre y sus categorias. IMPORTANTE: no son la cantidad de productos, sino las ordenes. Se asume que las relaciones entre las categorias padres y las categorias hijas son correctas.

```
df_order_items = dfs['order_items.pkl'].copy()
df_products = dfs['products.pkl'].copy()
df_categorias = dfs['categories.pkl'].copy()

df_categorias.loc[:,'parent_category'] = df_categorias['parent_category'].str.strip().str.lower().str.capitalize()
df_categorias.loc[:,'category_name'] = df_categorias['category_name'].str.strip().str.lower().str.capitalize()

df_products.fillna({'category_id':'Undefined'}, inplace=True)
df_categorias.fillna({'category_id':'Undefined'}, inplace=True)
df_categorias.fillna({'parent_category':'Undefined'}, inplace=True)

df_merged = df_order_items.merge(df_products, on='product_id')

df_merged = df_merged[['category_id','order_id']].merge(
    df_categorias[['category_id','parent_category','category_name']], on='category_id'
)

df_grouped = df_merged.groupby(['parent_category','category_id','category_name']).size()
df_grouped = df_grouped.reset_index().rename(columns={0:'Ordenes totales'})
```

```
fig = px.treemap(
    df_grouped,
    path=['parent_category', 'category_name'],
    values='Ordenes totales',
    color='Ordenes totales',
    hover_data={'Ordenes totales': True},
    color_continuous_scale='Reds'
)
```

```
fig.update_layout(
    title="Treemap jerárquico: Categoria padre → Categoria",
    margin=dict(t=50, l=25, r=25, b=25)
)

fig.show()
```

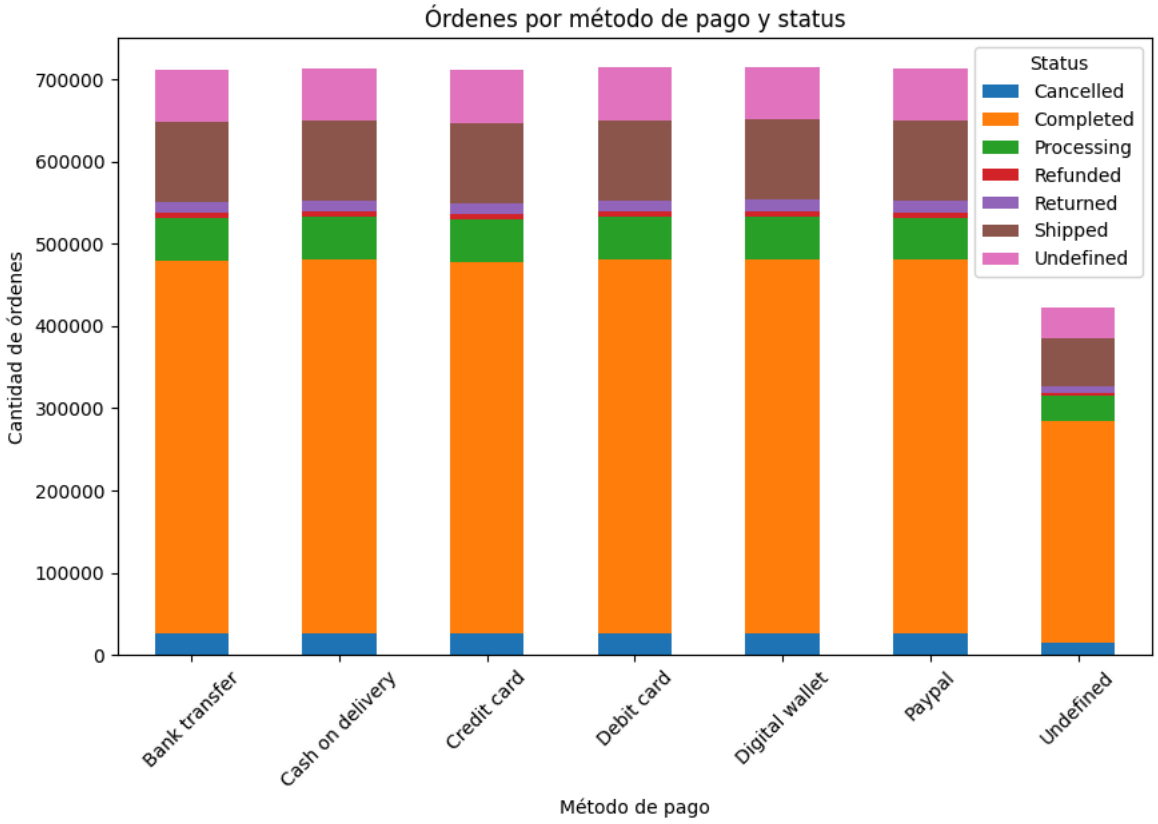


Ordenes en base a su estado y el medio de pago

Comparativa con stacked bar plot de tipos de pagos y estado de orden

```
df_orders=dfs['orders.pkl'].copy()

df_orders = df_orders[['order_id', 'payment_method', 'status']]

df_orders.loc[:, 'payment_method'] = df_orders['payment_method'].str.strip().str.lower().str.capitalize()
df_orders.loc[:, 'status'] = df_orders['status'].str.strip().str.lower().str.capitalize()

df_orders.fillna('Undefined', inplace=True)

df_orders_grouped = df_orders.pivot_table(
    index="payment_method",
    columns="status",
    values="order_id",
    aggfunc="count",
    fill_value=0
)
```

```
df_orders_grouped.plot(kind="bar", stacked=True, figsize=(10,6))

plt.title("Órdenes por método de pago y status")
plt.ylabel("Cantidad de órdenes")
plt.xlabel("Método de pago")
plt.xticks(rotation=45)
plt.legend(title="Status")
plt.show()
```



Clientes y pago

Ordenes devueltas y codigos postales

Analisis de ordenes que se hayan devuelto. Se busca los 5 codigos postales los cuales hayan tenido mas devoluciones. Ademas se bucan el cliente con mas devoluciones en esos codigos postales. Se toman en cuenta las shipping address las cuales contienen 5 numeros consecutivos.

```
df = dfs["orders.pkl"].copy()
df_clientes = dfs["customers.pkl"].copy()

refund_df = df[df["status"] == "Refunded"].copy()

refund_df["codigo_postal"] = (
    refund_df["shipping_address"]
    .str.split(",").str[1]
    .str.strip()
    .str.split().str[-1]
)

top_5_codigos = refund_df["codigo_postal"].value_counts().head(5)

refund_top5 = refund_df[refund_df["codigo_postal"].isin(top_5_codigos.index)]

clientes_frecuentes = refund_top5["customer_id"].mode()[0]

nombre_cliente = df_clientes.loc[df_clientes["customer_id"] == clientes_frecuentes, "first_name"].values[0]

print("Top 5 códigos postales con más reembolsos:")
print(top_5_codigos)
print(f"\nEl nombre más frecuente entre clientes de esas direcciones es: {nombre_cliente}")
```

```
Top 5 códigos postales con más reembolsos:
codigo_postal
70696    6
76449    4
59883    4
```

```
71463    4
90226    4
Name: count, dtype: int64

El nombre más frecuente entre clientes de esas direcciones es: Ashley
```

Metodos de pago y segmento de cliente

Para cada tipo de pago y segmento de cliente, muestra la suma y el promedio expresado como porcentaje, de clientes activos y de consentimiento de marketing. Se tienen en cuentaa los nana y los undefined en todas las combinaciones pues aun es informacion util

```
df_orders = dfs['orders.pkl'].copy()
df_customers = dfs['customers.pkl'].copy()

df_orders = df_orders[['customer_id', 'payment_method']]
df_customers = df_customers[['customer_id', 'customer_segment', 'marketing_consent','is_active']]

df_orders['payment_method'] = df_orders['payment_method'].fillna('no definido')
df_customers['customer_segment']= df_customers['customer_segment'].fillna('no definido')

df_customers['customer_segment'] = df_customers['customer_segment'].replace('undefined', 'No definido')
df_orders['payment_method'] = df_orders['payment_method'].replace('undefined', 'No definido')

df_orders['payment_method'] = df_orders['payment_method'].str.strip().str.lower().str.capitalize()
df_customers['customer_segment'] = df_customers['customer_segment'].str.strip().str.lower().str.capitalize()



df = df_orders.merge(df_customers, on='customer_id')


resumen = df.groupby(['payment_method', 'customer_segment']).agg(
    total_clientes_activos=('is_active', 'sum'),
    total_clientes_marketing=('marketing_consent', 'sum'),
    cantidad_total_clientes=('customer_id', 'count')
).reset_index()

traduccion_pago = {
    'Bank transfer': 'Transferencia bancaria',
    'Cash on delivery': 'Efectivo en entrega',
    'Credit card': 'Tarjeta de crédito',
    'Digital waller' : 'Billetera virtual',

}

resumen['payment_method'] = resumen['payment_method'].replace(traduccion_pago)

resumen['porcentaje_clientes_activos'] = (resumen['total_clientes_activos'] / resumen['cantidad_total_clientes'] * 100
resumen['porcentaje_clientes_marketing'] = (resumen['total_clientes_marketing'] / resumen['cantidad_total_clientes'] *

resumen = resumen.rename(columns={
    'payment_method': 'Tipo de pago',
    'customer_segment': 'Segmento de cliente',
    'total_clientes_activos': 'Clientes activos (total)',
    'total_clientes_marketing': 'Clientes con consentimiento marketing (total)',
    'cantidad_total_clientes': 'Total de clientes',
    'porcentaje_clientes_activos': 'Clientes activos (%)',
    'porcentaje_clientes_marketing': 'Clientes con consentimiento marketing (%)'
})

print(resumen.to_string())
```

```
             Tipo de pago Segmento de cliente  Clientes activos (total)  Clientes con consentimiento marketing (total)
0   Transferencia bancaria              Budget                    114885                                          89153
1   Transferencia bancaria         No definido                     58775                                          45603
2   Transferencia bancaria             Premium                    116360                                          90546
3   Transferencia bancaria             Regular                    350102                                         273106
4      Efectivo en entrega              Budget                    114670                                          88842
5      Efectivo en entrega         No definido                     58901                                          45678
6      Efectivo en entrega             Premium                    117736                                          91760
7      Efectivo en entrega             Regular                    350185                                         273204
8       Tarjeta de crédito              Budget                    113910                                          88255
```

```
9       Tarjeta de crédito        No definido                58729                45383
10      Tarjeta de crédito            Premium               116702                91178
11      Tarjeta de crédito            Regular               350004               273322
12              Debit card             Budget               113869                88403
13              Debit card        No definido                58792                45409
14              Debit card            Premium               117625                91479
15              Debit card            Regular               351591               273540
16          Digital wallet             Budget               114641                89114
17          Digital wallet        No definido                58542                45196
18          Digital wallet            Premium               117553                91429
19          Digital wallet            Regular               351015               274082
20             No definido             Budget                68152                52689
21             No definido        No definido                34851                27029
22             No definido            Premium                69600                54181
23             No definido            Regular               207565               161836
24                  Paypal             Budget               114113                88744
25                  Paypal        No definido                58814                45809
26                  Paypal            Premium               117467                91581
27                  Paypal            Regular               351110               274019
```

Distribucion de tiempo de compras de clientes

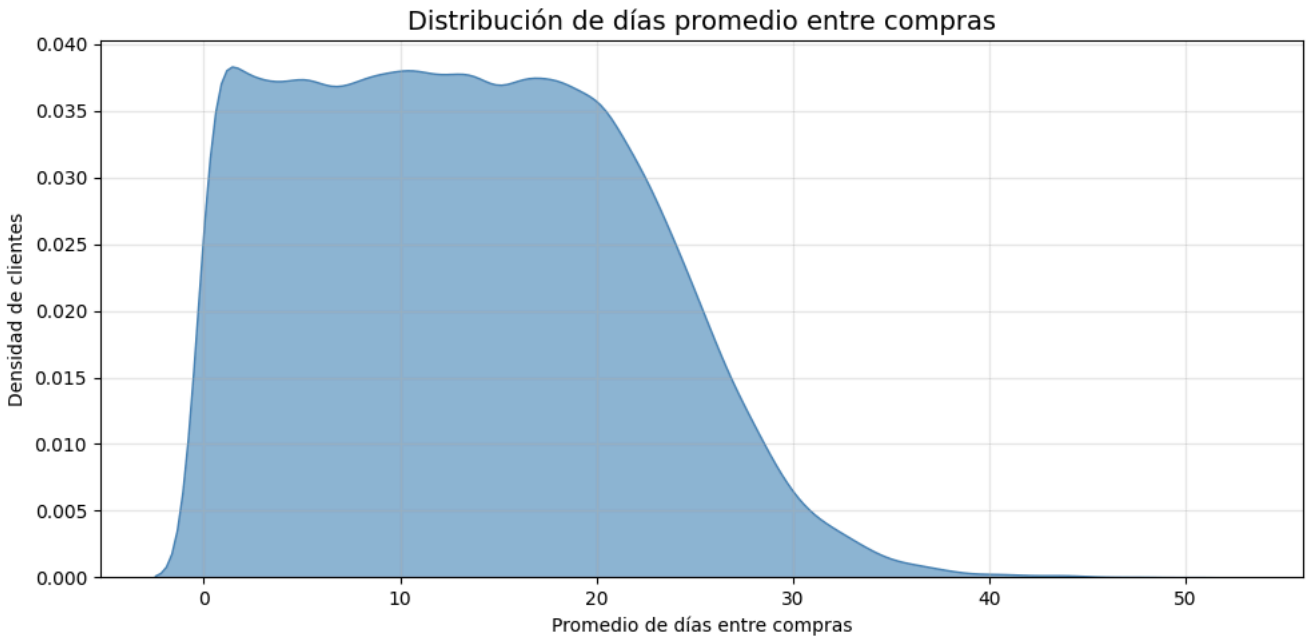
Comparacion entre tiempos de compra promedios de los clientes. Cualquier orden sin fecha de orden queda descartada para el analisis asi como las ordenes que no tengan un cliente asociado

```
df_orders = dfs['orders.pkl'].copy()
df_orders = df_orders[['order_id','order_date','customer_id']].dropna()
df_orders['order_date'] = pd.to_datetime(df_orders['order_date'],errors='coerce')

df_orders = df_orders.sort_values(['customer_id', 'order_date'])
df_orders['days_between'] = df_orders.groupby('customer_id')['order_date'].diff().dt.days

df_summary = df_orders.groupby('customer_id').agg({
    'order_id': 'count',
    'days_between': 'mean'
}).round(1).reset_index()

df_summary.columns = ['customer_id', 'order_count', 'avg_days_between']

def classify_detailed(row):
    order_count = row['order_count']
    avg_days = row['avg_days_between']

    if order_count == 1:
        return 'Una sola compra'
    elif pd.isna(avg_days):
        return 'Sin datos válidos'
    elif avg_days == 0:
        return 'Mismo día (0 días)'
    elif avg_days <= 10:
        return 'Muy frecuente (1-10 días)'
    elif avg_days <= 20:
        return 'Frecuente (11-20 días)'
    elif avg_days <= 30:
        return 'Regular (21-30 días)'
    elif avg_days <= 40:
        return 'Ocasional (31-40 días)'
    elif avg_days <= 50:
        return 'Ocasional (41-50 días)'
    elif avg_days <= 60:
        return 'Poco frecuente (51-60 días)'
    return 'Muy esporádico (>60 días)'

df_summary['segment'] = df_summary.apply(classify_detailed, axis=1)

results = df_summary['segment'].value_counts()
for segment, count in results.items():
    pct = (count / len(df_summary)) * 100
    print(f"{segment}: {count:,} ({pct:.1f}%)")

recurring = df_summary[df_summary['order_count'] > 1]['avg_days_between']
print(f"\nRecurrentes - Promedio: {recurring.mean():.1f}, Mediana: {recurring.median():.1f}")
```

```
Muy frecuente (1-10 días): 37,997 (38.0%)
Frecuente (11-20 días): 37,387 (37.4%)
```

```
Regular (21-30 días): 21,273 (21.3%)
Ocasional (31-40 días): 1,994 (2.0%)
Mismo día (0 días): 1,235 (1.2%)
Ocasional (41-50 días): 104 (0.1%)
Poco frecuente (51-60 días): 2 (0.0%)

Recurrentes - Promedio: 13.3, Mediana: 12.9
```

```
plt.figure(figsize=(10,5))
sb.kdeplot(df_summary["avg_days_between"], fill=True, color="steelblue", alpha=0.6)
plt.title("Distribución de días promedio entre compras", fontsize=14)
plt.xlabel("Promedio de días entre compras")
plt.ylabel("Densidad de clientes")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



Productos y reviews

Productos con stuff en la descripcion

Para los productos que contienen en su descripción la palabra “stuff” (sin importar mayúsculas o minúsculas), calcular el peso total de su inventario agrupado por marca, mostrar sólo la marca y el peso total de las 5 más pesadas. Se rellenan el peso y el stock con 0 si es que hay un nan para no generar datos enganiosos

```
df_products = dfs['products.pkl'].copy()

df_stuff = df_products[df_products['description'].str.contains('stuff', case=False, na=False)].copy()

df_stuff.loc[:, 'weight_kg'] = df_stuff['weight_kg'].fillna(0)
df_stuff.loc[:, 'stock_quantity'] = df_stuff['stock_quantity'].fillna(0)

df_stuff.loc[:, 'brand'] = df_stuff['brand'].fillna("Desconocida")
df_stuff.loc[:, 'brand'] = df_stuff['brand'].replace(["undefined", "Undefined"], "Desconocida")

df_stuff = df_stuff[df_stuff['brand'] != "Desconocida"].copy()

df_stuff.loc[:, 'peso_total'] = df_stuff['weight_kg'] * df_stuff['stock_quantity']

top_5_pesadas = df_stuff.groupby('brand')['peso_total'].sum().nlargest(5)

top_5_pesadas_df = top_5_pesadas.reset_index().rename(columns={
    'brand': 'Marca',
```

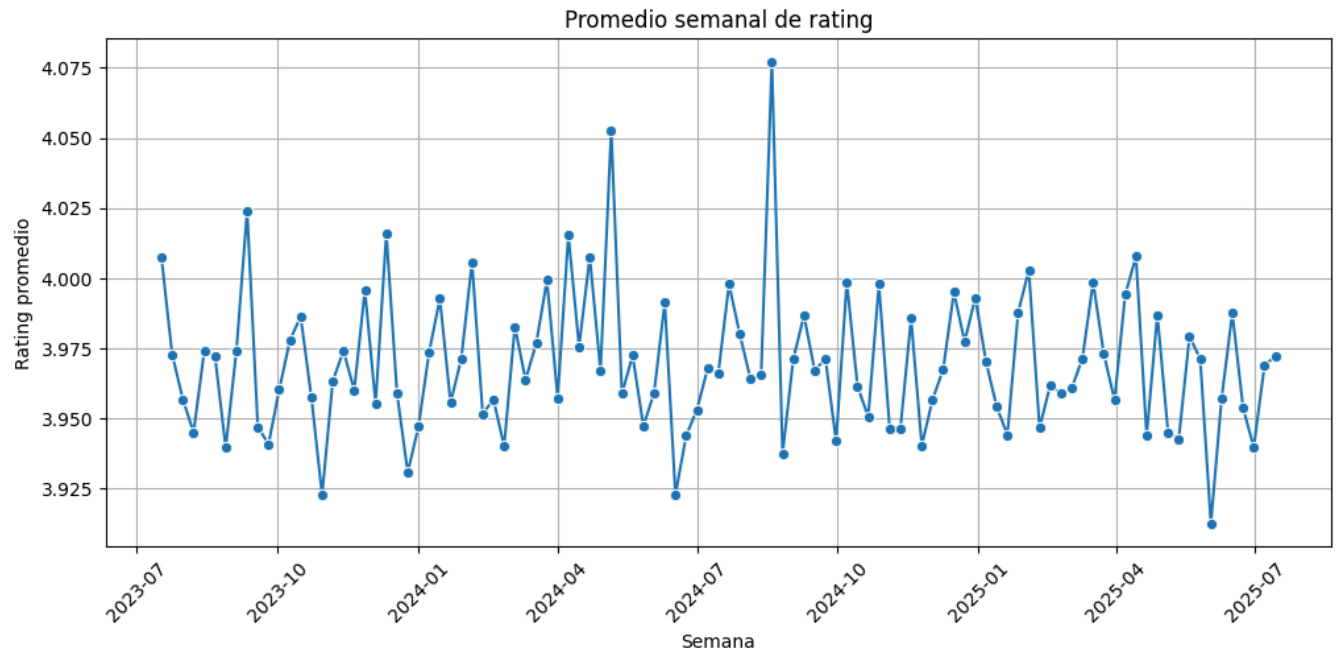
```
    'peso_total': 'Peso total'
})

print(top_5_pesadas_df)
```

```
     Marca  Peso total
0       3M  2037214.18
1   Adidas  1783357.68
2  Wayfair  1508354.30
3   Hasbro  1420969.47
4     Nike  1408352.56
```

Reviews a lo largo del tiempo

Promedio de reviews en base al tiempo. Cualquier review sin fecha registrada no es tenida en cuenta

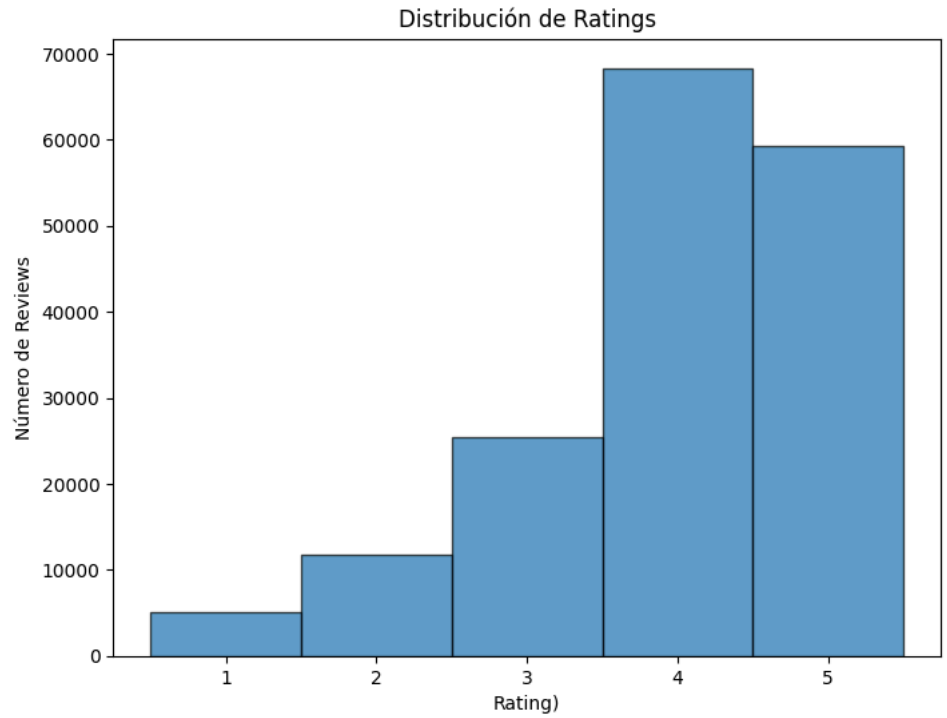
```
df_reviews=dfs['reviews.pkl'].copy()

df_reviews['created_at'] = pd.to_datetime(df_reviews['created_at'],errors='coerce')

df_reviews = df_reviews.dropna(subset=['created_at'])

df_reviews['review_week'] = df_reviews['created_at'].dt.to_period('W').apply(lambda r: r.start_time)

df_reviews=df_reviews[['review_id','review_week','rating']]

df_reviews=df_reviews.groupby(['review_week']).agg({'rating':'mean'}).reset_index()

plt.figure(figsize=(12,5))
sb.lineplot(data=df_reviews, x='review_week', y='rating', marker='o')
plt.xticks(rotation=45)
plt.title("Promedio semanal de rating")
plt.ylabel("Rating promedio")
plt.xlabel("Semana")
plt.grid(True)

# Descomentar para verlo mas general
#plt.ylim(0, 5)

plt.show()
```



Distribucion de rating

Se busca analizar el rating. Se cuentan la cantidad de reviews que hay para poder separar el rating y ver el las resenias en general

```
df_reviews = dfs['reviews.pkl'].copy()
df_reviews = df_reviews[['product_id','rating']]
df_reviews['rating'] = pd.to_numeric(df_reviews['rating'], errors='coerce')
df_reviews = df_reviews.dropna(subset=['rating'])

plt.figure(figsize=(8, 6))
plt.hist(df_reviews['rating'],
         bins=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5],
         edgecolor='black',
         alpha=0.7)
plt.xlabel('Rating)')
plt.ylabel('Número de Reviews')
plt.title('Distribución de Ratings')
plt.xticks([1, 2, 3, 4, 5])
plt.show()

print(f"Total reviews: {len(df_reviews):,}")
print(f"Rating promedio: {df_reviews['rating'].mean():.2f}★")
```




```
Total reviews: 170,000
Rating promedio: 3.97★
```

Palabras mas utilizadas en las reviews

```
from wordcloud import WordCloud

df_reviews = dfs['reviews.pkl'].copy()

texto_completo = ' '.join(df_reviews['title'].fillna('').astype(str))
#descomentar la proxima linea para ver la nube con los comentarios en vez de los titulos y comentar la anterior
#texto_completo = ' '.join(df_reviews['comment'].fillna('').astype(str))

wordcloud = WordCloud(
    width=1200,
    height=600,
    background_color='white',
    max_words=100,
    colormap='viridis',
    stopwords=set(['the', 'and', 'or', 'but', 'in', 'on', 'at', 'to', 'for', 'of', 'with', 'by','a','it'])
).generate(texto_completo.lower())

plt.figure(figsize=(15, 8))
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')

plt.show()
```

Inventario y rentabilidad

Robos y roturas en el inventario

Se analiza la cantidad de incidentes ocacionadas por danios o por robos a lo largo de los registros del inventario. Ademas se se separa en categorias padres para ver si

```
df_logs = dfs['inventory_logs.pkl'].copy()
df_products = dfs['products.pkl'].copy()
df_categorias = dfs['categories.pkl'].copy()

df_logs['movement_type'] = df_logs['movement_type'].str.upper().str.strip()
df_logs['reason'] = df_logs['reason'].fillna('undefined').str.lower().str.strip()
df_logs['quantity_change'] = pd.to_numeric(df_logs['quantity_change'], errors='coerce')

df_categorias['parent_category'] = df_categorias['parent_category'].fillna('Undefined').str.strip().str.lower().str.cap
df_categorias['category_name'] =df_categorias['category_name'].fillna('Undefined').str.strip().str.lower().str.capital

theft_damage = df_logs[df_logs['reason'].isin(['theft', 'damage'])].copy()
print(f"Total incidentes de theft/damage: {len(theft_damage):,}")

df_theft_analysis = (
    theft_damage
    .merge(df_products[['product_id', 'category_id']], on='product_id')
    .merge(df_categorias[['category_id', 'category_name', 'parent_category']], on='category_id')
)

print("\n=== INCIDENTES POR CATEGORÍA PADRE TOP 10 ===")

parent_summary = df_theft_analysis.groupby(['parent_category', 'reason']).agg({
    'quantity_change': 'count',  # Solo incidentes
    'product_id': 'nunique'
}).round(2)
```

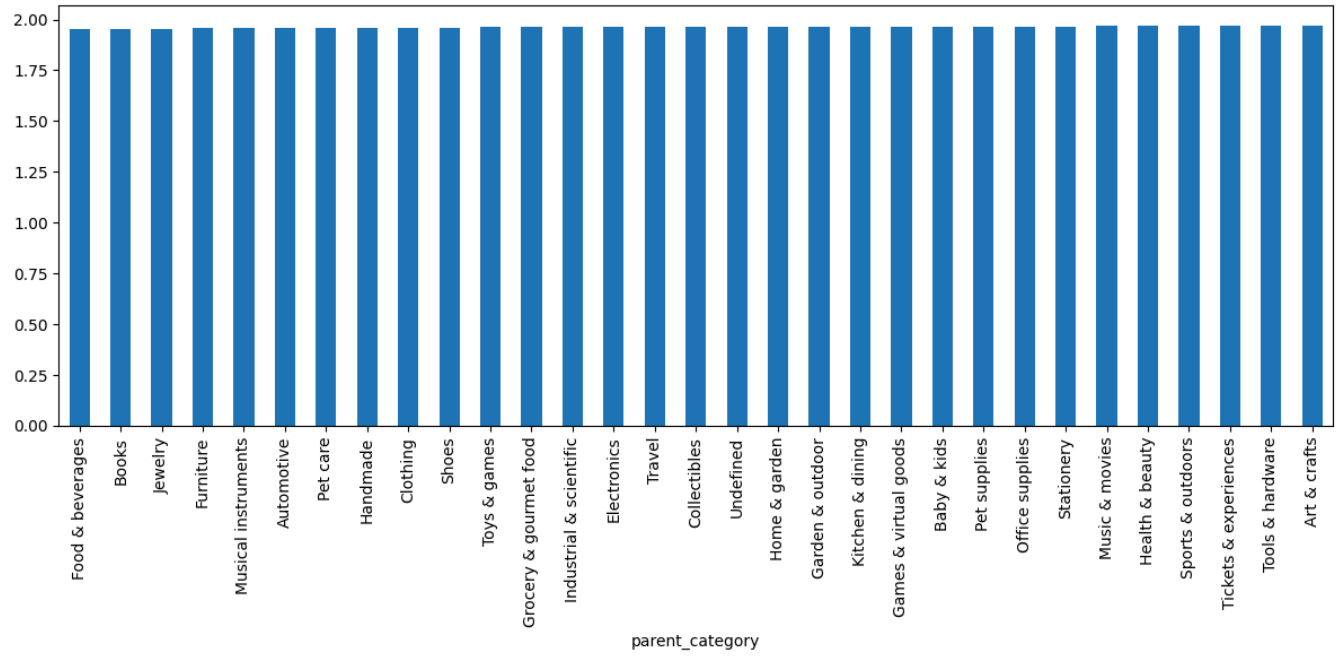
```
parent_summary.columns = ['incidents', 'products_affected']
parent_summary = parent_summary.reset_index()

parent_pivot = parent_summary.pivot_table(
    index='parent_category',
    columns='reason',
    values='incidents',
    fill_value=0
).sort_values(['theft', 'damage'], ascending=False)

print(parent_pivot.head(10))

# Resumen final
theft_incidents = parent_summary[parent_summary['reason'] == 'theft']['incidents'].sum()
damage_incidents = parent_summary[parent_summary['reason'] == 'damage']['incidents'].sum()
theft_categories = parent_summary[parent_summary['reason'] == 'theft']['parent_category'].nunique()
damage_categories = parent_summary[parent_summary['reason'] == 'damage']['parent_category'].nunique()

print(f"\n=== RESUMEN ===")
print(f"Total incidentes de robo: {theft_incidents}")
print(f"Total incidentes de daño: {damage_incidents}")
print(f"Total incidentes: {theft_incidents + damage_incidents}")
print(f"Categorías padre afectadas por robos: {theft_categories}")
print(f"Categorías padre afectadas por daños: {damage_categories}")
```

```
Total incidentes de theft/damage: 103,921

=== INCIDENTES POR CATEGORÍA PADRE TOP 10 ===
reason                   damage   theft
parent_category
Undefined                4548.0  4459.0
Furniture                2006.0  1984.0
Kitchen & dining         1719.0  1673.0
Grocery & gourmet food   1497.0  1557.0
Tools & hardware         1392.0  1553.0
Games & virtual goods    1511.0  1536.0
Shoes                    1470.0  1533.0
Industrial & scientific  1531.0  1518.0
Music & movies           1502.0  1488.0
Pet care                 1460.0  1487.0

=== RESUMEN ===
Total incidentes de robo: 44147
Total incidentes de daño: 44071
Total incidentes: 88218
Categorías padre afectadas por robos: 31
Categorías padre afectadas por daños: 31
```

Profit por categoria padre

Comparacion porcentual de ganancia por cada producto agrupado por categoria padre

```
df_products = dfs['products.pkl'].copy()
df_categorias = dfs['categories.pkl'].copy()

df_products = df_products[['product_id','category_id','price','cost']]
df_products = df_products.dropna()

df_categorias['parent_category'] = df_categorias['parent_category'].fillna('Undefined').str.strip().str.lower().str.ca

df_merge = df_products.merge(df_categorias, left_on='category_id', right_on='category_id')
df_merge['profit_multiplier'] = df_merge['price'] / df_merge['cost']
df_merge = df_merge[['product_id','parent_category','profit_multiplier']]
df_merge = df_merge.groupby(['parent_category']).agg({'profit_multiplier':'mean'}).reset_index()

df_merge.sort_values('profit_multiplier').plot(x='parent_category', y='profit_multiplier', kind='bar', figsize=(12, 6)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Dividiendo los ingresos en los costos y las ganancias

```
df_order_items = dfs['order_items.pkl'].copy()
df_products = dfs['products.pkl'].copy()
df_categories = dfs['categories.pkl'].copy()

df_merged = df_order_items.merge(
    df_products[['product_id', 'category_id', 'cost']],
    on='product_id'
).merge(
    df_categories[['category_id', 'category_name', 'parent_category']],
    on='category_id'
)

df_merged.dropna(subset=['line_total','cost','quantity'],inplace=True)

df_merged['revenue'] = df_merged['line_total']
df_merged['total_cost'] = df_merged['cost'] * df_merged['quantity']
df_merged['profit'] = df_merged['revenue'] - df_merged['total_cost']

df_merged['parent_category']=df_merged['parent_category'].str.strip().str.lower().str.capitalize()
df_merged['parent_category'].fillna('Undefined', inplace=True)

parent_summary = df_merged.groupby('parent_category').agg({
    'revenue': 'sum',
    'total_cost': 'sum',
    'profit': 'sum'
}).reset_index()

top_parents = parent_summary['parent_category'].tolist()

labels = []
sources = []
targets = []
values = []

# Nodos: Ingresos → Categorías Padre → Costos/Ganancias
labels.extend(['INGRESOS'])
labels.extend([f'{parent}' for parent in top_parents])
labels.extend(['COSTOS TOTALES', 'GANANCIA NETA'])

# Flujos: Ingresos → TODAS las Categorías Padre
```

```
# Flujos: Ingresos → TODAS las Categorías Padre
for i, parent in enumerate(top_parents):
    revenue = parent_summary[parent_summary['parent_category'] == parent]['revenue'].iloc[0]
    sources.append(0)  # Desde INGRESOS
    targets.append(i + 1)  # Hacia categoría padre
    values.append(revenue)

# Flujos: TODAS las Categorías Padre → Costos
total_costs_idx = len(labels) - 2
for i, parent in enumerate(top_parents):
    cost = parent_summary[parent_summary['parent_category'] == parent]['total_cost'].iloc[0]
    sources.append(i + 1)  # Desde categoría padre
    targets.append(total_costs_idx)  # Hacia COSTOS
    values.append(cost)

# Flujos: TODAS las Categorías Padre → Ganancias
profit_idx = len(labels) - 1
for i, parent in enumerate(top_parents):
    profit = parent_summary[parent_summary['parent_category'] == parent]['profit'].iloc[0]
    if profit > 0:  # Solo mostrar ganancias positivas
        sources.append(i + 1)  # Desde categoría padre
        targets.append(profit_idx)  # Hacia GANANCIA
        values.append(profit)

category_colors = (px.colors.qualitative.Set1 + px.colors.qualitative.Set2 +
                   px.colors.qualitative.Set3 + px.colors.qualitative.Pastel1)[:len(top_parents)]

node_colors = (
    ['#1f77b4'] +  # Ingresos (azul)
    category_colors +  # Categorías con colores distintos
    ['#ff4444', '#00aa00']  # Costos (rojo), Ganancia (verde)
)

link_colors = []
for s, t in zip(sources, targets):
    if t == total_costs_idx:  # Hacia costos
        link_colors.append('rgba(255,68,68,0.4)')
    elif t == profit_idx:  # Hacia ganancia
        link_colors.append('rgba(0,170,0,0.4)')
    else:  # Hacia categorías
        link_colors.append('rgba(31,119,180,0.4)')

# Crear Sankey
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=20,
        thickness=25,
        line=dict(color="black", width=0.8),
        label=labels,
        color=node_colors
    ),
    link=dict(
        source=sources,
        target=targets,
        value=values,
        color=link_colors
    )
)])

fig.update_layout(
    title_text="Flujo de Dinero: Ingresos → Todas las Categorías → Costos/Ganancias",
    font_size=10,
    width=1400,
    height=900
)

fig.show()
```

```
/tmp/ipython-input-3944772421.py:20: FutureWarning:

A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[
```

Flujo de Dinero: Ingresos → Todas las Categorías → Costos/Ganancias