PROGRAMACIÓN PARA CIENCIA DE DATOS II

Jorge Andres Melo Mayorga
Facultad de Ingeniería, Ciencia de Datos, Universidad Compensar
Actividad Practica Aplicada, Etapa de Transferencia

Sebastián Rodriguez Muñoz
Docente

27 de septiembre de 2024

Contenidos temáticos:

- Métodos Estadísticos Computacionales.
- Diseño de experimentos computacional.
- Manuales de manejo de software específico.
- Protocolo de presentación final de trabajo.

Descripción de la actividad:

El propósito de esta actividad es desarrollar sus habilidades para presentar los resultados que se han obtenido a lo largo del curso. Con esto en mente el entregable que se estará evaluando es un dashboard con los principales descubrimientos sobre la problemática que fue planeada al inicio. Teniendo esto en cuenta, se realizan las siguientes recomendaciones para mejorar el modelo que se ha estado trabajando.

1. Evalué los resultados que ha obtenido en relación con la problemática que se definió durante la primera actividad. Asegúrese, que cada uno de los análisis que haya planteado sea con el objetivo de mejorar la compresión que se tiene del problema.
2. Realice una evaluación del rendimiento que presenta el modelo actual, la habilidad que tiene de ofrecer información sobre la problemática. Define si debe mejorar el modelo iterativamente, o se debe replantear el enfoque de la pregunta inicial para obtener mejores resultados.
3. Considere la recopilación de datos o un procesamiento adicionales de acuerdo a las técnicas que se vieron en el apartado teórico de esta etapa. Puede realizar diferentes y comprobar como mejora el rendimiento dependiendo de la calidad de los datos utilizados.
4. Experimente con los hiperparámetros que definen el modelo. Es importante que tenga métricas que le permitan evaluar el desempeño de los mismos con el objetivo de optimizar los resultados obtenidos. experimentación controlada con los hiperparámetros es una práctica recomendable.

5. Considere si alguno de las métricas del modelo mejoraría información del algún aspecto del problema. La mejora de la capacidad predictiva se puede lograr mediante la creación de nuevas características o la adaptación de las existentes.
6. Considere la Regularización: En caso de que se detecte un sobreajuste en el modelo, se debe considerar la aplicación de técnicas de regularización. Estas implican la incorporación de términos de penalización en la función de pérdida.

Mejore el modelo iterativamente hasta que las evidencias indiquen que el modelo es fiable y se tenga una comprensión avanzada del problema que permitan realizar planteamientos para solucionar el problema.

Una vez este satisfecho con las conclusiones que ha obtenido del modelo, es momento de presentarlo ante una audiencia. Como se mencionó anteriormente, se debe realizar un tablero dashboard con las conclusiones que se hayan obtenido del modelo. A continuación, se presenta un guía de las actividades que debe realizar.

1. Comienza importando las bibliotecas necesarias, como Dash, Plotly y Pandas. Luego, carga los datos de tu modelo de ciencia de datos en un DataFrame de Pandas para que estén listos para su uso.
2. Define las visualizaciones que deseas mostrar y utiliza la biblioteca Dash para crear componentes como gráficos, tablas y filtros interactivos. Organiza estos componentes de manera lógica en la interfaz del dashboard.
3. Da vida a tu dashboard mediante el uso de elementos interactivos. Utiliza callbacks de Dash para permitir a los usuarios seleccionar datos específicos, cambiar escalas o aplicar filtros dinámicos. Asegúrate de que las visualizaciones respondan en tiempo real a las acciones de los usuarios.
4. Agrega contexto y narrativa a tus datos. Utiliza texto descriptivo y explicativo en el dashboard para contar una historia coherente. Acompaña tus puntos clave con gráficos y visualizaciones relevantes para respaldar tu narrativa.
5. Personaliza la apariencia del dashboard para que sea atractivo y coherente con tu narrativa. Utiliza colores, fuentes y diseños que mejoren la legibilidad y la

comprensión. Presta atención a la organización y al espacio en blanco para una presentación ordenada.

6. Realiza pruebas exhaustivas del dashboard. Asegúrate de que todas las interacciones funcionen correctamente y de que la narrativa sea clara. Obtén retroalimentación de otros usuarios o colegas para mejorar la calidad de la presentación y, finalmente, optimiza el rendimiento del dashboard para una carga eficiente.
7. Organiza toda la documentación, archivos y scripts requeridos para ejecutar el dashboard en una carpeta. Realiza pruebas en otros PC o en máquinas virtuales para asegurarte que todo siga funcionando como se espera.

Entregable:

Debe entregar una carpeta comprimida que contenga los siguientes archivos que evidencien su trabajo en el proyecto.

- Informe sobre el reportando todo el proceso de desarrollo del proyecto, se deben presentar una justificación de las decisiones tomas a lo largo del proceso.
- Carpeta con todos los archivos requeridos para visualizar el dashboard.
- Un archivo de text con dos links: el primero con un link al repositorio en GitHub del proyecto. Y otro link para visualizar el dashboard in Binder.
- Diligencie únicamente para actividades tipo foro.

DASHBOARD DE E-COMMERCE - ANÁLISIS DE DATOS Y PUBLICACIÓN RESULTADOS

El proyecto consiste en realizar el análisis y visualización de resultados de los datos de e-commerce, basándose en el dataset público de comercio electrónico brasileño proporcionado por Olist. Fuente: Brazilian E-Commerce Public Dataset by Olist URL: *https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce*.
Conjunto de datos públicos de comercio electrónico brasileño de pedidos realizados en Olist Store. El conjunto de datos tiene información de 100 000 pedidos de 2016 a 2018 realizados en varios mercados de Brasil. Sus características permiten ver un pedido desde múltiples dimensiones: desde el estado del pedido, el precio, el pago y el rendimiento del flete hasta la ubicación del cliente, los atributos del producto y, finalmente, las reseñas escritas por los clientes. También publicamos un conjunto de datos de geolocalización que relaciona los códigos postales brasileños con las coordenadas de latitud y longitud. Se trata de datos comerciales reales, que han sido anonimizados y las referencias a las empresas y socios en el texto de la reseña han sido sustituidas por los nombres de las grandes casas de Juego de Tronos.

OBJETIVOS

Comprender cómo los factores geoespaciales, los tiempos de entrega y los costos de envío se relacionan con los retrasos, con el fin de optimizar la logística y mejorar la eficiencia en las entregas. Para lograrlo, me enfocaré en los siguientes aspectos:

A. **Identificar los tiempos de entrega**: relación con los costos asociados en diferentes regiones para detectar áreas geográficas específicas donde los retrasos son más frecuentes.
B. **Utilizar modelos de regresión**: Analizar cómo variables como la distancia, el costo de envío y los métodos de transporte afectan los tiempos de entrega. Esto me permitirá cuantificar la influencia de cada factor y priorizar las acciones de mejora.

C. **Examinar la variabilidad en los tiempos de entrega**: En función de factores como los días de la semana, las horas pico y las condiciones meteorológicas. Esta información será crucial para mejorar la planificación logística.

D. **Evaluar el impacto de los costos de envío**: en los tiempos de entrega y en la satisfacción del cliente. Esto me ayudará a determinar si es necesario ajustar las tarifas o renegociar acuerdos con proveedores logísticos.

E. **Dashboard**: para visualizar el resultado del análisis obtenido.

CONTENIDO

El proyecto está organizado de manera estructurada y cargado en mi GitHub:[1]
https://github.com/jandresmelo/EDUCATIVO/tree/a8ccd4c6574db01e0cefcbdad155ddccc95f4a4d/OLIST_ECOMMERCE

1. **Datos descargados**:
   Contiene los datasets originales utilizados en el proyecto. Esta sección almacena los datos sin procesar tal como fueron descargados desde la fuente. Carpeta: OLIST_ECOMMERCE/00_DatosDescargados
2. **Base de datos PostgreSQL**:
   Almacena los datos estructuradamente para su posterior análisis. Incluye la configuración de la base de datos PostgreSQL con extensiones GIS para manejar datos geoespaciales y realizar consultas complejas. Carpeta: OLIST_ECOMMERCE/01_BaseDatosSQL.
3. **Análisis descriptivo de los datos**:
   Contiene los scripts y notebooks que realizan el análisis exploratorio y descriptivo de los datos, utilizando tanto PostgreSQL GIS como Python. Esta sección incluye visualizaciones, estadísticas descriptivas y mapas geoespaciales.
   Carpeta: OLIST_ECOMMERCE/02_AnalisisPython

---

[1] *https://github.com/jandresmelo/EDUCATIVO/tree/a8ccd4c6574db01e0cefcbdad155ddccc95f4a4d/OLIST_ECOMMERCE*

4. **Regresión lineal y logística**:

   Desarrolla los modelos de regresión lineal y logística para predecir diferentes variables de interés. Incluye la implementación de los modelos, la evaluación del rendimiento y la interpretación de los resultados.

   Carpeta: OLIST_ECOMMERCE/03_Regresión

5. **Dashboard interactivo**:

   Un dashboard interactivo desarrollado en Python utilizando Streamlit. Este dashboard permite a los usuarios explorar los datos de manera dinámica y visualizar los principales resultados del análisis.

   Carpeta: : OLIST_ECOMMERCE/04_Dashboard, publicado en: *https://educativo-qiqzvyawdt4jmyzjtafrth.streamlit.app/#resultados-analisis-de-datos-bd-olist-e-commerce*

6. **Documento del proyecto**:

   Proporciona una descripción detallada del proyecto, incluyendo el objetivo, metodología, resultados y conclusiones. Este documento sirve como una guía completa para entender el desarrollo y los hallazgos del proyecto. Ubicación: OLIST_ECOMMERCE/05_Documento

CARACTERÍSTICAS DEL PROYECTO

- **Análisis de clientes y pedidos**: Número de clientes por estado, mapa de calor de ubicación de clientes, distribución de órdenes por estado del pedido.
- **Análisis de retrasos**: Distribución de retrasos en la entrega, boxplot de retrasos por método de pago, Q-Q plots de residuos de regresión lineal y Random Forest.
- **Información de productos y vendedores**: Radar chart para la cantidad de vendedores en las principales ciudades, visualización de categorías de productos mediante caras personalizadas.
- **Métodos de pago y evaluaciones**: Distribución de métodos de pago, valor de pagos por número de cuotas, análisis de reseñas.
- **Matriz de correlaciones**: Análisis de correlación entre variables clave.

CONCLUCIONES

La actividad propuesta por el docente señala la importancia fundamental de la integración de conocimientos estadísticos y herramientas tecnológicas avanzadas para llevar a cabo un análisis de datos efectivo y significativo. A través de la construcción de un dashboard interactivo utilizando Python y tecnologías como Streamlit, PostgreSQL, y diversas bibliotecas de visualización, se logra no solo procesar y analizar grandes volúmenes de datos, presentando así los resultados de una manera clara y comprensible.

El conocimiento estadístico es esencial para comprender los datos en profundidad, identificar patrones, tendencias y posibles anomalías. Durante el ejercicio, se aplicaron técnicas estadísticas como la regresión lineal y la evaluación de modelos (MSE, R^2) para predecir y entender la relación entre diversas variables, como el retraso en la entrega y el precio de los productos. Sin una sólida comprensión de estos conceptos, el análisis podría haber sido superficial, perdiendo la oportunidad de extraer información valiosa para la toma de decisiones.

El Uso de Herramientas Tecnológicas para el Análisis de Datos, como Python, PostgreSQL, y bibliotecas de visualización como Matplotlib, Seaborn, y Folium, fue crucial para el éxito del análisis. Estas herramientas permitieron manejar grandes conjuntos de datos, realizar análisis complejos, y generar visualizaciones claras y persuasivas. Además, la implementación en Streamlit permitió que el análisis sea accesible de manera interactiva a través de un dashboard, facilitando la interpretación de los resultados y mejorando la experiencia del usuario final.

La visualización de datos geográficos mediante mapas proporcionó una dimensión adicional crucial al análisis. Herramientas como Folium permitieron representar espacialmente la distribución de clientes y los retrasos en las entregas, revelando patrones que no habrían sido evidentes en gráficos convencionales. Esta capacidad de visualizar datos geográficos es

especialmente útil en escenarios de comercio electrónico, donde la ubicación geográfica puede influir significativamente en la logística y la experiencia del cliente. Los mapas no solo embellecen los informes, sino que también permiten una comprensión más profunda y accesible de los datos analizados.

En conjunto, este ejercicio resalta cómo la sinergia entre conocimiento estadístico y herramientas tecnológicas avanzadas puede transformar datos en decisiones estratégicas, permitiendo a las organizaciones optimizar operaciones y mejorar sus servicios en un entorno competitivo.

Anexo 1 - Ajuste de Análisis Estadístico.
Anexo 2 – Ajuste de Regresión.
Anexo 3 – Resultados Dashboard.

AnalisisDatos

September 27, 2024

[1]:
```
# ESTADISTICA DESCRIPTIVA BASE DATOS OLIST ECOMMERCE
#### Analisis Datos base de datos Brazilian E-Commerce Public Dataset by Olist
#### Fuente: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce
```

[2]:
```
## Instalacion librerias
### pip install folium pandas psycopg2
### pip install scikit-learn
### pip install sqlalchemy
### pip install pyclustertend pandas psycopg2 sqlalchemy
### pip install matplotlib pandas psycopg2 sqlalchemy
### pip install --upgrade psycopg2-binary SQLAlchemy
```

[3]:
```
## Importar Librerias
import numpy as np
import psycopg2
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import folium
from folium.plugins import HeatMap
from sklearn.preprocessing import MinMaxScaler
from sqlalchemy import create_engine
```

[4]:
```
# Identificar tablas

#### Tabla 1 - olist_geolocation: Información de geolocalización.
#### Tabla 2 - olist_order_customers: Información de los clientes.
#### Tabla 3 - olist_orders: : Contiene la información de los pedidos.
#### Tabla 4 - olist_order_items: Detalles de los artículos de cada pedido.
#### Tabla 5 - olist_products: Información de los productos.
#### Tabla 6 - olist_sellers: Información sobre los vendedores.
#### Tabla 7 - olist_order_payments: Información de pagos.
#### Tabla 8 - olist_order_reviews: Reseñas de los pedidos.
#### Tabla 9 - product_category_name_translation: traducciones de los nombres␣
 ↪de las categorías de productos desde portugués a ingles.
#### Tabla 10 - spatial_ref_sys: Extensión de PostgreSQL que permite manejar␣
 ↪datos geográficos.
```

[5]:
```
## 1) Tabla olist_geolocation: Almacena las coordenadas tipo punto de cada␣
 ↪codigo zip.
### Generaliza la capa para ubicar los estados a partir de centroide



# Conectar a la base de datos PostgreSQL y ejecutar la consulta
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Ejecutar la consulta SQL para calcular el centroide por geolocation_state
    query = """
    SELECT
        geolocation_state as estado,
        AVG(geolocation_lat) AS lat_centroide,
        AVG(geolocation_lng) AS lng_centroide
    FROM
        olist_geolocation
    GROUP BY
       geolocation_state
    ORDER BY
       geolocation_state;
    """

    df = pd.read_sql_query(query, conn)

    # Cerrar la conexión
    conn.close()

    # Crear un mapa centrado en Brasil
    m = folium.Map(location=[-14.2350, -51.9253], zoom_start=4)

    # Añadir los centroides al mapa
    for index, row in df.iterrows():
        folium.Marker(
            location=[row['lat_centroide'], row['lng_centroide']],
            popup=f"Estado: {row['estado']}"
        ).add_to(m)

    # Guardar el mapa en un archivo HTML
```

```
    m.save('ubicacion_generalizada.html')
    print("Mapa de centroides guardado como 'ubicacion_generalizada.html'")

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    if conn:
        conn.close()
```

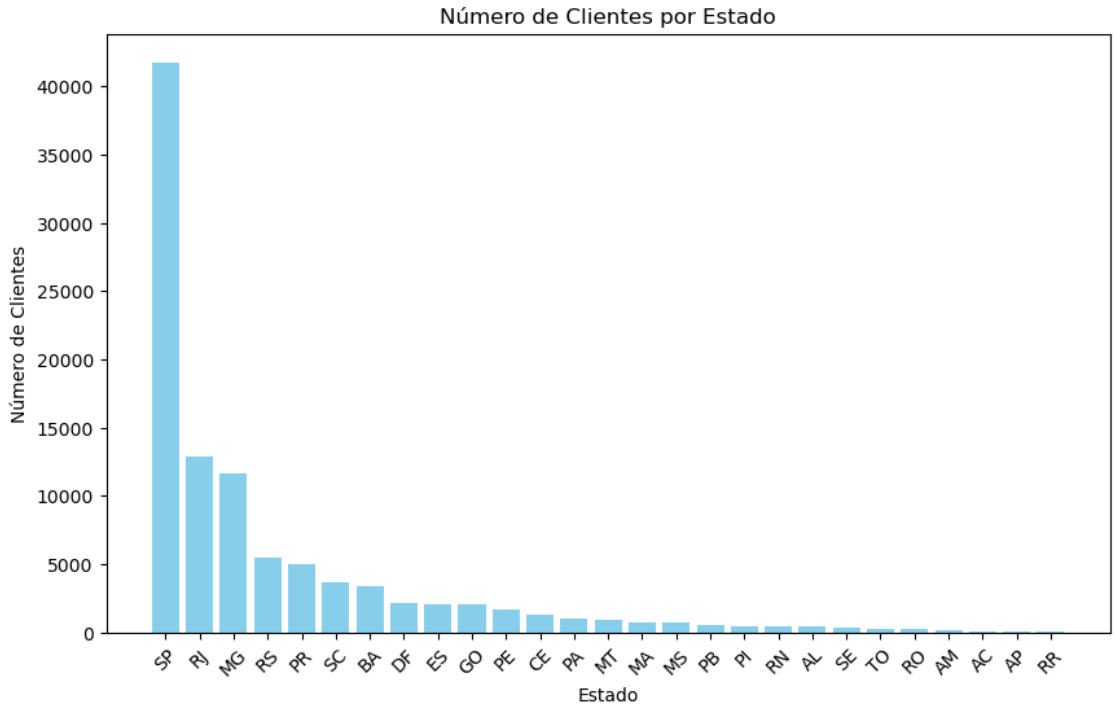
```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\1339922584.py:31: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```

```
Mapa de centroides guardado como 'ubicacion_generalizada.html'
```

[6]:
```
## 2) Tabla olist_order_customers: Información de los clientes.
### Cuantos Clientes tiene cada estado



# Conectar a la base de datos PostgreSQL
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar una consulta
    cur = conn.cursor()

    # Ejecutar una consulta de prueba para verificar la conexión
    cur.execute("SELECT version();")

    # Obtener el resultado
    db_version = cur.fetchone()
    print(f"Conectado a: {db_version[0]}")

    # Consulta para contar el número de clientes por estado
    query = """
    SELECT
        customer_state AS estado,
```

```
        COUNT(*) AS num_cliente
    FROM
        olist_order_customers
    GROUP BY
        customer_state
    ORDER BY
        num_cliente DESC;
    """

    # Ejecutar la consulta y leer los resultados en un DataFrame de Pandas
    df = pd.read_sql(query, conn)

    # Graficar los resultados
    plt.figure(figsize=(10, 6))
    plt.bar(df['estado'], df['num_cliente'], color='skyblue')
    plt.xlabel('Estado')
    plt.ylabel('Número de Clientes')
    plt.title('Número de Clientes por Estado')
    plt.xticks(rotation=45)
    plt.show()

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    # Cerrar la conexión si fue abierta
    if conn:
        cur.close()
        conn.close()
        print("Conexión cerrada.")
```

```
Conectado a: PostgreSQL 15.2, compiled by Visual C++ build 1914, 64-bit

C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\3638424074.py:41: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql(query, conn)
```



Conexión cerrada.

[7]:
```
## Mapa de ubicacion de clientes

# Conectar a la base de datos PostgreSQL
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar una consulta
    cur = conn.cursor()

    # Ejecutar una consulta para obtener latitud, longitud y contar clientes␣
 ↪por ciudad
    query = """
    SELECT
        customer_city,
        COUNT(*) AS num_cliente,
```

```
        AVG(ST_Y(location::geometry)) AS latitude,
        AVG(ST_X(location::geometry)) AS longitude
    FROM
        olist_order_customers
    WHERE
        location IS NOT NULL
    GROUP BY
        customer_city
    ORDER BY
        num_cliente DESC;
    """

    # Ejecutar la consulta y leer los resultados en un DataFrame de Pandas
    df = pd.read_sql(query, conn)

    # Cerrar la conexión
    cur.close()
    conn.close()

    # Crear el mapa Brasil
    m = folium.Map(location=[-14.2350, -51.9253], zoom_start=4)

    # Preparar los datos para el HeatMap (lista de [latitud, longitud, peso])
    heat_data = [[row['latitude'], row['longitude'], row['num_cliente']] for␣
 ↪index, row in df.iterrows()]

    # Añadir el HeatMap al mapa
    HeatMap(heat_data).add_to(m)

    # Guardar el mapa en un archivo HTML
    m.save('clientes_estado_heatmap.html')
    print("Mapa de calor guardado como 'clientes_estado_heatmap.html'")

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    # Asegurar que la conexión se cierre incluso si hay un error
    if conn:
        cur.close()
        conn.close()
        print("Conexión cerrada.")
```

```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\1303304301.py:35: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql(query, conn)
```

```
Mapa de calor guardado como 'clientes_estado_heatmap.html'
Conexión cerrada.
```

[10]:
```python
### Ciudades con Más de 500 Pedidos en la Plataforma de E-commerce

# Crear la cadena de conexión con SQLAlchemy
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/
 ↪olist_ecommerce')

# Consulta SQL
query = """
SELECT
    customer_city,
    COUNT(customer_id) AS CantidadPedidos
FROM
    olist_order_customers
GROUP BY
    customer_city
HAVING
    COUNT(customer_id) > 500
ORDER BY
    CantidadPedidos DESC
LIMIT 10;
"""

# Ejecutar la consulta y cargar los datos en un DF
zona = pd.read_sql_query(query, engine)

# Mostrar los datos resultantes
print(zona)
```

```
           customer_city  cantidadpedidos
0              sao paulo            15540
1         rio de janeiro             6882
2         belo horizonte             2773
3               brasilia             2131
4               curitiba             1521
5               campinas             1444
6           porto alegre             1379
7               salvador             1245
8              guarulhos             1189
9  sao bernardo do campo              938
```

[11]:
```python
### Cantidad de Pedidos por Estado


# Crear la cadena de conexión con SQLAlchemy
```

```
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/
 ↪olist_ecommerce')

try:
    # Consulta SQL
    query = """
    SELECT
        customer_city,
        customer_state,
        COUNT(customer_id) AS cantidadpedidos,
        AVG(ST_Y(location::geometry)) AS latitude,
        AVG(ST_X(location::geometry)) AS longitude
    FROM
        public.olist_order_customers
    WHERE
        location IS NOT NULL
    GROUP BY
        customer_city, customer_state
    HAVING
        COUNT(customer_id) > 1000
    ORDER BY
        cantidadpedidos DESC;
    """

    # Ejecutar la consulta y cargar los resultados en un DataFrame de Pandas
    df = pd.read_sql_query(query, engine)

    # Verificar el contenido del DataFrame
    print(df.head())

    # Crear un mapa centrado en Brasil
    m = folium.Map(location=[-14.2350, -51.9253], zoom_start=4)

    # Añadir los centroides al mapa
    for _, row in df.iterrows():
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=f"Ciudad: {row['customer_city']}, Pedidos:␣
 ↪{row['cantidadpedidos']}",
            icon=folium.Icon(color='green')
        ).add_to(m)

    # Guardar el mapa en un archivo HTML
    m.save('pedidos-estado.html')
    print("Mapa de centroides guardado como 'upedidos-estado.html'")

except Exception as e:
```

```
    print(f"Error al conectar a la base de datos: {e}")
```

```
    customer_city customer_state  cantidadpedidos   latitude  longitude
0       sao paulo             SP            15538 -23.572600 -46.633861
1  rio de janeiro             RJ             6882 -22.923370 -43.328131
2  belo horizonte             MG             2773 -19.910936 -43.958786
3        brasilia             DF             1960 -15.814284 -47.960272
4        curitiba             PR             1521 -25.454796 -49.275508
Mapa de centroides guardado como 'upedidos-estado.html'
```

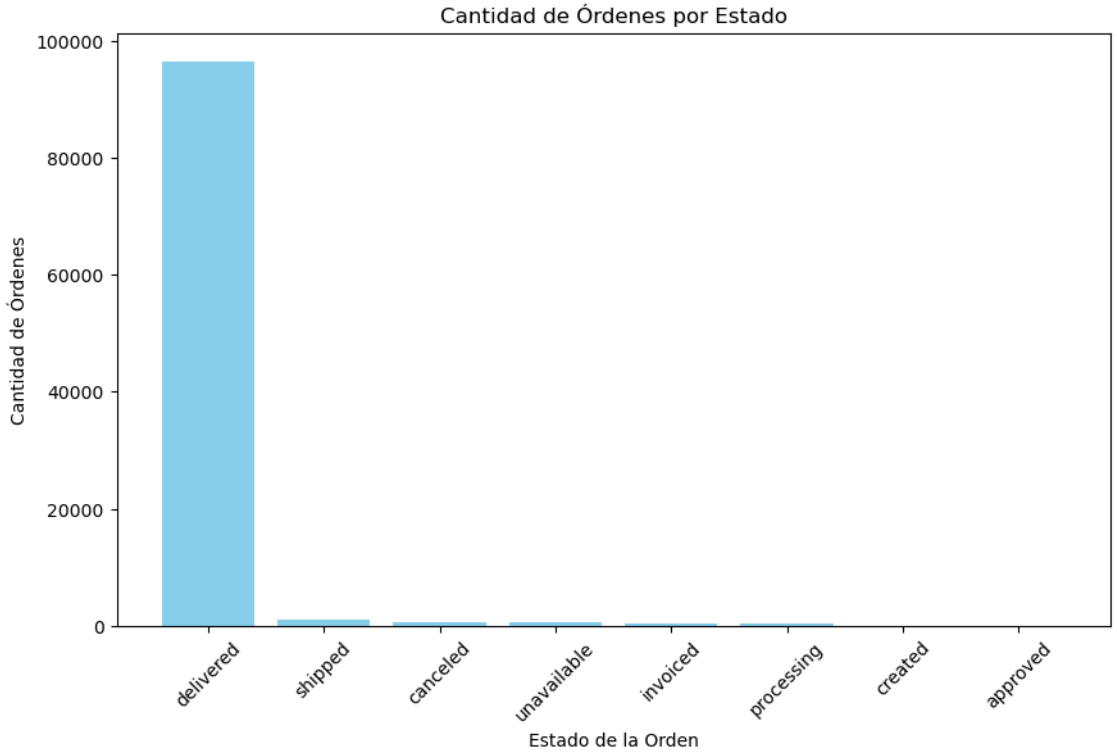
[12]:
```
#### visualizar las zonas geográficas donde se realizan más entregas de productos en una plataforma de e-commerce.
```

[13]:
```
## 3) Tabla olist_orders: : Contiene la información de los pedidos.

### Relación entre el estado del pedido y el retraso en la entrega
orders = pd.read_sql_query("SELECT * FROM olist_orders;", con=engine)

orders_status = pd.read_sql_query("""
    SELECT
        order_status,
        COUNT(order_id) AS CantidadOrdenes
    FROM
        olist_orders
    GROUP BY
        order_status
    ORDER BY
        CantidadOrdenes DESC;
""", con=engine)

print(orders_status)
```

```
  order_status  cantidadordenes
0    delivered            96478
1      shipped             1107
2     canceled              625
3  unavailable              609
4     invoiced              314
5   processing              301
6      created                5
7     approved                2
```

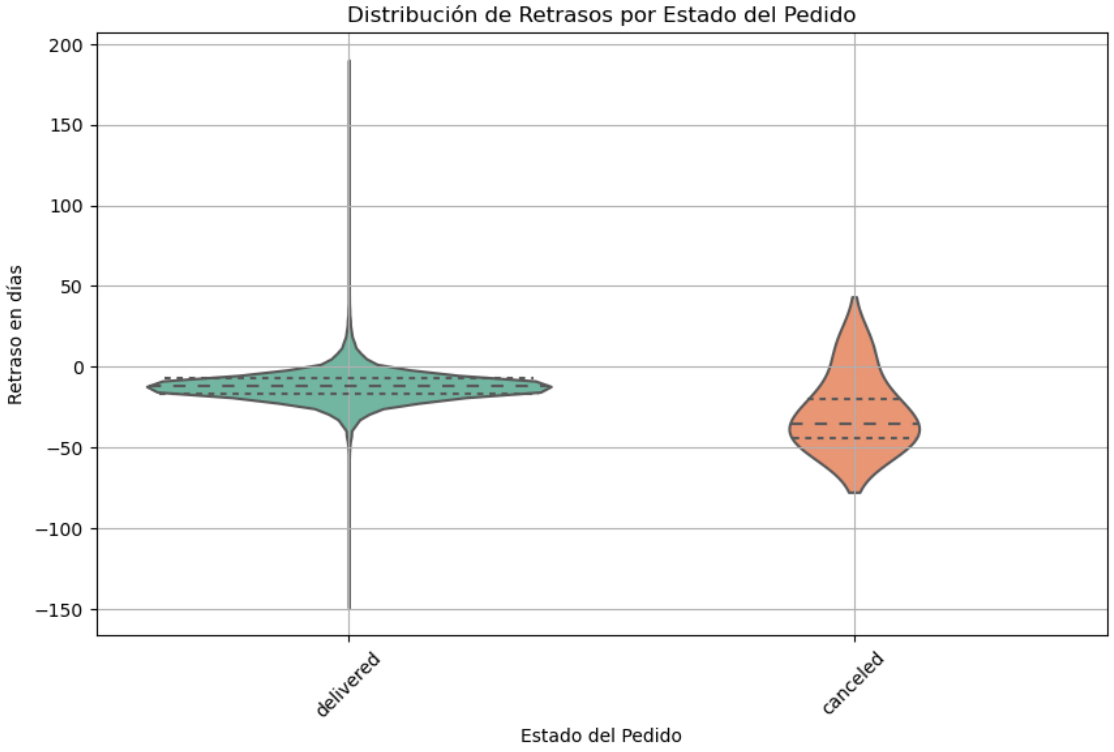
[14]:
```
### Gráfico Cantidad de Órdenes por Estado

plt.figure(figsize=(10, 6))
plt.bar(orders_status['order_status'], orders_status['cantidadordenes'], color='skyblue')
plt.title('Cantidad de Órdenes por Estado')
```

```
plt.xlabel('Estado de la Orden')
plt.ylabel('Cantidad de Órdenes')
plt.xticks(rotation=45)
plt.show()
```



[15]:
```
### Estado del Pedido y Retraso en la Entrega

query_status_delay = """
SELECT
        order_id,
        order_status,
        order_delivered_customer_date,
        order_estimated_delivery_date
FROM olist_orders
WHERE order_delivered_customer_date IS NOT NULL;
"""

# Ejecutar la consulta SQL y cargar los resultados en un DataFrame
df_status = pd.read_sql_query(query_status_delay, con=engine)

# Calcular el retraso en la entrega (en días)
df_status['delivery_delay'] = (
```

```
    pd.to_datetime(df_status['order_delivered_customer_date']) - pd.
 ↪to_datetime(df_status['order_estimated_delivery_date'])
).dt.days

# Configuración del tamaño del gráfico
plt.figure(figsize=(10, 6))

# Gráfico de violín para mostrar la distribución del retraso según el estado␣
 ↪del pedido
sns.violinplot(x='order_status', y='delivery_delay', data=df_status,␣
 ↪inner="quartile", palette="Set2")

# Ajustes del gráfico
plt.title('Distribución de Retrasos por Estado del Pedido')
plt.xlabel('Estado del Pedido')
plt.ylabel('Retraso en días')
plt.xticks(rotation=45)
plt.grid(True)

# Mostrar el gráfico
plt.show()
```



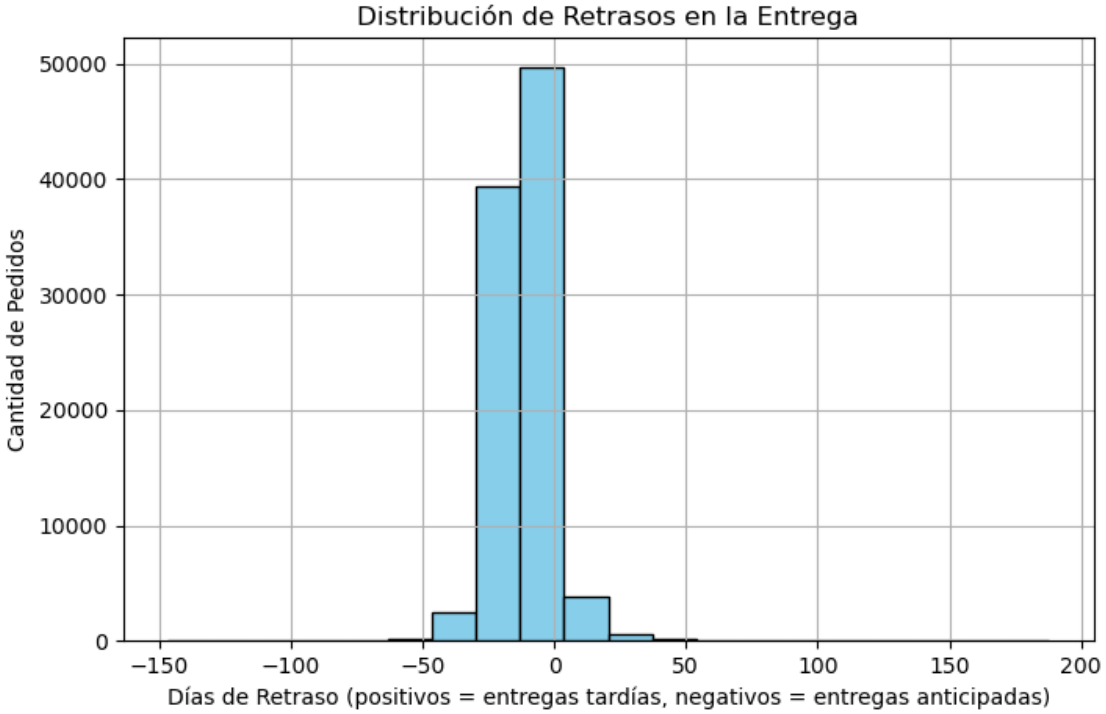
[16]:
```
'\nLa distribucion describe que entre los pedidos entregados se entregan␣
↪anticipadamente (valores negativos) y otros con pequeños retrasos \n(valores␣
↪positivos). En cuanto a pedidos cancelados se observan que fueron cancelados␣
↪antes de la fecha de entrega estimada.\n'
```

[16]:
```
'\nLa distribucion describe que entre los pedidos entregados se entregan
anticipadamente (valores negativos) y otros con pequeños retrasos \n(valores
positivos). En cuanto a pedidos cancelados se observan que fueron cancelados
antes de la fecha de entrega estimada.\n'
```

[17]:
```
# Conectar a base de datos
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Consulta SQL fechas de entrega estimadas vs  enregadas
    query = """
    SELECT
        order_id,
        order_estimated_delivery_date,
        order_delivered_customer_date
    FROM
        olist_orders
    WHERE
        order_delivered_customer_date IS NOT NULL
        AND order_estimated_delivery_date IS NOT NULL;
    """

    # Ejecutar la consulta y cargar los resultados en un DataFrame de Pandas
    df = pd.read_sql_query(query, conn)

    # Cerrar la conexión
    conn.close()

    # Calcular el retraso en días como la diferencia entre la fecha de entrega␣
↪real y la estimada
    df['delivery_delay'] = (df['order_delivered_customer_date'] -␣
↪df['order_estimated_delivery_date']).dt.days

    # Verificar si la columna 'delivery_delay' se ha calculado correctamente
    if 'delivery_delay' in df.columns:
```

```
        # Visualización gráfica: Histograma de retrasos en las entregas
        plt.figure(figsize=(8, 5))
        plt.hist(df['delivery_delay'], bins=20, color='skyblue',␣
 ↪edgecolor='black')
        plt.title('Distribución de Retrasos en la Entrega')
        plt.xlabel('Días de Retraso (positivos = entregas tardías, negativos =␣
 ↪entregas anticipadas)')
        plt.ylabel('Cantidad de Pedidos')
        plt.grid(True)
        plt.show()
    else:
        print("Error: La columna 'delivery_delay' no se calculó correctamente.")

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    if conn:
        conn.close()
```

```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\1615506034.py:26: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```

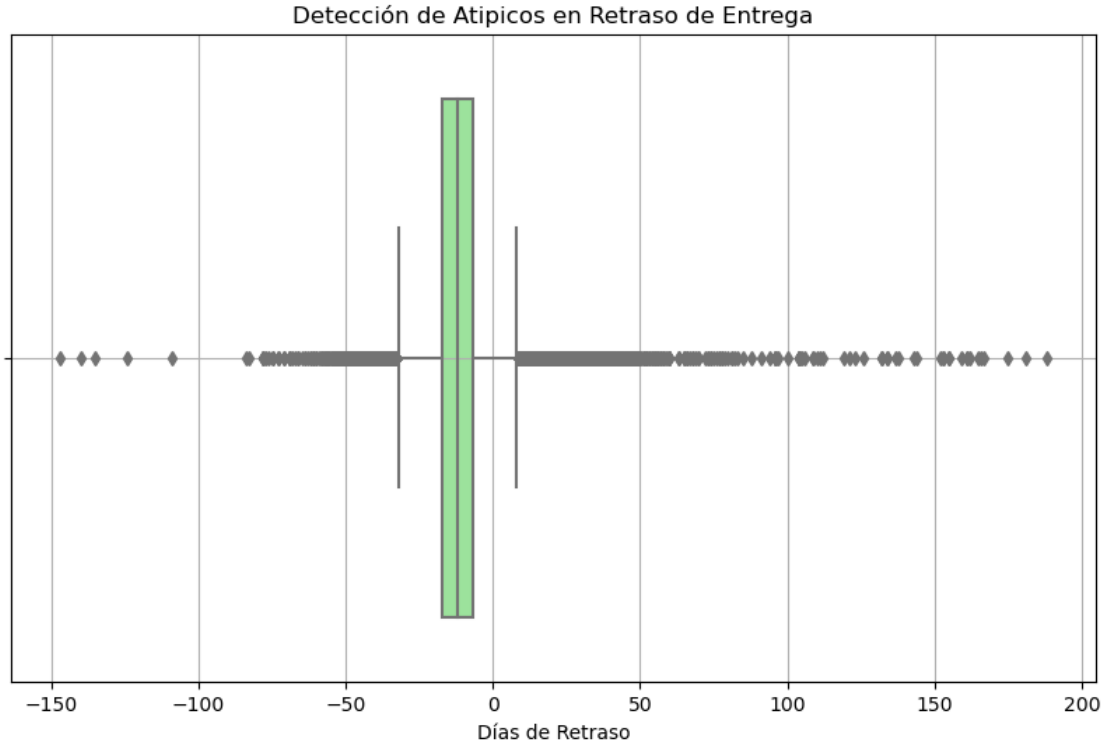


[18]:
```
#### la mayoría de las entregas se realizan antes de la fecha estimada, con un␣
↪pico alrededor de los 10 a 30 días antes de la entrega esperada.
```

[19]:
```
### Detección de Atipicos en Retraso de Entrega

# Conectar a la base de datos PostgreSQL y ejecutar la consulta
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Consulta SQL para obtener las fechas de entrega estimadas y reales
    query = """
    SELECT
        order_id,
        order_estimated_delivery_date,
        order_delivered_customer_date
    FROM
        olist_orders
    WHERE
        order_delivered_customer_date IS NOT NULL
        AND order_estimated_delivery_date IS NOT NULL;
    """

    # Ejecutar la consulta y cargar los resultados en un DataFrame de Pandas
    df = pd.read_sql_query(query, conn)

    # Cerrar la conexión
    conn.close()

    # Calcular el retraso en días como la diferencia entre la fecha de entrega␣
↪real y la estimada
    df['delivery_delay'] = (df['order_delivered_customer_date'] -␣
↪df['order_estimated_delivery_date']).dt.days

    # Verificar si la columna 'delivery_delay' se ha calculado correctamente
    if 'delivery_delay' in df.columns:
        # Visualización gráfica: Boxplot para detectar outliers en los retrasos␣
↪de entrega
        plt.figure(figsize=(10, 6))
```

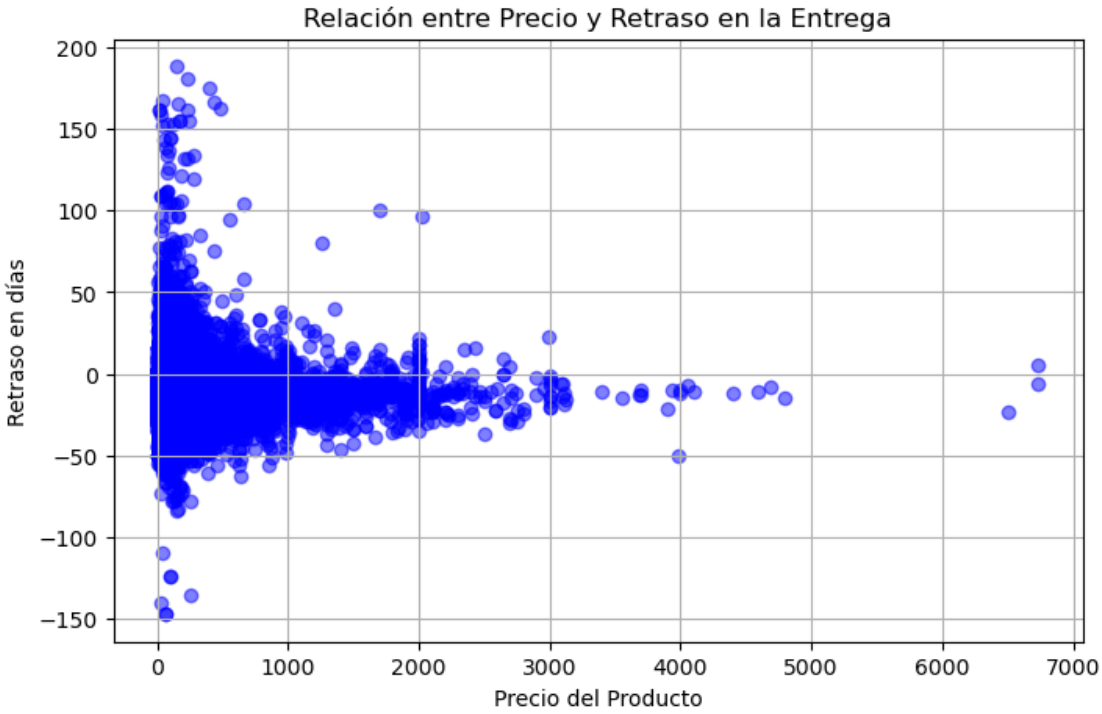
```
        sns.boxplot(x=df['delivery_delay'], color='lightgreen')
        plt.title('Detección de Atipicos en Retraso de Entrega')
        plt.xlabel('Días de Retraso')
        plt.grid(True)
        plt.show()
    else:
        print("Error: La columna 'delivery_delay' no se calculó correctamente.")

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    if conn:
        conn.close()
```

```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\4269501460.py:28: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```



[20]:
```
### Relación entre Precio y Retraso de la Entrega
```

```
query_price_delay = """
SELECT  o.order_id,
        o.order_delivered_customer_date,
        o.order_estimated_delivery_date,
        oi.price
FROM olist_orders o

JOIN olist_order_items oi ON o.order_id = oi.order_id

WHERE o.order_delivered_customer_date IS NOT NULL;
"""

# Ejecutar la consulta SQL y cargar los resultados en un DataFrame
df_price = pd.read_sql_query(query_price_delay, con=engine)

# Calcular el retraso en la entrega (en días)
df_price['delivery_delay'] = (
    pd.to_datetime(df_price['order_delivered_customer_date']) - pd.
 ↪to_datetime(df_price['order_estimated_delivery_date'])
).dt.days

# Visualización gráfica: Relación entre el precio y el retraso en la entrega
plt.figure(figsize=(8, 5))
plt.scatter(df_price['price'], df_price['delivery_delay'], color='blue',␣
 ↪alpha=0.5)
plt.title('Relación entre Precio y Retraso en la Entrega')
plt.xlabel('Precio del Producto')
plt.ylabel('Retraso en días')
plt.grid(True)
plt.show()
```



[21]:
```
"""
La mayoría de los puntos se concentran en precios bajos, por debajo de 1000␣
 ↪unidades, para estos productos, los retrasos varían considerablemente,
desde entregas anticipadas hasta entregas tardías. Entonces se puede indicar␣
 ↪que a medida que aumenta el precio del producto, la variabilidad
en los retrasos disminuye.
"""
```

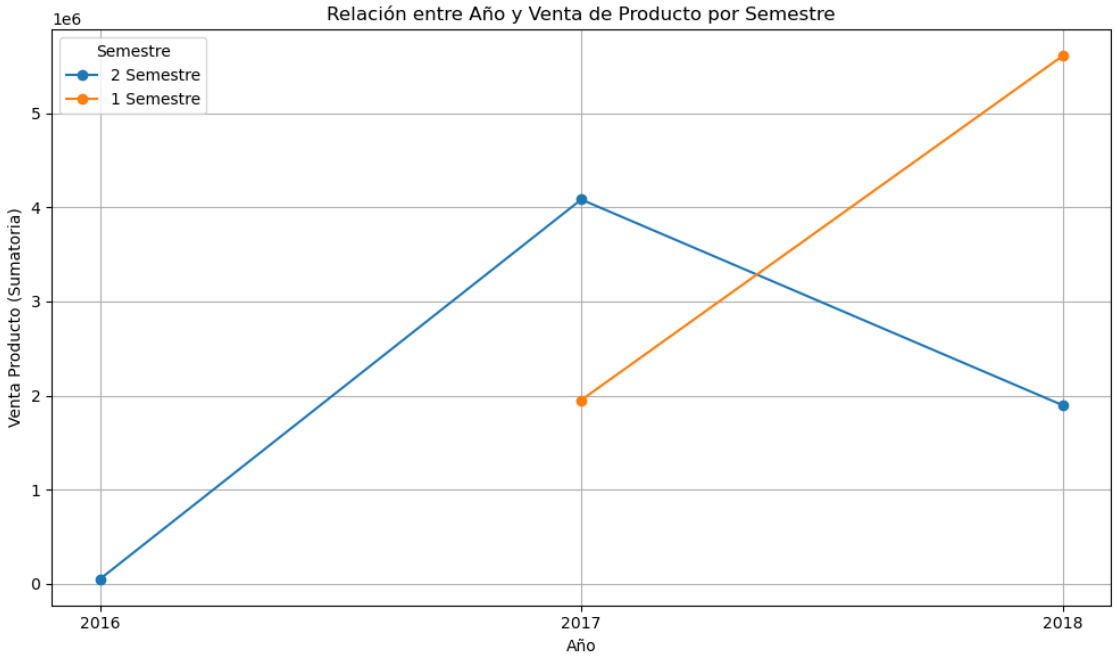
[21]:
```
'\nLa mayoría de los puntos se concentran en precios bajos, por debajo de 1000
unidades, para estos productos, los retrasos varían considerablemente, \ndesde
entregas anticipadas hasta entregas tardías. Entonces se puede indicar que a
medida que aumenta el precio del producto, la variabilidad \nen los retrasos
disminuye.\n'
```

[22]:
```
## 4) Tabla olist_order_items: Detalles de los artículos de cada pedido.
### Relación entre Año y Venta de Producto por Semestre.


# Conectar a la base de datos PostgreSQL
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
```

```
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar una consulta
    cur = conn.cursor()

    # Consulta SQL para obtener anio, semestre, venta_producto y costo_envio
    query = """
    SELECT
        EXTRACT(YEAR FROM shipping_limit_date) AS anio,
        CASE
            WHEN EXTRACT(MONTH FROM shipping_limit_date) <= 6 THEN '1 Semestre'
            ELSE '2 Semestre'
        END AS semestre,
        ROUND(SUM(price)) AS venta_producto,
        ROUND(SUM(freight_value)) AS costo_envio
    FROM
        olist_order_items
    WHERE
        EXTRACT(YEAR FROM shipping_limit_date) <> 2020
    GROUP BY
        anio, semestre
    ORDER BY
        anio, semestre;
    """

    # Ejecutar la consulta y leer los resultados en un DataFrame de Pandas
    df = pd.read_sql(query, conn)

    # Graficar la relación entre anio y venta_producto
    plt.figure(figsize=(10, 6))
    for semestre in df['semestre'].unique():
        df_semestre = df[df['semestre'] == semestre]
        plt.plot(df_semestre['anio'], df_semestre['venta_producto'],␣
↪marker='o', label=semestre)

    plt.title('Relación entre Año y Venta de Producto por Semestre')
    plt.xlabel('Año')
    plt.ylabel('Venta Producto (Sumatoria)')
    plt.xticks(df['anio'].unique())
    plt.grid(True)
    plt.legend(title='Semestre')
    plt.tight_layout()
    plt.show()
```

```
except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    # Cerrar la conexión si fue abierta
    if conn:
        cur.close()
        conn.close()
        print("Conexión cerrada.")
```

```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\3427020997.py:40: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql(query, conn)
```




```
Conexión cerrada.
```

[23]:
```
### Tabla olist_orders items: Datos simplificados ventas por semestre


# Conectar a la base de datos PostgreSQL y ejecutar la consulta
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
```

```
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar una consulta
    cur = conn.cursor()

    # Consulta SQL para obtener el año, semestre, venta_producto y costo_envio
    query = """
    SELECT
        EXTRACT(YEAR FROM shipping_limit_date) AS anio,
        CASE
            WHEN EXTRACT(MONTH FROM shipping_limit_date) <= 6 THEN '1 Semestre'
            ELSE '2 Semestre'
        END AS semestre,
        ROUND(SUM(price)) AS venta_producto,
        ROUND(SUM(freight_value)) AS costo_envio
    FROM
        olist_order_items
    WHERE
        EXTRACT(YEAR FROM shipping_limit_date) <> 2020
    GROUP BY
        anio, semestre
    ORDER BY
        anio, semestre;
    """

    # Ejecutar la consulta y leer los resultados en un DataFrame de Pandas
    df = pd.read_sql(query, conn)

    # Mostrar los primeros 5 registros del DataFrame para visualización
    print(df.head())

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    # Cerrar la conexión si fue abierta
    if conn:
        cur.close()
        conn.close()
```

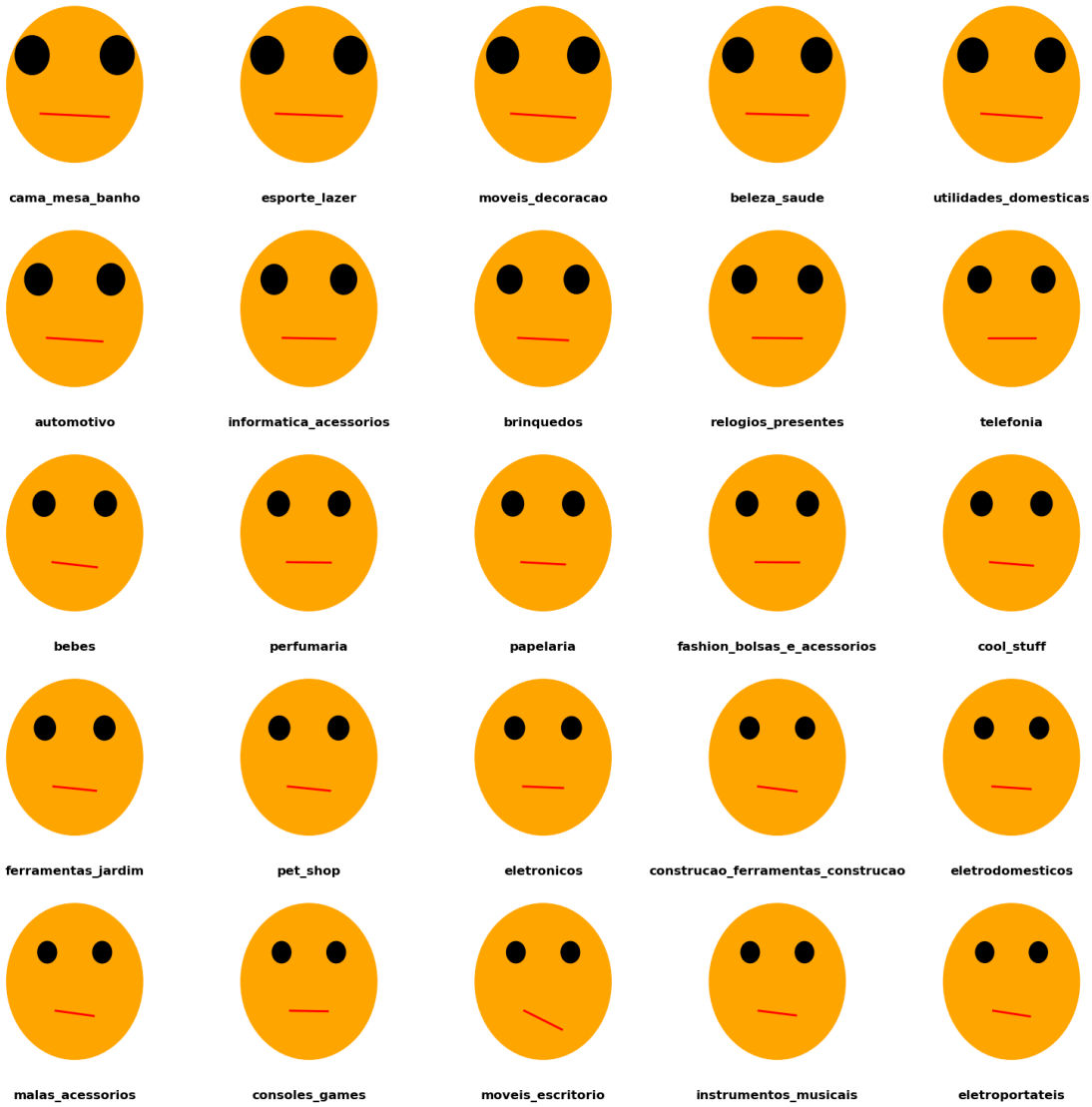
```
     anio    semestre  venta_producto  costo_envio
0  2016.0  2 Semestre         49786.0       7397.0
1  2017.0  1 Semestre       1952154.0     302009.0
```

```
2  2017.0  2 Semestre       4082714.0     663731.0
3  2018.0  1 Semestre       5608861.0     936351.0
4  2018.0  2 Semestre       1897782.0     342269.0
```

```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\2716449766.py:39: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql(query, conn)
```

[24]:
```
## Tabla 5 - olist_products: Información de los productos.

# Conectar a la base de datos PostgreSQL
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/
 ↪olist_ecommerce')

# Consulta SQL
query = """
SELECT
    product_category_name AS categoria_producto,
    COUNT(product_id) AS cnt_productos,
    ROUND(AVG(product_weight_g)) AS media_peso
FROM
    public.olist_products
WHERE
    product_category_name IS NOT NULL
GROUP BY
    product_category_name
ORDER BY
    cnt_productos DESC;
"""

df = pd.read_sql_query(query, engine)

# Normalizar las características para ajustarlas a las visualizaciones
scaler = MinMaxScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df[['media_peso',␣
 ↪'cnt_productos']]),
                             columns=['media_peso', 'cnt_productos'])

# Añadir la columna de categoría de producto nuevamente al DataFrame normalizado
df_normalized['categoria_producto'] = df['categoria_producto']

# Funcióncara personalizada del producto
def draw_custom_face(ax, media_peso, cnt_productos, label):
    # Cara
    face = plt.Circle((0.5, 0.5), 0.4, color='orange', fill=True)
    ax.add_patch(face)
```

```
    # Ojos conteo de productos
    eye_y = 0.65
    eye_x_dist = 0.15 + 0.1 * cnt_productos
    eye_size = 0.05 + 0.05 * cnt_productos
    left_eye = plt.Circle((0.5 - eye_x_dist, eye_y), eye_size, color='black')
    right_eye = plt.Circle((0.5 + eye_x_dist, eye_y), eye_size, color='black')
    ax.add_patch(left_eye)
    ax.add_patch(right_eye)

    # Boca
    mouth_y = 0.35
    mouth_width = 0.2 + 0.2 * cnt_productos
    mouth_height = -0.1 * media_peso
    ax.plot([0.5 - mouth_width / 2, 0.5 + mouth_width / 2],
            [mouth_y, mouth_y + mouth_height], color='red', linewidth=2)


    # Etiqueta de la categoría
    ax.text(0.5, -0.1, label, horizontalalignment='center', fontsize=12,␣
 ↪fontweight='bold')

    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.axis('off')

# Crear figuras para cada categoría de producto
fig, axs = plt.subplots(nrows=5, ncols=5, figsize=(15, 15))
axs = axs.flatten()

for i, (index, row) in enumerate(df_normalized.iterrows()):
    if i >= len(axs):
        break
    draw_custom_face(axs[i], row['media_peso'], row['cnt_productos'],␣
 ↪row['categoria_producto'])

plt.tight_layout()
plt.show()
```



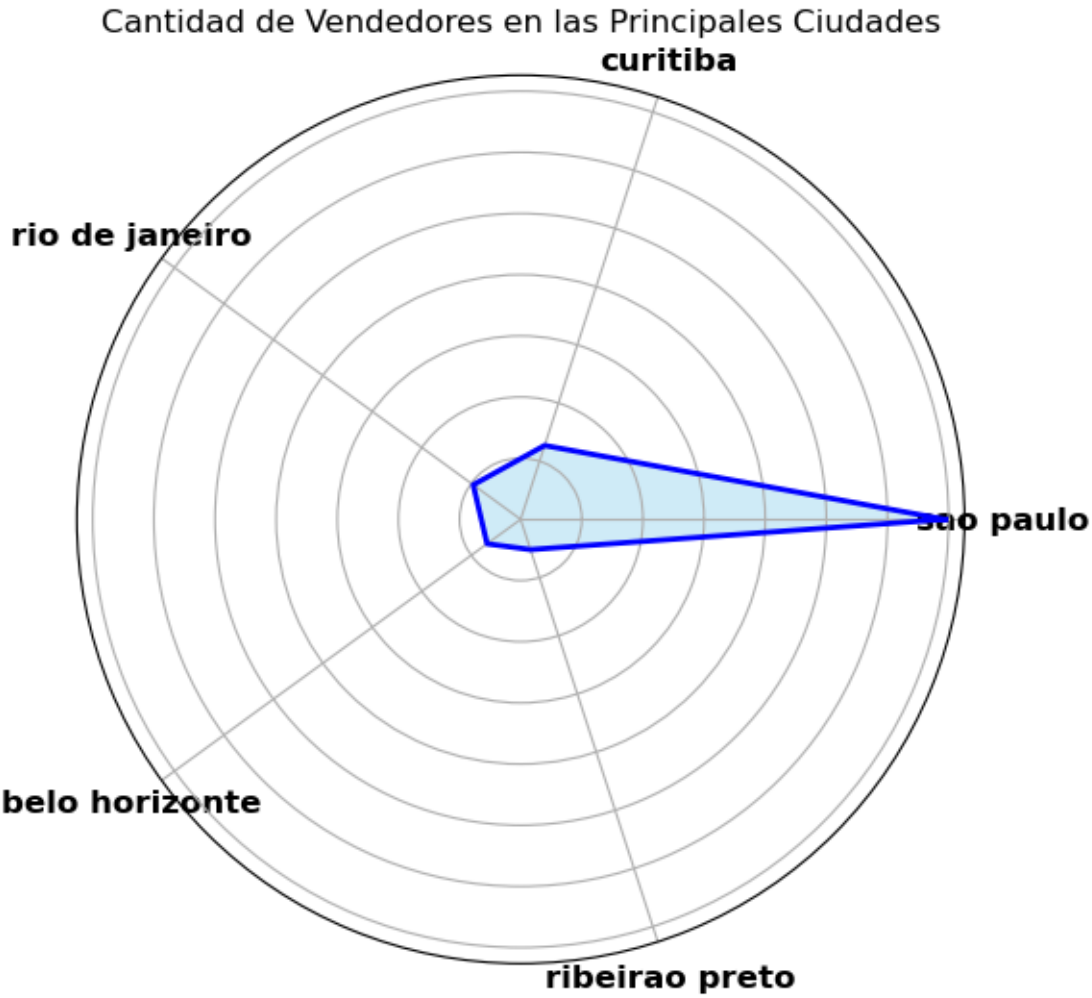
[25]:
```
"""
La cara esta compuesta por:
Cara: Tipo de categorias.
Ojos: tamaño del numero de productos asociados a la categoria
Boca: Media del peso de los productos asociados a la categoria
"""
```

[25]:
```
'\nLa cara esta compuesta por:\nCara: Tipo de categorias.\nOjos: tamaño del
numero de productos asociados a la categoria\nBoca: Media del peso de los
productos asociados a la categoria\n'
```

[27]:
```
## 6) Tabla olist_sellers: Información sobre los vendedores.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sqlalchemy import create_engine

# Crear la cadena de conexión con SQLAlchemy
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/
 ↪olist_ecommerce')

# Ejecutar la consulta SQL y cargar los resultados en un DataFrame de Pandas
query = """
SELECT
    seller_city AS ciudad,
    COUNT(seller_id) AS cnt_vendedores
FROM
    public.olist_sellers
GROUP BY
    seller_city
ORDER BY
    cnt_vendedores DESC
LIMIT 5;
"""

df = pd.read_sql_query(query, engine)

# Verificar que las columnas se cargaron correctamente
print(df.columns)

# Crear un gráfico de estrellas para las cinco ciudades con más vendedores
def radar_chart(df):
    labels = df['ciudad']
    num_vars = len(labels)

    # Ángulos para cada eje en el gráfico de radar
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()

    # La primera ciudad se repite para cerrar el gráfico de radar
    stats = df['cnt_vendedores'].tolist()
    stats += stats[:1]
    angles += angles[:1]

    # Inicializar el gráfico de radar
    fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

    ax.fill(angles, stats, color='skyblue', alpha=0.4)
```

```
    ax.plot(angles, stats, color='blue', linewidth=2)

    # Ajustar los atributos del gráfico
    ax.set_yticklabels([])
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(labels, size=12, fontweight='bold')

    plt.title('Cantidad de Vendedores en las Principales Ciudades')
    plt.show()

# Crear el gráfico de radar
radar_chart(df)
```

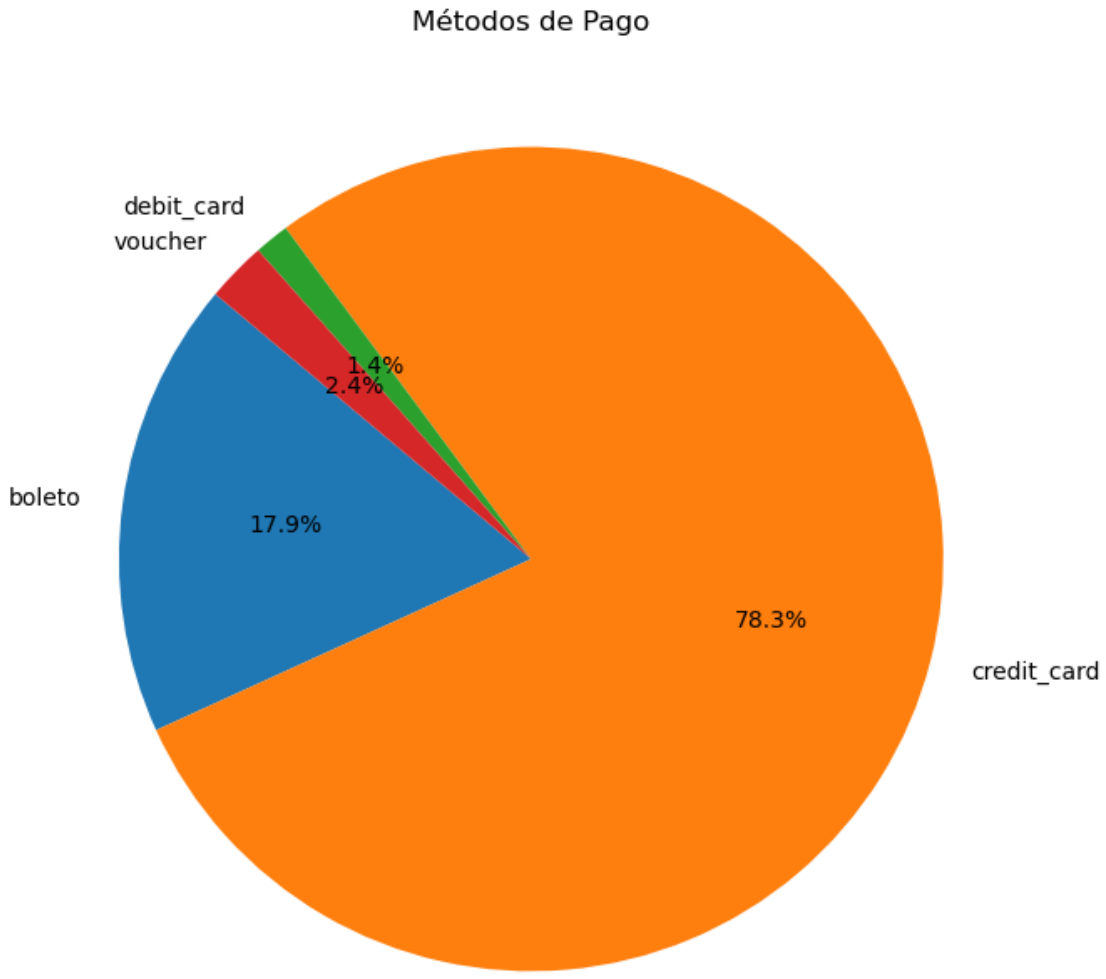
```
Index(['ciudad', 'cnt_vendedores'], dtype='object')
```



[28]:
```
## 7) Tabla olist_order_payments: Información de pagos.


# Conectar a la base de datos PostgreSQL y ejecutar la consulta
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar una consulta
    cur = conn.cursor()

    # Consulta SQL para obtener el método de pago y el valor de los pagos
    query = """
    SELECT DISTINCT
        payment_type AS metodo_pago,
        ROUND(SUM(payment_value)) AS valor_pagos
    FROM
        olist_order_payments
    WHERE
        payment_type <> 'not_defined'
    GROUP BY
        metodo_pago;
    """

    # Ejecutar la consulta y leer los resultados en un DataFrame de Pandas
    df = pd.read_sql_query(query, conn)

    # Cerrar la conexión
    cur.close()
    conn.close()

    # Crear el gráfico de tortas (pastel)
    plt.figure(figsize=(8, 8))
    plt.pie(df['valor_pagos'], labels=df['metodo_pago'], autopct='%1.1f%%',␣
 ↪startangle=140)
    plt.title('Métodos de Pago')
    plt.show()

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")
```

```
finally:
    # Asegurarse de que la conexión se cierra
    if conn:
        cur.close()
        conn.close()
```

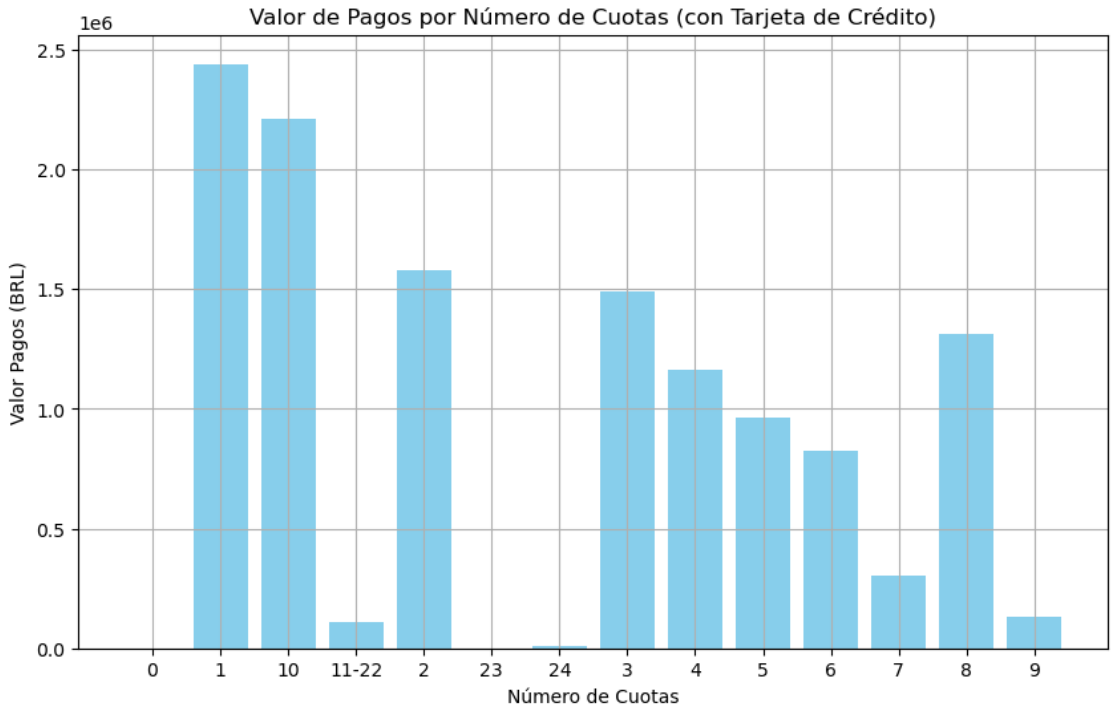
```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\3609653470.py:32: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```



[29]:
```
### Gráfico de barras Pagos Tarjeta Credito vs Numero de cuotas

# Conectar a la base de datos PostgreSQL y ejecutar la consulta
try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar una consulta
    cur = conn.cursor()

    # Consulta SQL para obtener el método de pago, número de cuotas, valor de␣
 ↪pagos y porcentaje
    query = """
    SELECT
        payment_type AS metodo_pago,
        payment_installments AS cuota,
        ROUND(SUM(payment_value)) AS valor_pagos,
        ROUND(CAST(SUM(payment_value) * 100.0 / SUM(SUM(payment_value)) OVER ()␣
 ↪AS numeric), 2) AS porcentaje
    FROM
        olist_order_payments
    WHERE
        payment_type = 'credit_card'
    GROUP BY
        metodo_pago, cuota
    ORDER BY
        cuota ASC;
    """

    # Ejecutar la consulta y leer los resultados en un DataFrame de Pandas
    df = pd.read_sql_query(query, conn)

    # Agrupar cuotas entre 11 y 22
    df['cuota_grupo'] = df['cuota'].apply(lambda x: '11-22' if 11 <= x <= 22␣
 ↪else str(x))

    # Agrupar por cuota_grupo y sumar valores de pagos
    df_grouped = df.groupby('cuota_grupo').agg({'valor_pagos': 'sum'}).
 ↪reset_index()

    # Ordenar por número de cuotas para visualización
```

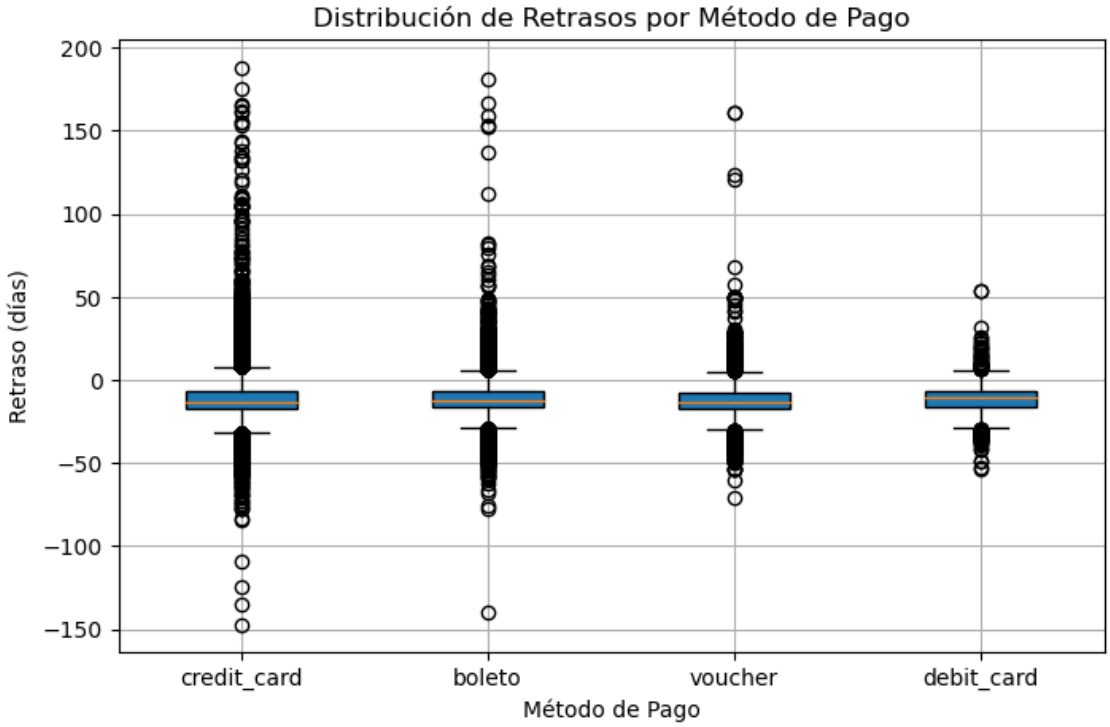
```
    df_grouped['cuota_grupo'] = pd.Categorical(df_grouped['cuota_grupo'],
                                               ␣
 ↪categories=sorted(df_grouped['cuota_grupo'], key=lambda x: int(x.
 ↪split('-')[0]) if '-' in x else int(x)),
                                                ordered=True)

    # Crear el gráfico de barras
    plt.figure(figsize=(10, 6))
    plt.bar(df_grouped['cuota_grupo'], df_grouped['valor_pagos'],␣
 ↪color='skyblue')

    # Añadir etiquetas y título
    plt.xlabel('Número de Cuotas')
    plt.ylabel('Valor Pagos (BRL)')
    plt.title('Valor de Pagos por Número de Cuotas (con Tarjeta de Crédito)')
    plt.grid(True)

    # Mostrar el gráfico
    plt.show()

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    # Asegurarse de que la conexión se cierra
    if conn:
        cur.close()
        conn.close()
```

```
C:\Users\jmelo\AppData\Local\Temp\ipykernel_71568\3525691494.py:35: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql_query(query, conn)
```



[30]:
```
### Métodos de Pago y Retrasos de Entrega

query_payments_delay = """
SELECT p.payment_type,
       o.order_delivered_customer_date, o.order_estimated_delivery_date
FROM olist_order_payments p
JOIN olist_orders o ON p.order_id = o.order_id
WHERE o.order_delivered_customer_date IS NOT NULL;
"""

# Ejecutar la consulta SQL y cargar los resultados en un DataFrame
df_payments = pd.read_sql_query(query_payments_delay, con=engine)

# Calcular el retraso en la entrega (en días)
df_payments['delay'] = (
    pd.to_datetime(df_payments['order_delivered_customer_date']) - pd.
 ↪to_datetime(df_payments['order_estimated_delivery_date'])
).dt.days

# Gráfico de caja y bigotes para mostrar la distribución de los retrasos por␣
 ↪método de pago
plt.figure(figsize=(8, 5))
```

```
plt.boxplot([df_payments[df_payments['payment_type'] ==␣
 ↪payment]['delivery_delay'] for payment in df_payments['payment_type'].
 ↪unique()],
            labels=df_payments['payment_type'].unique(), patch_artist=True)

plt.title('Distribución de Retrasos por Método de Pago')
plt.xlabel('Método de Pago')
plt.ylabel('Retraso (días)')
plt.grid(True)
plt.show()
```



[60]:
```
"""
Em el gráfico de caja y bigotes se observa la distribución de los retrasos en␣
 ↪la entrega según el método de pago,
en promedio todos los métodos tienen entregas anticipadas, no se observan␣
 ↪diferencias significativas entre
los métodos de pago. Existen valores atípicos notables, con algunas entregas␣
 ↪extremadamente anticipadas o muy tardías,
lo que sugiere variabilidad en los tiempos de entrega independientemente del␣
 ↪método de pago.
"""
```

[60]:
```
'\nEm el gráfico de caja y bigotes se observa la distribución de los retrasos en
la entrega según el método de pago, \nen promedio todos los métodos tienen
entregas anticipadas, no se observan diferencias significativas entre \nlos
métodos de pago. Existen valores atípicos notables, con algunas entregas
extremadamente anticipadas o muy tardías, \nlo que sugiere variabilidad en los
tiempos de entrega independientemente del método de pago.\n'
```

[31]:
```
## 8) Tabla olist_order_reviews: Reseñas de los pedidos.


# Definir los detalles de la conexión
user = 'postgres'
password = 'postgres'
host = 'localhost'
port = '5432'
dbname = 'olist_ecommerce'

# Crear el engine usando SQLAlchemy
connection_string = f'postgresql://{user}:{password}@{host}:{port}/{dbname}'
engine = create_engine(connection_string)

# Definir y ejecutar la consulta SQL
query1 = """
SELECT
    review_score as Calificacion,
    COUNT(review_id) AS Cantidad,
    ROUND(COUNT(review_id) * 100.0 / (SELECT COUNT(*) FROM␣
 ↪olist_order_reviews)) AS Porcentaje
FROM
    olist_order_reviews
GROUP BY
    review_score
ORDER BY
    review_score;
"""

# Ejecutar la consulta y cargar los datos en un DataFrame
opiniones_clientes = pd.read_sql_query(query1, engine)

# Mostrar los datos
print(opiniones_clientes)
```

```
   calificacion  cantidad  porcentaje
0             1     11282        11.0
1             2      3114         3.0
2             3      8097         8.0
3             4     19007        19.0
4             5     56910        58.0
```

[37]:
```
"""
El 58% de las calificaciones otorgadas por los clientes corresponde a la␣
 ↪puntuación más alta (5), lo que indica un alto nivel de satisfacción.
Un 11% de los clientes ha calificado con la puntuación más baja (0), lo que␣
 ↪podría reflejar un grupo insatisfecho o con problemas significativos en su␣
 ↪experiencia de compra.
En total, solo el 14% de las calificaciones se encuentran entre 1 y 2, lo que␣
 ↪muestra que los clientes tienden a polarizar sus opiniones, otorgando muy␣
 ↪pocas calificaciones bajas (entre 1 y 2).

"""
```

[37]:
```
'\nEl 58% de las calificaciones otorgadas por los clientes corresponde a la
puntuación más alta (5), lo que indica un alto nivel de satisfacción.\nUn 11% de
los clientes ha calificado con la puntuación más baja (0), lo que podría
reflejar un grupo insatisfecho o con problemas significativos en su experiencia
de compra.\nEn total, solo el 14% de las calificaciones se encuentran entre 1 y
2, lo que muestra que los clientes tienden a polarizar sus opiniones, otorgando
muy pocas calificaciones bajas (entre 1 y 2).\n\n'
```

[32]:
```
###  Pago Total
total_payment = pd.read_sql_query("""
    SELECT
        SUM(payment_value) AS PagoTotal
    FROM
        olist_order_payments;
""", con=engine)

print(total_payment)
```

```
     pagototal
0  1.600887e+07
```

[33]:
```
# Union de Variables
```

[35]:
```
### Conexión BD
user = 'postgres'
password = 'postgres'
host = 'localhost'
port = '5432'
dbname = 'olist_ecommerce'

### Crear el engine usando SQLAlchemy
connection_string = f'postgresql://{user}:{password}@{host}:{port}/{dbname}'
engine = create_engine(connection_string)

### Consulta SQL
query1 = """
```

```
SELECT
    o.order_id,                              -- ID del pedido (olist_orders)
    o.order_delivered_customer_date,         -- Fecha de entrega al cliente␣
 ↪(olist_orders)
    o.order_estimated_delivery_date,         -- Fecha estimada de entrega␣
 ↪(olist_orders)
    c.customer_zip_code_prefix AS customer_zip,  -- Código postal del cliente␣
 ↪(olist_order_customers)
    s.seller_zip_code_prefix AS seller_zip,      -- Código postal del vendedor␣
 ↪(olist_sellers)
    ST_Distance(c.location, s.location) AS distancia, -- Distancia entre el␣
 ↪cliente y el vendedor (calculado con PostGIS)
    r.review_score,                          -- Puntuación de la reseña del␣
 ↪pedido (olist_order_reviews)
    p.payment_type,                          -- Tipo de pago utilizado␣
 ↪(olist_order_payments)
    oi.price                                 -- Precio del producto␣
 ↪(olist_order_items)
FROM
    olist_orders o
JOIN
    olist_order_customers c ON o.customer_id = c.customer_id
JOIN
    olist_order_items oi ON o.order_id = oi.order_id
JOIN
    olist_sellers s ON oi.seller_id = s.seller_id
JOIN
    olist_order_reviews r ON o.order_id = r.order_id
JOIN
    olist_order_payments p ON o.order_id = p.order_id
WHERE
    o.order_delivered_customer_date IS NOT NULL
LIMIT 5;
"""

### Ejecutar la consulta y cargar los datos en un DataFrame
union_datos = pd.read_sql_query(query1, engine)

### Mostrar los datos
print(union_datos)
```

```
                          order_id order_delivered_customer_date  \
0  b81ef226f3fe1789b1e8b2acac839d17           2018-05-09 17:36:51
1  a9810da82917af2d9aefd1278f1dcfa0           2018-06-29 20:32:09
2  25e8ea4e93396b6fa0d3dd708e76c1bd           2017-12-18 17:24:41
3  ba78997921bbcdc1373bb41e913ab953           2017-12-21 01:35:51
4  ba78997921bbcdc1373bb41e913ab953           2017-12-21 01:35:51
```

```
  order_estimated_delivery_date  customer_zip  seller_zip      distancia  \
0                    2018-05-22         39801       13321  845683.597248
1                    2018-07-16          2422        4660   21319.322471
2                    2018-01-04          2652        9015   26790.100620
3                    2018-01-04         36060       13405  458723.781313
4                    2018-01-04         36060       13405  458723.781313

   review_score payment_type  price
0             1  credit_card  79.80
1             5  credit_card  17.00
2             5  credit_card  56.99
3             5  credit_card  89.90
4             5  credit_card  89.90
```

[38]:
```
# Medidas Estadísticas
```

[39]:
```
## Resumen estadístico de las variables numéricas
print("Resumen Estadístico:")
print(df.describe())
```

```
Resumen Estadístico:
           cuota   valor_pagos  porcentaje
count  24.000000  2.400000e+01   24.000000
mean   11.708333  5.225868e+05    4.165417
std     7.369025  7.747085e+05    6.177154
min     0.000000  1.890000e+02    0.000000
25%     5.750000  2.171000e+03    0.017500
50%    11.500000  2.305300e+04    0.180000
75%    17.250000  1.011858e+06    8.065000
max    24.000000  2.440445e+06   19.460000
```

[40]:
```
## Análisis Estadístico de la Distribución de Calificaciones de los Clientes
## Importar NumPy
import numpy as np

## Extraer los valores de la columna 'cantidad'
review_counts = opiniones_clientes['cantidad'].values

## Calcular la media
mean = np.mean(review_counts)

## Calcular la mediana
median = np.median(review_counts)

## Calcular la desviación estándar
std_dev = np.std(review_counts)
```

```
## Mostrar los resultados
print(f"Media: {mean}")
print(f"Mediana: {median}")
print(f"Desviación Estándar: {std_dev}")
```

```
Media: 19682.0
Mediana: 11282.0
Desviación Estándar: 19316.575566077958
```

[41]:
```
"""
El análisis calcula tres métricas estadísticas de las calificaciones recibidas␣
 ↪en cada categoría (0 a 5) en la plataforma de e-commerce: entonces:
a) La Media o promedio de calificaciones es 19,682 indica que cada categoría␣
 ↪recibe una calificación.
b) La mediana es 11,282, lo que significa que la mitad de las categorías tienen␣
 ↪un número de calificaciones menor o igual, esto sugiere que,
algunas calificaciones tienen una cantidad significativamente mayor de␣
 ↪opiniones, lo cual está respaldado por la concentración de altas␣
 ↪calificaciones
c) La desviación estándar es 19,316.58, lo que indica una gran variabilidad en␣
 ↪el número de calificaciones recibidas por cada categoría,
esto significa que no todas las calificaciones tienen cantidades similares y␣
 ↪algunas categorías están por encima del promedio,
mientras que otras tienen menos participación.
"""
```

[41]:
```
'\nEl análisis calcula tres métricas estadísticas de las calificaciones
recibidas en cada categoría (0 a 5) en la plataforma de e-commerce:
entonces:\na) La Media o promedio de calificaciones es 19,682 indica que cada
categoría recibe una calificación.\nb) La mediana es 11,282, lo que significa
que la mitad de las categorías tienen un número de calificaciones menor o igual,
esto sugiere que,\nalgunas calificaciones tienen una cantidad significativamente
mayor de opiniones, lo cual está respaldado por la concentración de altas
calificaciones\nc) La desviación estándar es 19,316.58, lo que indica una gran
variabilidad en el número de calificaciones recibidas por cada categoría, \nesto
significa que no todas las calificaciones tienen cantidades similares y algunas
categorías están por encima del promedio, \nmientras que otras tienen menos
participación.\n'
```

[48]:
```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine

# Crear la cadena de conexión con SQLAlchemy
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/
 ↪olist_ecommerce')
```

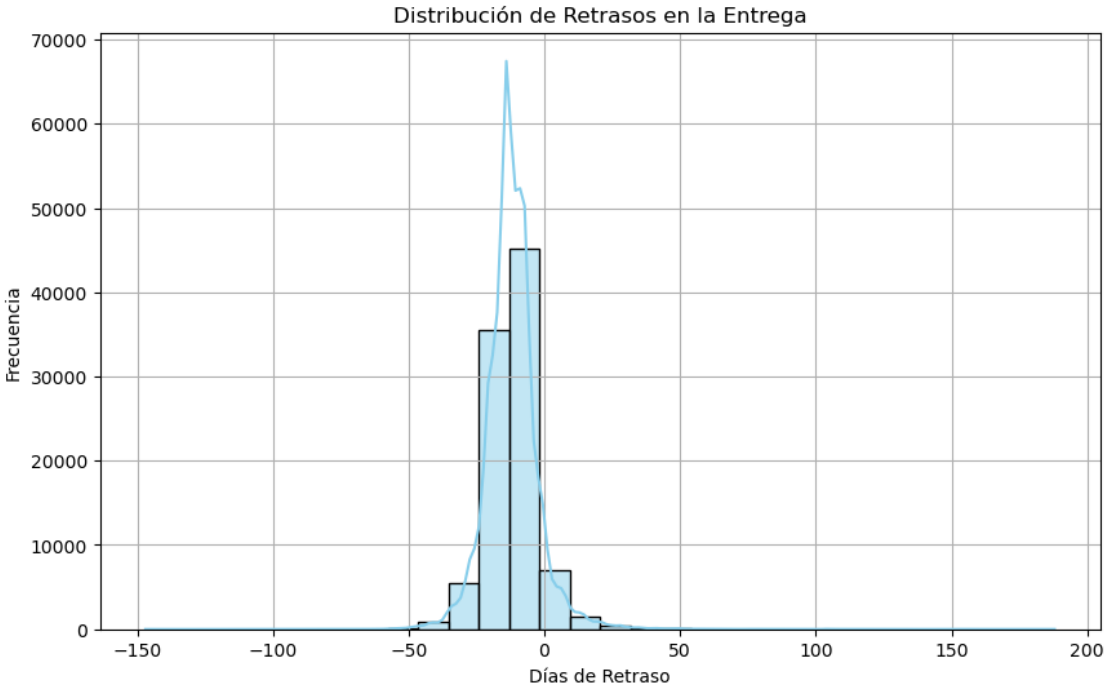
```
# Cargar los datos de la tabla de órdenes, que debe contener las fechas de␣
 ↪entrega
df = pd.read_sql_query('SELECT * FROM olist_orders', con=engine)

# Verificar las columnas presentes
print(df.columns)

# Cálculo del retraso en la entrega (en días)
if 'order_delivered_customer_date' in df.columns and␣
 ↪'order_estimated_delivery_date' in df.columns:
    df['delivery_delay'] = (
        pd.to_datetime(df['order_delivered_customer_date']) - pd.
 ↪to_datetime(df['order_estimated_delivery_date'])
    ).dt.days

    # Medidas estadísticas descriptivas: Media y desviación estándar de los␣
 ↪retrasos
    media_retraso = np.mean(df['delivery_delay'])
    desviacion_retraso = np.std(df['delivery_delay'])

    print(f"Media de retraso en días: {media_retraso}")
    print(f"Desviación estándar de retraso: {desviacion_retraso}")
else:
    print("Las columnas necesarias no están presentes en el DataFrame.")
```

```
Index(['order_id', 'customer_id', 'order_status', 'order_purchase_timestamp',
       'order_approved_at', 'order_delivered_carrier_date',
       'order_delivered_customer_date', 'order_estimated_delivery_date'],
      dtype='object')
Media de retraso en días: -11.876881296902857
Desviación estándar de retraso: 10.183801336071676
```

[ ]:
```
"""
Media de retraso en días -11.876881296902857: El valor negativo de la media␣
 ↪sugiere que, en promedio, los pedidos fueron entregados 11.88 días antes
de la fecha estimada de entrega. Un valor negativo en el contexto del retraso␣
 ↪indica que los pedidos fueron entregados más rápido de lo esperado.

Desviación estándar de retraso 10.183801336071676: La desviación estándar de␣
 ↪aproximadamente 10.18 días indica que hay una variabilidad significativa
en los tiempos de entrega respecto a la fecha estimada, significa que, aunque␣
 ↪en promedio los pedidos se entregaron antes, hay atipicos  en los tiempos
de entrega.
"""
```

[50]:
```
# Visualización de la Distribución de 'delivery_delay'
plt.figure(figsize=(10, 6))
```

```
sns.histplot(df['delivery_delay'], bins=30, kde=True, color='skyblue')
plt.title('Distribución de Retrasos en la Entrega')
plt.xlabel('Días de Retraso')
plt.ylabel('Frecuencia')
plt.grid(True)
plt.show()
```



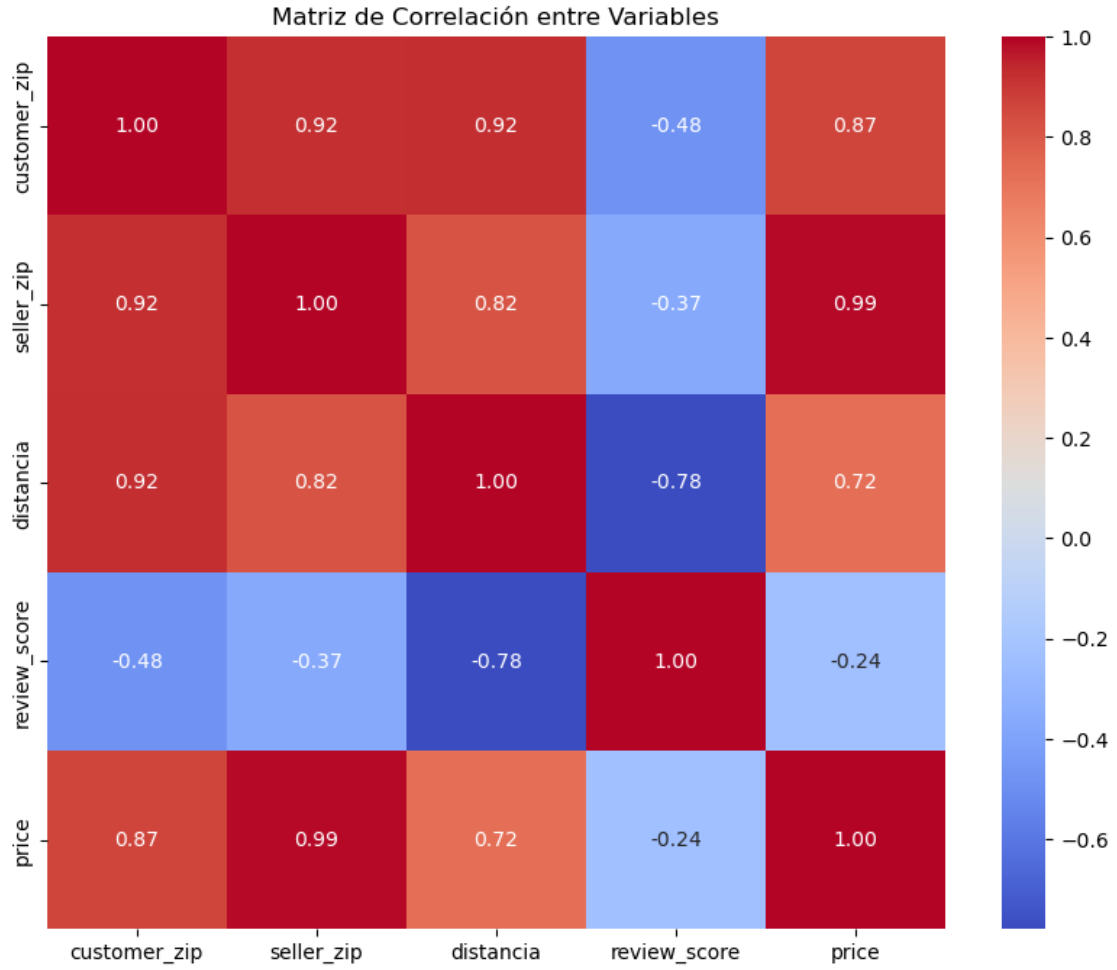
[54]:
```
# Regresion Lineal y Logistica
```

[ ]:
```
## Verifiquemos la correlacion
```

[71]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine

# Conectar a la base de datos PostgreSQL
engine = create_engine('postgresql+psycopg2://postgres:postgres@localhost:5432/
 ↪olist_ecommerce')

# Ejecutar la consulta SQL y cargar los resultados en un DataFrame de Pandas
query = """
SELECT
```

```
    o.order_id,                              -- ID del pedido (olist_orders)
    o.order_delivered_customer_date,         -- Fecha de entrega al cliente␣
 ↪(olist_orders)
    o.order_estimated_delivery_date,         -- Fecha estimada de entrega␣
 ↪(olist_orders)
    c.customer_zip_code_prefix AS customer_zip,  -- Código postal del cliente␣
 ↪(olist_order_customers)
    s.seller_zip_code_prefix AS seller_zip,      -- Código postal del vendedor␣
 ↪(olist_sellers)
    ST_Distance(c.location, s.location) AS distancia, -- Distancia entre el␣
 ↪cliente y el vendedor (calculado con PostGIS)
    r.review_score,                          -- Puntuación de la reseña del␣
 ↪pedido (olist_order_reviews)
    p.payment_type,                          -- Tipo de pago utilizado␣
 ↪(olist_order_payments)
    oi.price                                 -- Precio del producto␣
 ↪(olist_order_items)
FROM
    olist_orders o
JOIN
    olist_order_customers c ON o.customer_id = c.customer_id
JOIN
    olist_order_items oi ON o.order_id = oi.order_id
JOIN
    olist_sellers s ON oi.seller_id = s.seller_id
JOIN
    olist_order_reviews r ON o.order_id = r.order_id
JOIN
    olist_order_payments p ON o.order_id = p.order_id
WHERE
    o.order_delivered_customer_date IS NOT NULL
LIMIT 5;
"""

# Cargar los datos en un DataFrame
df = pd.read_sql_query(query, engine)

# Filtrar solo las columnas numéricas para la matriz de correlación
df_numeric = df.select_dtypes(include=[np.number])

# Manejo de valores NaN (eliminación en este caso)
df_numeric = df_numeric.dropna()

# Calcular la matriz de correlación
corr_matrix = df_numeric.corr()
```

```
# Visualizar la matriz de correlación
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de Correlación entre Variables')
plt.show()
```



[58]:
```
"""
1) Hay una alta correlación positiva entre el código postal del cliente y el␣
 ↪del vendedor. por ello se identifica que que los clientes y vendedores
tienden a estar geográficamente cercanos

2) Hay una correlación negativa significativa entre la distancia y la␣
 ↪puntuación de la reseña. Los clientes que están más lejos del vendedor
tienden a dar peores puntuaciones, posiblemente debido a tiempos de entrega más␣
 ↪largos o mayores costos de envío.
price vs seller_zip (0.99):
```

```
3) El precio tiene una correlación casi perfecta con el código postal del␣
 ↪vendedor, ciertos vendedores en áreas específicas tienden a vender productos
a precios similares por ello se indica que los precios están asociados a␣
 ↪ciertas regiones.
"""
```

[58]: '\n1) Hay una alta correlación positiva entre el código postal del cliente y el del vendedor. por ello se identifica que que los clientes y vendedores \ntienden a estar geográficamente cercanos\n\n2) Hay una correlación negativa significativa entre la distancia y la puntuación de la reseña. Los clientes que están más lejos del vendedor \ntienden a dar peores puntuaciones, posiblemente debido a tiempos de entrega más largos o mayores costos de envío.\nprice vs seller_zip (0.99):\n\n3) El precio tiene una correlación casi perfecta con el código postal del vendedor, ciertos vendedores en áreas específicas tienden a vender productos \na precios similares por ello se indica que los precios están asociados a ciertas regiones.\n'

regresion_80903390

September 27, 2024

[38]:
```
# REGRESIÓN
```

[39]:
```
## 1) Seleccion de Variables
```

[40]:
```
'''
Debido a la correlación alta entre algunas variables de la matriz␣
 ↪proporcionada, podemos seleccionar las siguientes:

Variables independientes :

A) seller_zip (Código postal del vendedor)
B) price (Precio del producto)
C) delivery_delay (Retraso en la entrega)

Variable dependiente:
 review_score (Puntuación de la reseña)
'''
```

[40]:
```
'\nDebido a la correlación alta entre algunas variables de la matriz
proporcionada, podemos seleccionar las siguientes:\n\nVariables independientes
:\n\nA) seller_zip (Código postal del vendedor)\nB) price (Precio del
producto)\nC) delivery_delay (Retraso en la entrega)\n\nVariable dependiente:\n
review_score (Puntuación de la reseña)\n'
```

[41]:
```
## 2) Importar Bibliotecas y Librerias

import pandas as pd
import psycopg2
import seaborn as sns
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

[42]:
```
## 3) Consulta Postgres
"""
SELECT
```

```
    o.order_id,                             -- ID del pedido
    o.order_status,                         -- Estado del pedido
    o.order_purchase_timestamp,             -- Fecha de compra
    o.order_approved_at,                    -- Fecha de aprobación
    o.order_delivered_carrier_date,         -- Fecha de entrega al transportista
    o.order_delivered_customer_date,        -- Fecha de entrega al cliente
    o.order_estimated_delivery_date,        -- Fecha estimada de entrega
    c.customer_id,                          -- ID del cliente
    c.customer_zip_code_prefix,             -- Código postal del cliente
    s.seller_id,                            -- ID del vendedor
    s.seller_zip_code_prefix,               -- Código postal del vendedor
    oi.product_id,                          -- ID del producto
    oi.price,                               -- Precio del producto
    oi.freight_value,                       -- Valor del flete
    r.review_score,                         -- Puntuación de la reseña
    p.payment_type,                         -- Tipo de pago
    p.payment_installments,                 -- Número de cuotas
    p.payment_value,                        -- Valor del pago
    op.product_category_name,               -- Categoría del producto
    og.geolocation_lat,                     -- Latitud geográfica
    og.geolocation_lng                      -- Longitud geográfica
FROM
    olist_orders o
JOIN
    olist_order_customers c ON o.customer_id = c.customer_id
JOIN
    olist_order_items oi ON o.order_id = oi.order_id
JOIN
    olist_products op ON oi.product_id = op.product_id
JOIN
    olist_sellers s ON oi.seller_id = s.seller_id
JOIN
    olist_order_reviews r ON o.order_id = r.order_id
JOIN
    olist_order_payments p ON o.order_id = p.order_id
JOIN
    olist_geolocation og ON s.seller_zip_code_prefix = og.zip_code_prefix
WHERE
    c.customer_zip_code_prefix = og.zip_code_prefix AND
    o.order_delivered_customer_date IS NOT NULL
LIMIT 5;
"""
```

[42]:
```
'\nSELECT \n    o.order_id,                             -- ID del pedido\n
o.order_status,                         -- Estado del pedido\n
o.order_purchase_timestamp,             -- Fecha de compra\n
o.order_approved_at,                    -- Fecha de aprobación\n
```

```
o.order_delivered_carrier_date,         -- Fecha de entrega al transportista\n
o.order_delivered_customer_date,        -- Fecha de entrega al cliente\n
o.order_estimated_delivery_date,        -- Fecha estimada de entrega\n
c.customer_id,                          -- ID del cliente\n
c.customer_zip_code_prefix,             -- Código postal del cliente\n
s.seller_id,                            -- ID del vendedor\n
s.seller_zip_code_prefix,               -- Código postal del vendedor\n
oi.product_id,                          -- ID del producto\n    oi.price,
-- Precio del producto\n    oi.freight_value,                      -- Valor del
flete\n    r.review_score,                          -- Puntuación de la reseña\n
p.payment_type,                         -- Tipo de pago\n
p.payment_installments,                 -- Número de cuotas\n
p.payment_value,                        -- Valor del pago\n
op.product_category_name,               -- Categoría del producto\n
og.geolocation_lat,                     -- Latitud geográfica\n
og.geolocation_lng                      -- Longitud geográfica\nFROM\n
olist_orders o\nJOIN \n    olist_order_customers c ON o.customer_id =
c.customer_id\nJOIN \n    olist_order_items oi ON o.order_id = oi.order_id\nJOIN
\n    olist_products op ON oi.product_id = op.product_id \nJOIN \n
olist_sellers s ON oi.seller_id = s.seller_id\nJOIN \n    olist_order_reviews r
ON o.order_id = r.order_id\nJOIN \n    olist_order_payments p ON o.order_id =
p.order_id\nJOIN \n    olist_geolocation og ON s.seller_zip_code_prefix =
og.zip_code_prefix\nWHERE\n    c.customer_zip_code_prefix = og.zip_code_prefix
AND\n    o.order_delivered_customer_date IS NOT NULL\nLIMIT 5;\n'
```

[55]:
```
## Paso 4)  Conectar a Base de datos y Carga
'''
seller_zip: El código postal del vendedor.
price: El precio del producto.
delivery_delay: El retraso en la entrega expresado en un formato de duración␣
 ↪(days hh:mm:ss).
review_score: La puntuación de la reseña.
'''

try:
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar la consulta
    cur = conn.cursor()
```

```
    # Ejecutar la consulta para obtener los datos de la vista
    cur.execute("SELECT seller_zip_code_prefix as seller_zip, price,␣
 ↪(order_delivered_customer_date - order_estimated_delivery_date) as␣
 ↪delivery_delay, review_score FROM olist_ecommerce;")

    # Convertir los resultados a un DataFrame de pandas
    df = pd.DataFrame(cur.fetchall(), columns=['seller_zip', 'price',␣
 ↪'delivery_delay', 'review_score'])

    # Mostrar algunas filas para verificar los datos
    print(df.head())

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")

finally:
    # Cerrar la conexión si fue abierta
    if conn:
        cur.close()
        conn.close()
        print("Conexión cerrada.")
```

```
   seller_zip  price     delivery_delay  review_score
0       14940   59.9 -28 days +17:53:43             3
1       14940   59.9 -28 days +17:53:43             3
2       14940   59.9 -28 days +17:53:43             3
3       14940   59.9 -28 days +17:53:43             3
4       14940   59.9 -28 days +17:53:43             3
Conexión cerrada.
```

[56]:
```
#### Convertir 'delivery_delay' a días como número flotante
df['delivery_delay'] = df['delivery_delay'].dt.days

#### Verificar los cambios
print(df.head())
```

```
   seller_zip  price  delivery_delay  review_score
0       14940   59.9             -28             3
1       14940   59.9             -28             3
2       14940   59.9             -28             3
3       14940   59.9             -28             3
4       14940   59.9             -28             3
```

[44]:
```
## Paso 5.  Usando la función train_test_split para dividir los datos en un 70%␣
 ↪para entrenamiento y 30% para prueba:'''
```

[57]:
```
# Dividir los datos en características (X) y variable dependiente (y)
X = df[['seller_zip', 'price', 'delivery_delay']]
```

```
y = df['review_score']

# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Verificar las dimensiones de los conjuntos divididos
print("Tamaño del conjunto de entrenamiento:", X_train.shape)
print("Tamaño del conjunto de prueba:", X_test.shape)
```

```
Tamaño del conjunto de entrenamiento: (4167, 3)
Tamaño del conjunto de prueba: (1786, 3)
```

[46]:
```
# Paso 4. Modelo de regresión lineal y entrenamiento
```

[58]:
```
## Verificar tipos de datos
print(X_train.dtypes)

### Si alguna columna tiene tipo 'object', convertir a numérico (ejemplo: con␣
 ↪el código postal del vendedor)
X_train['seller_zip'] = pd.to_numeric(X_train['seller_zip'], errors='coerce')
X_test['seller_zip'] = pd.to_numeric(X_test['seller_zip'], errors='coerce')

### Verificar si existen valores nulos
print(X_train.isnull().sum())

### Opcional: Imputar valores nulos o eliminar filas/columnas con valores nulos
X_train = X_train.fillna(0)
X_test = X_test.fillna(0)
```

```
seller_zip          int64
price             float64
delivery_delay      int64
dtype: object
seller_zip        0
price             0
delivery_delay    0
dtype: int64
```

[48]:
```
# Convertir 'delivery_delay' a días como número flotante
X_train['delivery_delay'] = X_train['delivery_delay'].dt.days
X_test['delivery_delay'] = X_test['delivery_delay'].dt.days

# Verificar nuevamente los tipos de datos
print(X_train.dtypes)
```

```
seller_zip          int64
price             float64
delivery_delay      int64
```

```
dtype: object
```

[59]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Instanciar el modelo de regresión lineal
model = LinearRegression()

# Ajustar el modelo a los datos de entrenamiento
model.fit(X_train, y_train)

# Realizar predicciones sobre el conjunto de prueba
y_pred = model.predict(X_test)

# Calcular el Error Cuadrático Medio (MSE) y el coeficiente de determinación␣
 ↪(R²)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Mostrar los resultados
print(f"Error Cuadrático Medio (MSE): {mse}")
print(f"Coeficiente de determinación (R²): {r2}")
```

```
Error Cuadrático Medio (MSE): 1.0592313909535767
Coeficiente de determinación (R²): 0.1846004312287436
```

[60]:
```python
'''
Error Cuadrático Medio (MSE): 1.05921390953767, indica que en promedio el␣
 ↪modelo está en un error cuadrático medio de 1.06 puntos en la review_score.
El coeficiente de Determinación (R²): 0.1846004312287436, indica qué proporción␣
 ↪de la varianza en la variable dependiente (review_score)
es explicada por las variables independientes (seller_zip, price,␣
 ↪delivery_delay). Un valor de R² de 0.18 significa que el 18.46% de la␣
 ↪variación
en las puntuaciones de las reseñas es explicada por el modelo.
Esto indica que el modelo no está capturando bien la relación entre las␣
 ↪características y la variable objetivo.
'''
```

[60]:
```
' \nError Cuadrático Medio (MSE): 1.05921390953767, indica que en promedio el
modelo está en un error cuadrático medio de 1.06 puntos en la review_score.\nEl
coeficiente de Determinación (R²): 0.1846004312287436, indica qué proporción de
la varianza en la variable dependiente (review_score) \nes explicada por las
variables independientes (seller_zip, price, delivery_delay). Un valor de R² de
0.18 significa que el 18.46% de la variación \nen las puntuaciones de las
reseñas es explicada por el modelo. \nEsto indica que el modelo no está
capturando bien la relación entre las características y la variable objetivo.\n'
```

[61]:
```
## Paso 5. Ajustar el modelo a los datos de entrenamiento
model.fit(X_train, y_train)

### Obtener los coeficientes de la regresión
coef = model.coef_
intercept = model.intercept_

print("Coeficientes de regresión:", coef)
print("Intercepto:", intercept)
```

```
Coeficientes de regresión: [-5.07669123e-06  2.49602435e-03 -5.59793258e-02]
Intercepto: 3.602161411576974
```

[62]:
```
'''
Los coeficientes sugieren que las tres variables independientes:ubicación del␣
 ↪vendedor, precio, y retraso en la entrega. Tienen un impacto poco␣
 ↪signifiativo
en las reseñas. La variable con el mayor impacto es el delivery_delay, lo que␣
 ↪resalta la importancia de la entrega puntual en la satisfacción del cliente.
'''
```

[62]:
```
'\nLos coeficientes sugieren que las tres variables independientes:ubicación del
vendedor, precio, y retraso en la entrega. Tienen un impacto poco
signifiativo\nen las reseñas. La variable con el mayor impacto es el
delivery_delay, lo que resalta la importancia de la entrega puntual en la
satisfacción del cliente.\n'
```

[50]:
```
## Paso 6. Evaluacion del Modelo

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calcular el Error Cuadrático Medio (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Error Cuadrático Medio (MSE): {mse}")

# Calcular el coeficiente de determinación (R²)
r2 = r2_score(y_test, y_pred)
print(f"Coeficiente de determinación (R²): {r2}")
```

```
Error Cuadrático Medio (MSE): 1.0565704339797233
Coeficiente de determinación (R²): 0.1741225150664465
```

[ ]:
```
''' El MSE se mantenga constante sugiere que el modelo tiene un rendimiento␣
 ↪predecible,no óptimo, en cuanto a la precisión de las predicciones.
Sin embargo, debido al bajo R², las predicciones del modelo no son␣
 ↪particularmente útiles para explicar la variabilidad en las calificaciones.
'''
```

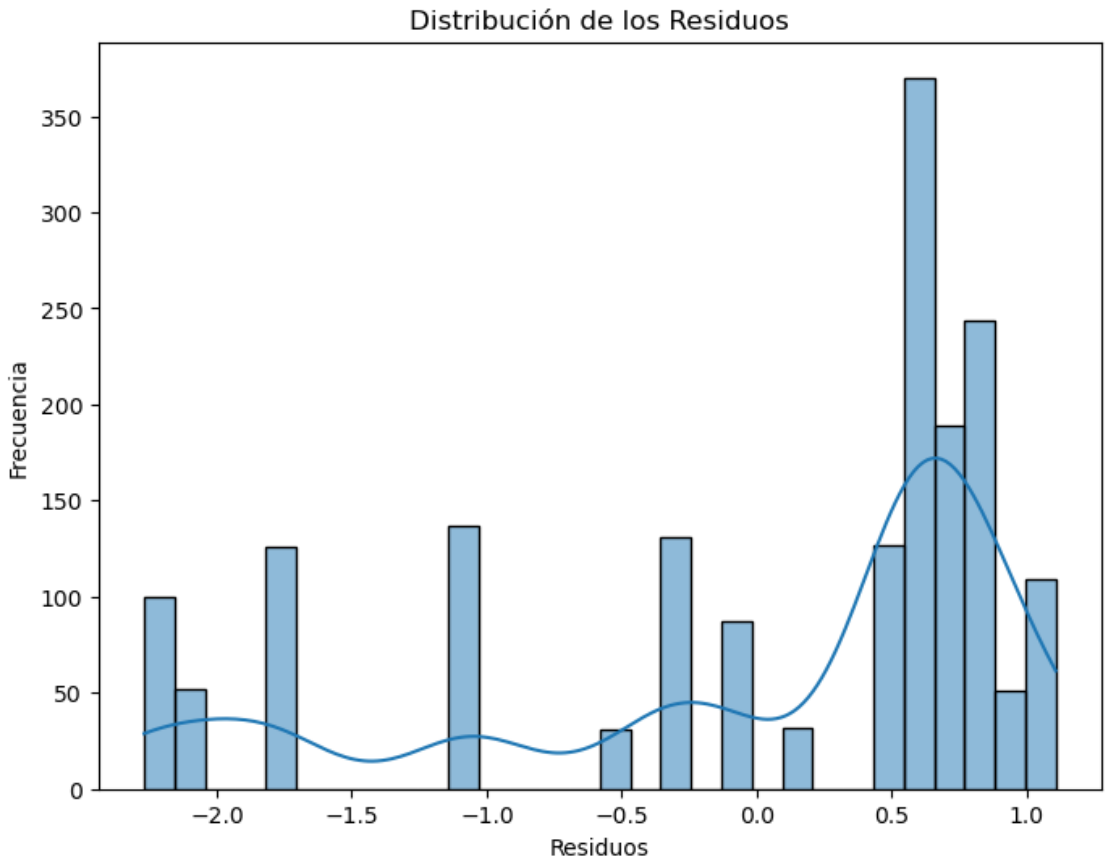
[51]:
```
# Calcular los residuos
residuos = y_test - y_pred

# Mostrar los primeros residuos
print(residuos.head())
```

```
4040   -1.055777
4248   -1.055777
1966    0.770830
4766    0.908241
1374    0.588686
Name: review_score, dtype: float64
```

[ ]:
```
#### Es la diferencia entre el valor real y el proporcionado por el modelo.
```

[52]:
```
# Histograma de los residuos
plt.figure(figsize=(8, 6))
sns.histplot(residuos, kde=True, bins=30)
plt.title('Distribución de los Residuos')
plt.xlabel('Residuos')
plt.ylabel('Frecuencia')
plt.show()
```

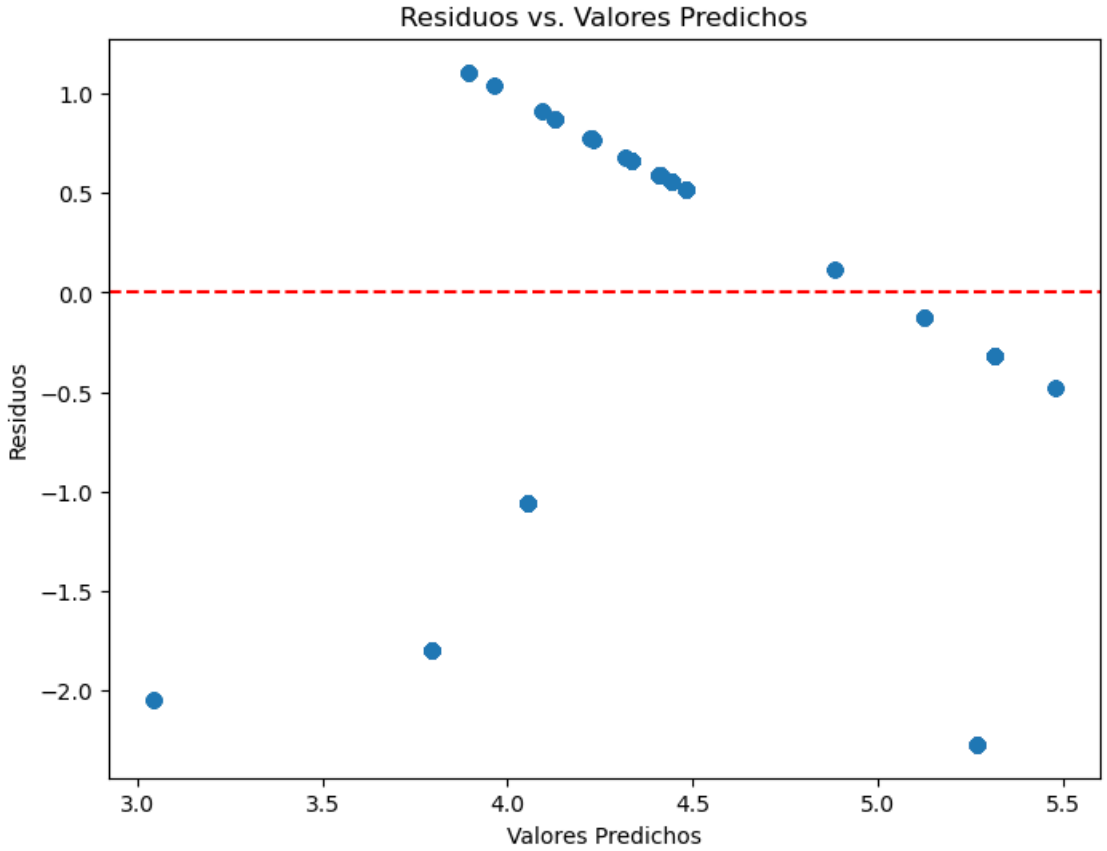
```
c:\Users\jmelo\AppData\Local\anaconda3\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[ ]:
```
'''
Los residuos no están simétricamente distribuidos alrededor de cero. En lugar␣
 ↪de una distribución normal, los residuos parecen estar concentrados
en el rango positivo, especialmente alrededor de 0.5, indicando una asimetría␣
 ↪hacia la derecha. Esto sugiere que el modelo tiende a subestimar
las review_scores más frecuentemente que a sobreestimarlas.

El histograma muestra múltiples picos (o modos), lo cual puede indicar la␣
 ↪presencia de varias subpoblaciones en los datos o que diferentes partes de␣
 ↪los
 datos están siendo modeladas de manera diferente por el modelo de regresión␣
 ↪lineal. Esto puede ser una señal de que un modelo lineal simple no está
 capturando adecuadamente la estructura de los datos.
'''
```

[53]:
```
## Visualizar la distribución de los residuos

# Residuos vs. valores predichos
plt.figure(figsize=(8, 6))
```

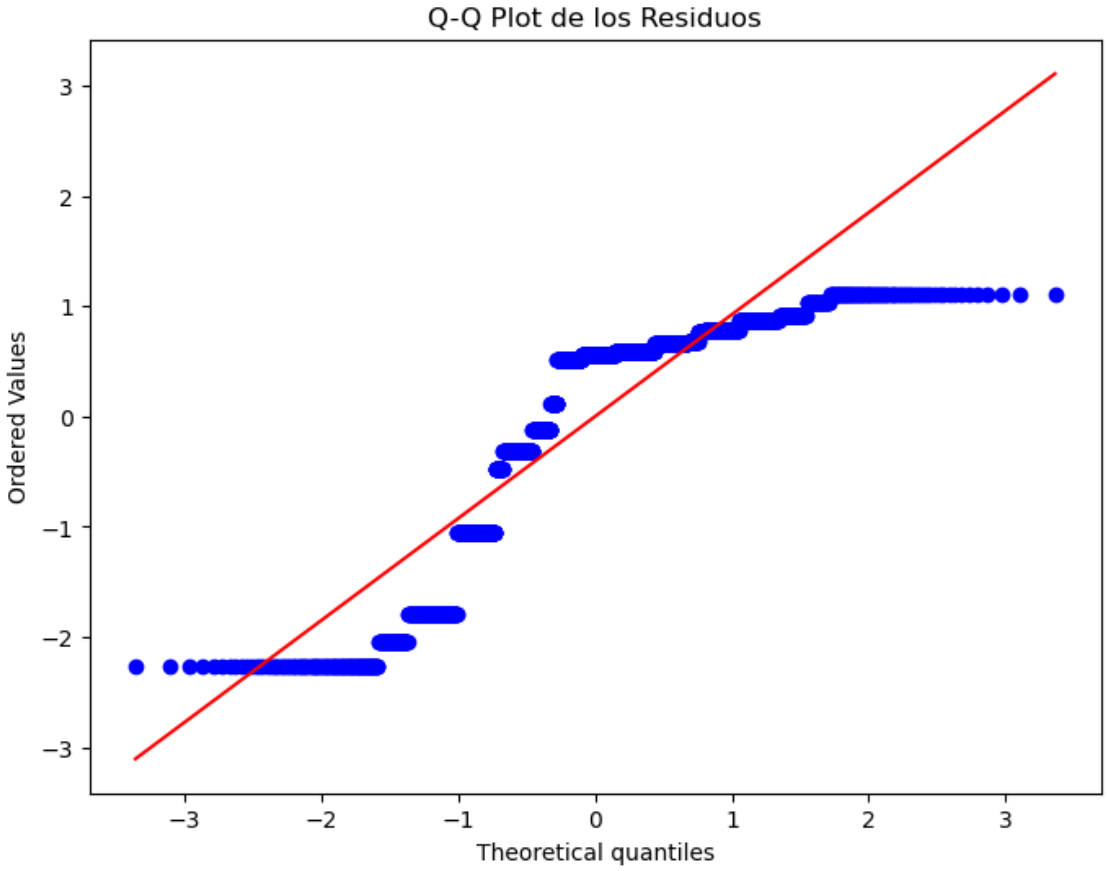
```
plt.scatter(y_pred, residuos)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuos vs. Valores Predichos')
plt.xlabel('Valores Predichos')
plt.ylabel('Residuos')
plt.show()
```



[63]:
```
'''Esto indica que el modelo de regresión lineal no está capturando␣
 ↪adecuadamente la relación entre las variables independientes y la variable␣
 ↪dependiente.
En este caso, parece que existe una relación no lineal entre los predictores y␣
 ↪la review_score.
'''
```

[63]:
```
'Esto indica que el modelo de regresión lineal no está capturando adecuadamente
la relación entre las variables independientes y la variable dependiente. \nEn
este caso, parece que existe una relación no lineal entre los predictores y la
review_score.\n'
```

[54]:
```
## Q-Q Plot (Gráfico Cuantil-Cuantil)

import scipy.stats as stats

# Q-Q plot para los residuos
plt.figure(figsize=(8, 6))
stats.probplot(residuos, dist="norm", plot=plt)
plt.title('Q-Q Plot de los Residuos')
plt.show()
```



[ ]:
```
'''
los residuos muestran desviaciones notables de la línea diagonal, especialmente␣
 ↪en los extremos y alrededor de la mediana.
La curva en el Q-Q plot confirma que los residuos no siguen una distribución␣
 ↪normal, lo que refuerza la idea de que el modelo de regresión lineal simple
puede no ser el más adecuado para este conjunto de datos.
'''
```

[64]:
```
### Paso 7. Modelo Random Forest
```

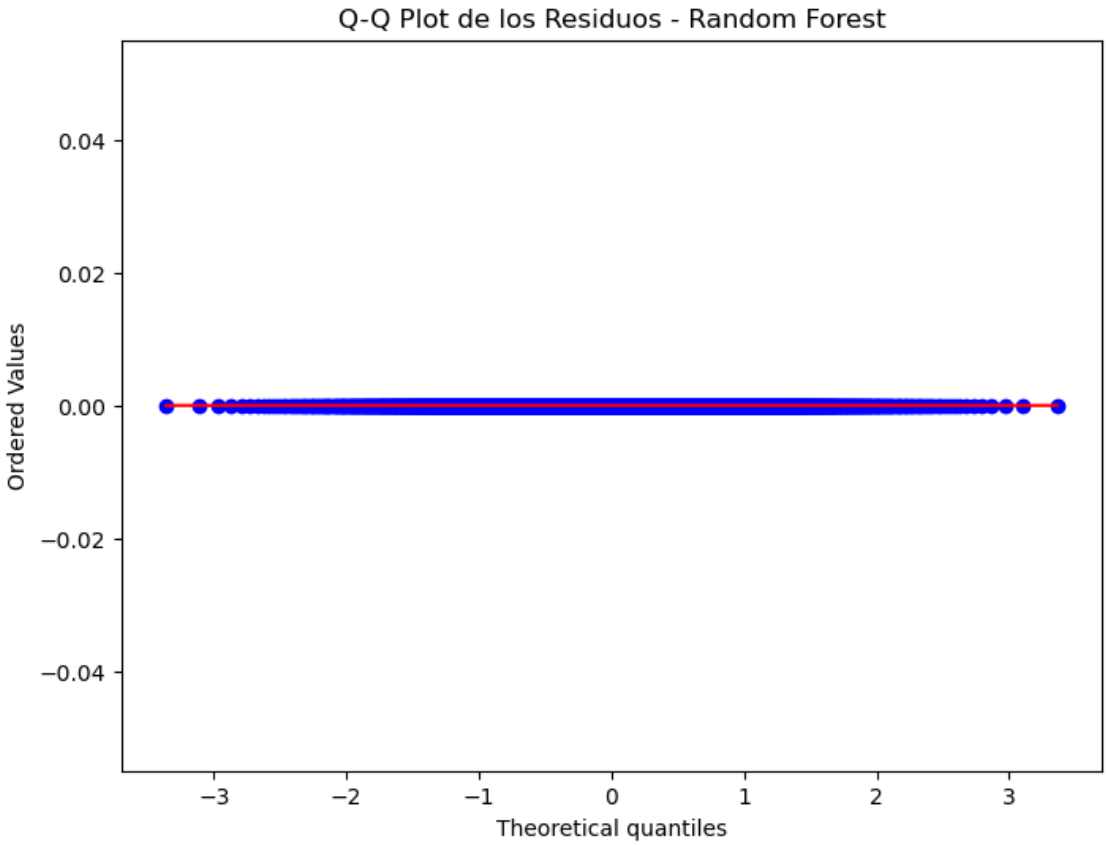
[65]:
```
from sklearn.ensemble import RandomForestRegressor

# Instanciar el modelo de Random Forest
rf_model = RandomForestRegressor(random_state=42, n_estimators=100)

# Ajustar el modelo a los datos de entrenamiento
rf_model.fit(X_train, y_train)

# Realizar predicciones sobre el conjunto de prueba
y_pred_rf = rf_model.predict(X_test)

# Calcular el MSE y R² para el modelo de Random Forest
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - MSE: {mse_rf}")
print(f"Random Forest - R²: {r2_rf}")
```

```
Random Forest - MSE: 0.0
Random Forest - R²: 1.0
```

[69]:
```
''' Un MSE de 0.0 indica que el modelo predice los valores de review_score de␣
 ↪manera perfecta, sin ningún error cuadrático medio.
Este resultado es extremadamente inusual y sugiere que el modelo se ha ajustado␣
 ↪de manera exacta a los datos.

Coeficiente de Determinación (R²): 1.0: El modelo explica el 100% de la␣
 ↪variabilidad en los datos de review_score, teniendo un ajuste perfecto,
donde las predicciones del modelo coinciden exactamente con los valores reales.

Los resultados indican un sobreajuste extremo. El modelo de Random Forest ha␣
 ↪aprendido los detalles específicos del conjunto de datos de entrenamiento
y prueba hasta el punto en que predice los valores exactamente.
'''
```

[69]:
```
' Un MSE de 0.0 indica que el modelo predice los valores de review_score de
manera perfecta, sin ningún error cuadrático medio. \nEste resultado es
extremadamente inusual y sugiere que el modelo se ha ajustado de manera exacta a
los datos.\n\nCoeficiente de Determinación (R²): 1.0: El modelo explica el 100%
de la variabilidad en los datos de review_score, teniendo un ajuste perfecto,
\ndonde las predicciones del modelo coinciden exactamente con los valores
reales.\n\nLos resultados indican un sobreajuste extremo. El modelo de Random
Forest ha aprendido los detalles específicos del conjunto de datos de
entrenamiento \ny prueba hasta el punto en que predice los valores
exactamente.\n'
```

[70]:
```
### Gráfico  Q-Q plot para los residuos

import matplotlib.pyplot as plt
import scipy.stats as stats

# Calcular los residuos
residuos_rf = y_test - y_pred_rf

# Q-Q plot para los residuos del modelo Random Forest
plt.figure(figsize=(8, 6))
stats.probplot(residuos_rf, dist="norm", plot=plt)
plt.title('Q-Q Plot de los Residuos - Random Forest')
plt.show()
```



[71]:
```
'''
El modelo de Random Forest ha sobreajustado los datos al punto de predecir los␣
↪valores de review_score sin ningún error.
Es un inconveneinte ya que muy seguramente no funcione bien con nuevos datos
'''
```

[71]:
```
' \nEl modelo de Random Forest ha sobreajustado los datos al punto de predecir los valores de review_score sin ningún error. \nEs un inconveneinte ya que muy seguramente no funcione bien con nuevos datos \n'
```

[ ]:
```
## Paso 8: Modelo de Regrsion Logistica
```

[ ]:
```
'''
Variable continua: distancia
Variable categórica/binaria: order_status
'''
```

[ ]:
```
### Preparación de datos
```

[82]:
```
import psycopg2
import pandas as pd

try:
    # Conectar a la base de datos PostgreSQL
    conn = psycopg2.connect(
        dbname="olist_ecommerce",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
        options="-c client_encoding=UTF8"
    )

    # Crear un cursor para ejecutar la consulta
    cur = conn.cursor()

    # Ejecutar la consulta SQL ajustada
    cur.execute("""
        SELECT
            o.order_id,
            o.order_status,
            o.order_purchase_timestamp,
            o.order_approved_at,
            o.order_delivered_carrier_date,
            o.order_delivered_customer_date,
            o.order_estimated_delivery_date,
            c.customer_id,
            c.customer_zip_code_prefix,
            s.seller_id,
            s.seller_zip_code_prefix,
            oi.product_id,
            oi.price,
            oi.freight_value,
```

```
            r.review_score,
            p.payment_type,
            p.payment_installments,
            p.payment_value,
            op.product_category_name,
            og.geolocation_lat,
            og.geolocation_lng
        FROM
            olist_orders o
        JOIN
            olist_order_customers c ON o.customer_id = c.customer_id
        JOIN
            olist_order_items oi ON o.order_id = oi.order_id
        JOIN
            olist_products op ON oi.product_id = op.product_id
        JOIN
            olist_sellers s ON oi.seller_id = s.seller_id
        JOIN
            olist_order_reviews r ON o.order_id = r.order_id
        JOIN
            olist_order_payments p ON o.order_id = p.order_id
        JOIN
            olist_geolocation og ON s.seller_zip_code_prefix = og.
 ↪zip_code_prefix
        WHERE
            c.customer_zip_code_prefix = og.zip_code_prefix
            AND o.order_delivered_customer_date IS NOT NULL;
    """)

    # Convertir los resultados a un DataFrame de pandas
    df = pd.DataFrame(cur.fetchall(), columns=[
        'order_id', 'order_status', 'order_purchase_timestamp',
        'order_approved_at', 'order_delivered_carrier_date',
        'order_delivered_customer_date', 'order_estimated_delivery_date',
        'customer_id', 'customer_zip_code_prefix', 'seller_id',
        'seller_zip_code_prefix', 'product_id', 'price',
        'freight_value', 'review_score', 'payment_type',
        'payment_installments', 'payment_value', 'product_category_name',
        'geolocation_lat', 'geolocation_lng'
    ])

    # Mostrar algunas filas para verificar los datos
    print(df.head())

except Exception as e:
    print(f"Error al conectar a la base de datos: {e}")
```

```python
finally:
    # Cerrar la conexión si fue abierta
    if conn:
        cur.close()
        conn.close()
        print("Conexión cerrada.")
```

```
                          order_id order_status order_purchase_timestamp  \
0  85b58bb30d219374cf9cfdebbc5fbb69    delivered      2017-02-10 09:37:46
1  85b58bb30d219374cf9cfdebbc5fbb69    delivered      2017-02-10 09:37:46
2  85b58bb30d219374cf9cfdebbc5fbb69    delivered      2017-02-10 09:37:46
3  85b58bb30d219374cf9cfdebbc5fbb69    delivered      2017-02-10 09:37:46
4  85b58bb30d219374cf9cfdebbc5fbb69    delivered      2017-02-10 09:37:46

    order_approved_at order_delivered_carrier_date  \
0 2017-02-11 06:50:17          2017-02-17 09:54:05
1 2017-02-11 06:50:17          2017-02-17 09:54:05
2 2017-02-11 06:50:17          2017-02-17 09:54:05
3 2017-02-11 06:50:17          2017-02-17 09:54:05
4 2017-02-11 06:50:17          2017-02-17 09:54:05

  order_delivered_customer_date order_estimated_delivery_date  \
0           2017-02-17 17:53:43                    2017-03-17
1           2017-02-17 17:53:43                    2017-03-17
2           2017-02-17 17:53:43                    2017-03-17
3           2017-02-17 17:53:43                    2017-03-17
4           2017-02-17 17:53:43                    2017-03-17

                       customer_id  customer_zip_code_prefix  \
0  c498555685a103ca04bcb5f81cd974ab                     14940
1  c498555685a103ca04bcb5f81cd974ab                     14940
2  c498555685a103ca04bcb5f81cd974ab                     14940
3  c498555685a103ca04bcb5f81cd974ab                     14940
4  c498555685a103ca04bcb5f81cd974ab                     14940

                         seller_id  …                       product_id  \
0  cca3071e3e9bb7d12640c9fbe2301306  …  e336c656869480e20d04ca9389b12167
1  cca3071e3e9bb7d12640c9fbe2301306  …  e336c656869480e20d04ca9389b12167
2  cca3071e3e9bb7d12640c9fbe2301306  …  e336c656869480e20d04ca9389b12167
3  cca3071e3e9bb7d12640c9fbe2301306  …  e336c656869480e20d04ca9389b12167
4  cca3071e3e9bb7d12640c9fbe2301306  …  e336c656869480e20d04ca9389b12167

  price  freight_value  review_score  payment_type payment_installments  \
0  59.9          11.81             3        boleto                    1
1  59.9          11.81             3        boleto                    1
2  59.9          11.81             3        boleto                    1
3  59.9          11.81             3        boleto                    1
4  59.9          11.81             3        boleto                    1
```

```
   payment_value    product_category_name geolocation_lat  geolocation_lng
0          71.71  fashion_roupa_masculina      -21.749938       -48.824524
1          71.71  fashion_roupa_masculina      -21.766477       -48.831547
2          71.71  fashion_roupa_masculina      -21.750621       -48.831497
3          71.71  fashion_roupa_masculina      -21.748903       -48.814598
4          71.71  fashion_roupa_masculina      -21.752014       -48.839119

[5 rows x 21 columns]
Conexión cerrada.
```

[89]:
```
# Calcular la diferencia entre las fechas de entrega real y estimada
df['delivery_delay'] = df['order_delivered_customer_date'] -␣
 ↪df['order_estimated_delivery_date']

# Convertir la diferencia en días
df['delivery_delay'] = df['delivery_delay'].dt.days

# Crear la variable binaria 'is_late': 1 si hubo retraso, 0 si no lo hubo
df['is_late'] = (df['delivery_delay'] > 0).astype(int)

# Seleccionar las características (variables predictoras) y la variable objetivo
X = df[['review_score', 'price']]
y = df['is_late']

# Calcular 'delivery_delay'
df['delivery_delay'] = df['order_delivered_customer_date'] -␣
 ↪df['order_estimated_delivery_date']
df['delivery_delay'] = df['delivery_delay'].dt.days

# Crear la variable binaria 'is_late'
df['is_late'] = (df['delivery_delay'] > 0).astype(int)

# Seleccionar las características y la variable objetivo
X = df[['review_score', 'price']]
y = df['is_late']
```

[90]:
```
### División del Conjunto de Datos

from sklearn.model_selection import train_test_split

# Dividir el conjunto de datos en entrenamiento (70%) y prueba (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[91]:
```
### Entrenamiento del Modelo de Regresión Logística
```

```
from sklearn.linear_model import LogisticRegression

# Crear el modelo de regresión logística
model = LogisticRegression()

# Entrenar el modelo con los datos de entrenamiento
model.fit(X_train, y_train)
```

[91]:
```
LogisticRegression()
```

[95]:
```
#### Entrenamiento del modelo con exito
```

[92]:
```
### Evaluación del Modelo
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo: {accuracy:.2f}')

# Generar la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
print('Matriz de confusión:')
print(conf_matrix)

# Generar un reporte de clasificación
class_report = classification_report(y_test, y_pred)
print('Reporte de clasificación:')
print(class_report)
```

```
Precisión del modelo: 1.00
Matriz de confusión:
[[1743    0]
 [   0   43]]
Reporte de clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1743
           1       1.00      1.00      1.00        43

    accuracy                           1.00      1786
   macro avg       1.00      1.00      1.00      1786
weighted avg       1.00      1.00      1.00      1786
```

[ ]:
```
'''
Resultado:
El modelo predice correctamente el estado de retraso o no, en un 100% de los␣
 ↪casos del conjunto de prueba.
-- 1743: Pedidos que no tuvieron retraso y fueron correctamente clasificados.
-- 43: Pedidos que tuvieron retraso y fueron correctamente clasificados.
-- 0 en las posiciones de errores: No hubo falsos positivos ni falsos negativos.

Los resultados indican que el modelo ha clasificado perfectamente todos los␣
 ↪ejemplos en el conjunto de prueba, sin ningún error.
Esto indica que hay sobreajuste.
'''
```

[93]:
```
# Coeficientes
coef = model.coef_[0]
intercept = model.intercept_[0]

print(f'Coeficientes: {coef}')
print(f'Intercepto: {intercept}')
```

```
Coeficientes: [-6.17720809  0.03626434]
Intercepto: 4.614624755070565
```

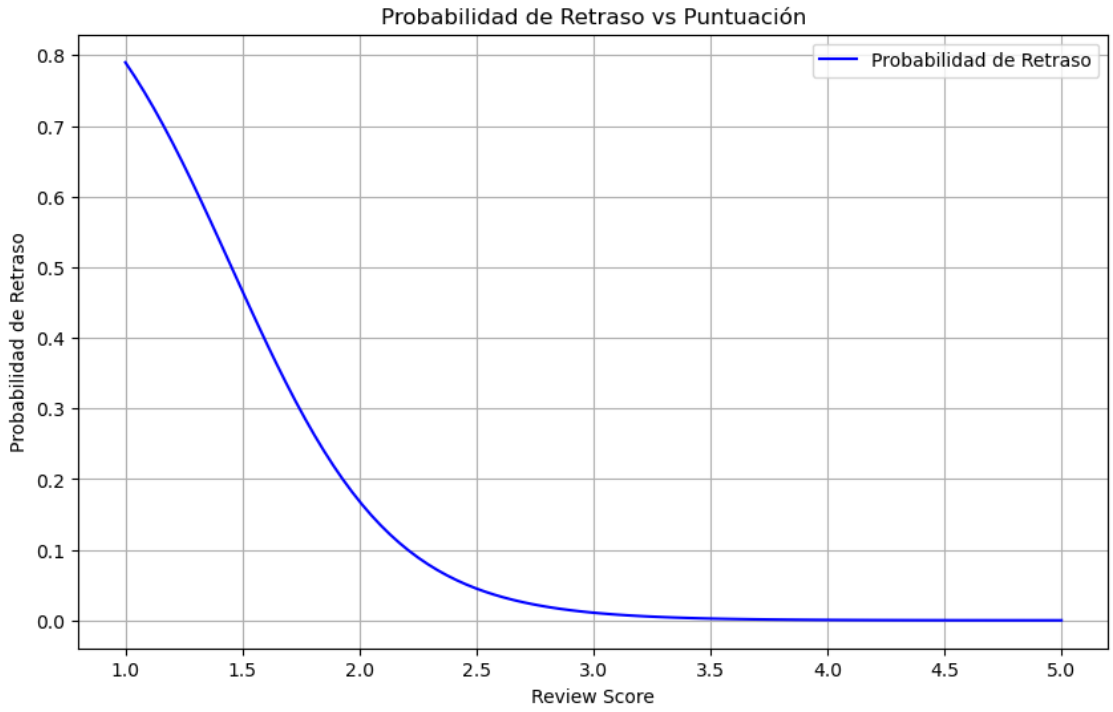
[94]:
```
# Modelo con un valor de regularización diferente
model = LogisticRegression(C=0.1, solver='liblinear')
model.fit(X_train, y_train)
```

[94]:
```
LogisticRegression(C=0.1, solver='liblinear')
```

[96]:
```
#### la probabilidad de que un pedido tenga un retraso disminuye, ya que una␣
 ↪puntuación alta esta asociada con mejores servicios o productos.
```

[98]:
```
### Gráfico de Probabilidades Predichas vs Puntación


import numpy as np
import matplotlib.pyplot as plt

# Crear una serie de valores de review_score
review_scores = np.linspace(X['review_score'].min(), X['review_score'].max(),␣
 ↪100)

# Usar el precio medio para el gráfico
mean_price = X['price'].mean()

# Crear un DataFrame con los valores de review_score y el precio fijo
X_plot = pd.DataFrame({'review_score': review_scores, 'price': mean_price})
```

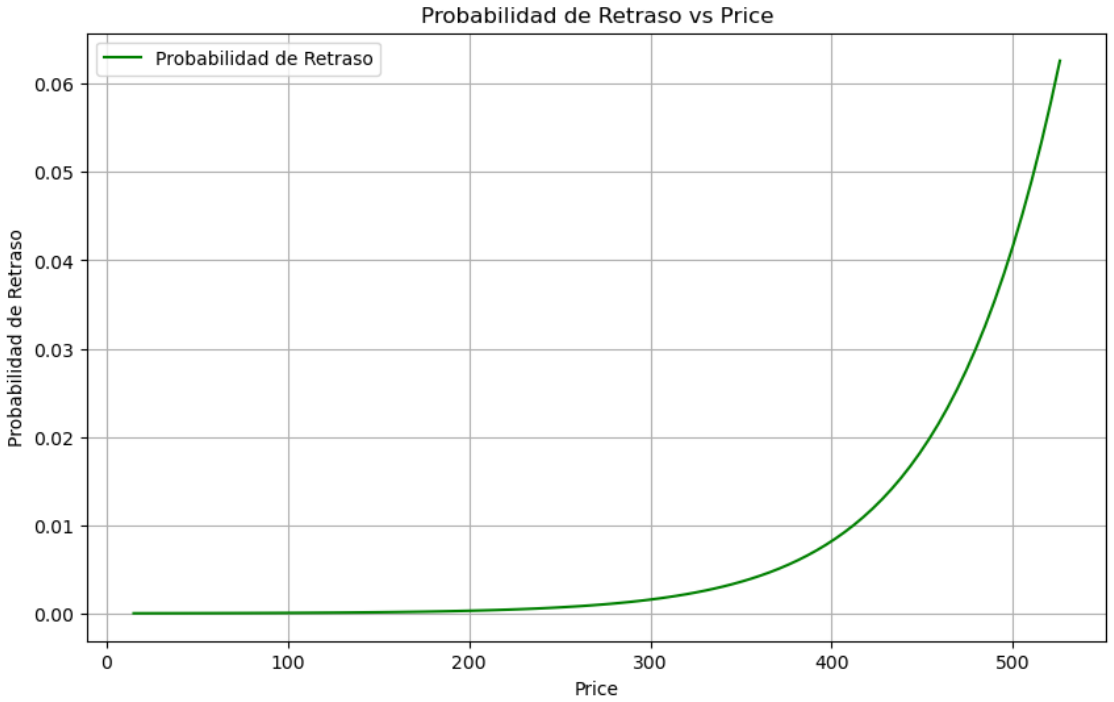
```
# Calcular las probabilidades predichas por el modelo
predicted_probs = model.predict_proba(X_plot)[:, 1]  # Probabilidad de que␣
 ↪is_late sea 1

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(review_scores, predicted_probs, color='blue', label='Probabilidad de␣
 ↪Retraso')
plt.xlabel('Review Score')
plt.ylabel('Probabilidad de Retraso')
plt.title('Probabilidad de Retraso vs Puntuación')
plt.legend()
plt.grid(True)
plt.show()
```



[101]:
```
'''
Presenta una tendencia inversamente proporcional: A medida que la puntuación de␣
 ↪la reseña aumenta, la probabilidad de retraso disminuye significativamente.
'''
```

[101]:
```
'\nPresenta una tendencia inversamente proporcional: A medida que la puntuación
de la reseña aumenta, la probabilidad de retraso disminuye
significativamente.\n'
```

[99]:
```
## Gráfico de Probabilidades Retraso vs Precio
# Crear una serie de valores de price
prices = np.linspace(X['price'].min(), X['price'].max(), 100)

# Usar el review_score medio para el gráfico
mean_review_score = X['review_score'].mean()

# Crear un DataFrame con los valores de price y el review_score fijo
X_plot_price = pd.DataFrame({'review_score': mean_review_score, 'price':␣
 ↪prices})

# Calcular las probabilidades predichas por el modelo
predicted_probs_price = model.predict_proba(X_plot_price)[:, 1]  # Probabilidad␣
 ↪de que is_late sea 1

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(prices, predicted_probs_price, color='green', label='Probabilidad de␣
 ↪Retraso')
plt.xlabel('Price')
plt.ylabel('Probabilidad de Retraso')
plt.title('Probabilidad de Retraso vs Price')
plt.legend()
plt.grid(True)
plt.show()
```



[100]:
```
'''
La curva es una función creciente, lo que significa que a medida que el precio␣
 ↪del producto aumenta, la probabilidad de que el pedido se retrase
también aumenta, aunque de manera no lineal.
'''
```

[100]:
```
'\nLa curva es una función creciente, lo que significa que a medida que el
precio del producto aumenta, la probabilidad de que el pedido se retrase
\ntambién aumenta, aunque de manera no lineal.\n'
```

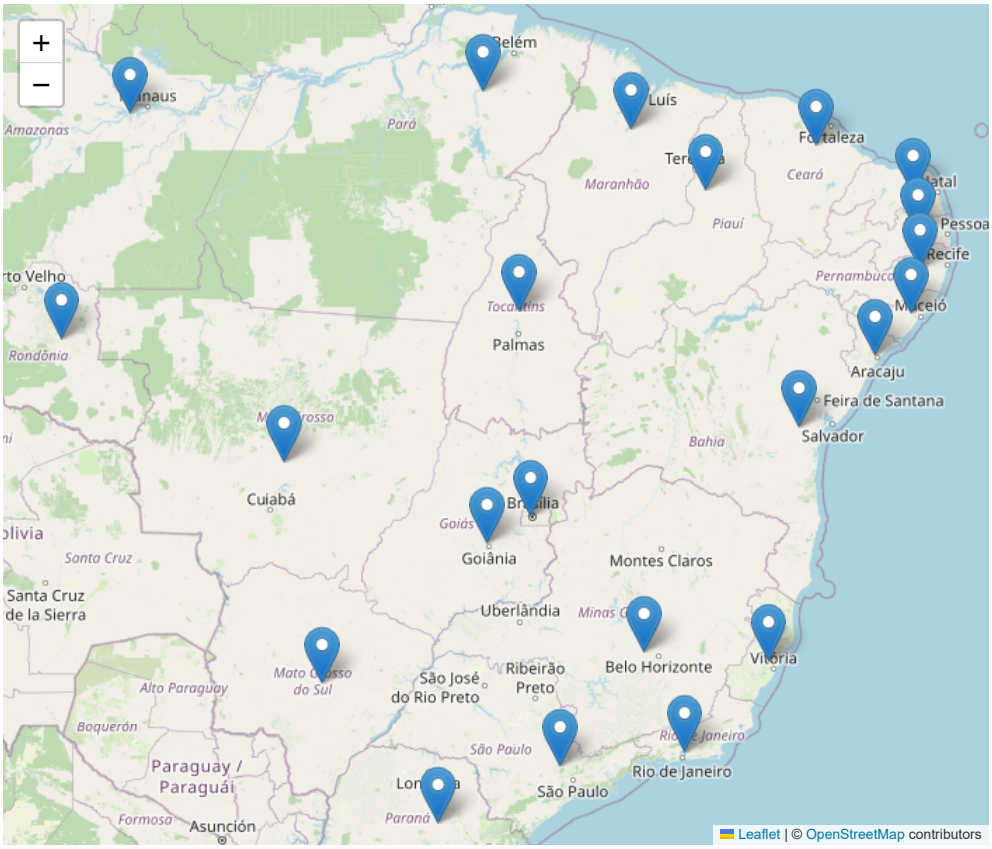
Resultados Analisis de Datos BD Olist E-commerce

Conjunto de datos públicos de comercio electrónico brasileño de pedidos realizados en Olist Store. El conjunto de datos tiene información de 100 000 pedidos de 2016 a 2018 realizados en varios mercados de Brasil. Sus características permiten ver un pedido desde múltiples dimensiones: desde el estado del pedido, el precio, el pago y el rendimiento del flete hasta la ubicación del cliente, los atributos del producto y, finalmente, las reseñas escritas por los clientes. También publicamos un conjunto de datos de geolocalización que relaciona los códigos postales brasileños con las coordenadas de latitud y longitud.

Realizare la identificación de las áreas geográficas que presentan mayores demoras en las entregas de pedidos, a través del análisis espacial de los datos de geolocalización de clientes y vendedores, junto con la información sobre tiempos de entrega y costos de envío

Estados Ubicación Clientes

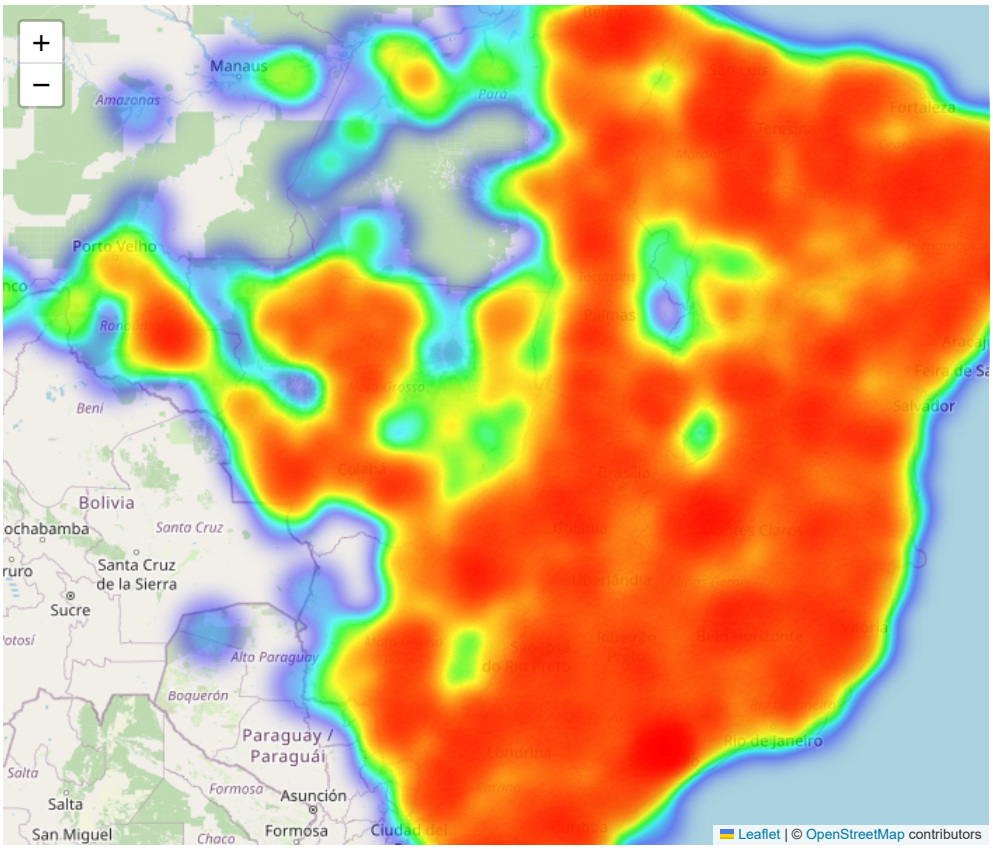
Visualización de estados donde existen clientes.



El mapa se contruye con una imagen del OpenStreetMap, representa la ubicación geográfica de los estados en Brasil donde existen clientes de la plataforma e-commerce. Los marcadores indican la posición aproximada de cada estado. La representación refleja mayor concentración de clientes en las regiones sudeste y noreste del país, con estados como São Paulo, Rio de Janeiro, y Minas Gerais, entre otros, destacándose como los principales mercados.

Mapa de calor - Ubicación Clientes

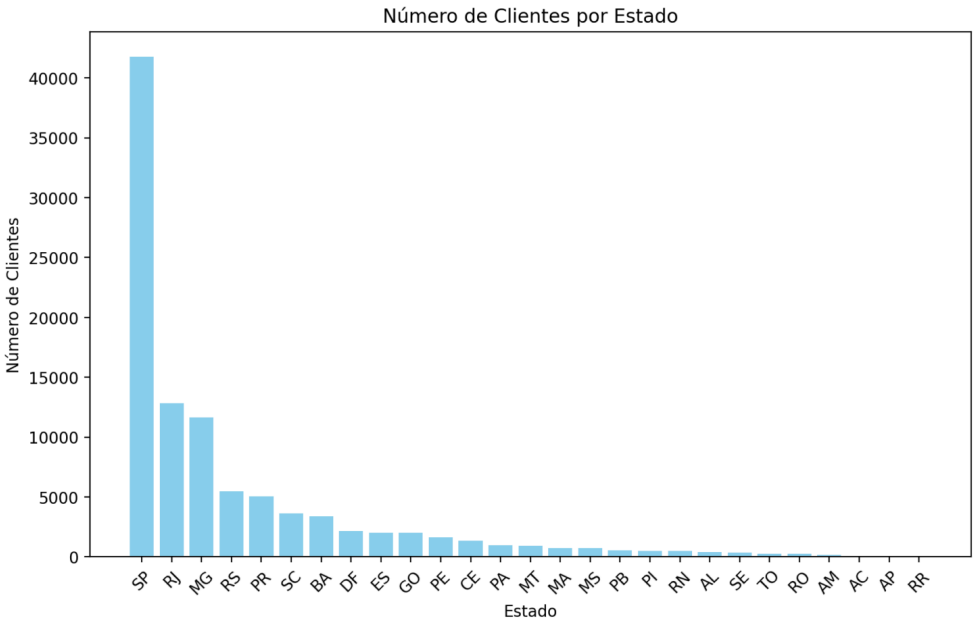
Ubicación de clientes en Brasil. Los colores varían desde tonos de azul, verde y amarillo, indicando las áreas con diferentes concentraciones de clientes.



El mapa se contruye con una imagen del OpenStreetMap, la cual incluyendo detalles geográficos como ciudades, carreteras y áreas protegidas. El color Verdes/Amarillo: Indican una alta concentración de clientes en esa ubicación específica. Estos son los puntos de mayor densidad, donde se observa la mayor actividad de clientes. El color Azul: Indican una menor concentración de clientes, pero aún así significativa, estas zonas tienen menos actividad en comparación con las áreas más cálidas.

Número de Clientes por Estado

Distribuyen los clientes de la plataforma e-commerce por estado, Sao Paulo como el principal mercado.



El gráfico representa la distribución del número de clientes por estado en la plataforma de e-commerce. El estado de Sao Paulo (SP) destaca como el principal mercado, con la mayor cantidad de clientes, seguido por Río de Janeiro (RJ) y Minas Gerais (MG). Los estados con menor población de clientes incluyen

Roraima (RR) y Amapá (AP), lo que sugiere una concentración de clientes en las áreas más pobladas y urbanizadas de Brasil.

Ciudades con Más de 500 Pedidos

Esta sección muestra las ciudades con más de 500 pedidos realizados en la plataforma de e-commerce.

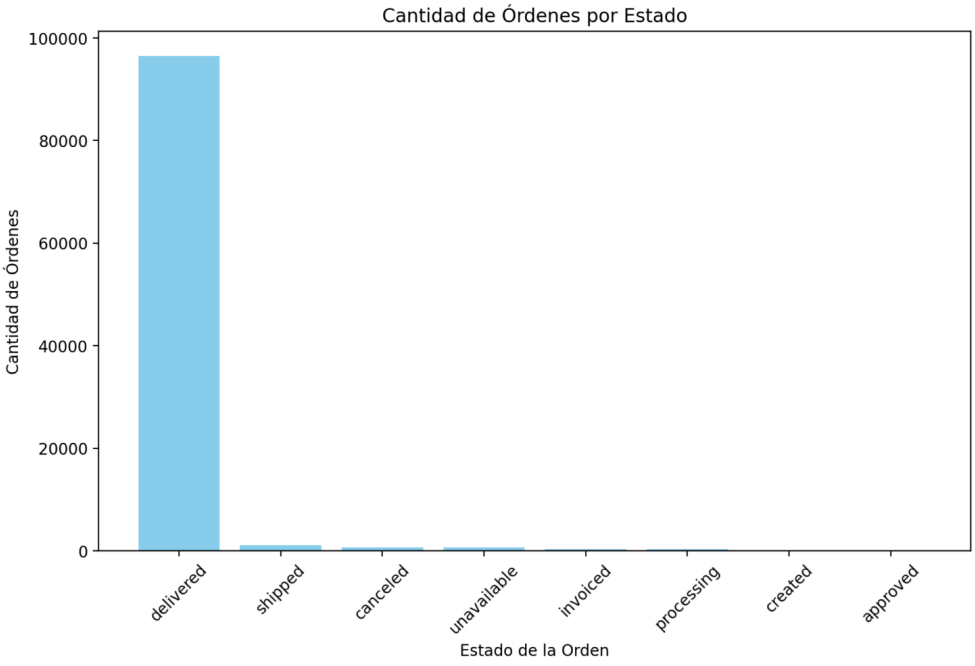
Top 10 Ciudades con Más de 500 Pedidos

A continuación se presenta una tabla que destaca las ciudades con más de 500 pedidos en la plataforma. Estas ciudades son los principales mercados que concentran un gran volumen de transacciones.

	ciudad	cantidadpedidos
0	sao paulo	15,540
1	rio de janeiro	6,882
2	belo horizonte	2,773
3	brasilia	2,131
4	curitiba	1,521
5	campinas	1,444
6	porto alegre	1,379
7	salvador	1,245
8	guarulhos	1,189
9	sao bernardo do campo	938

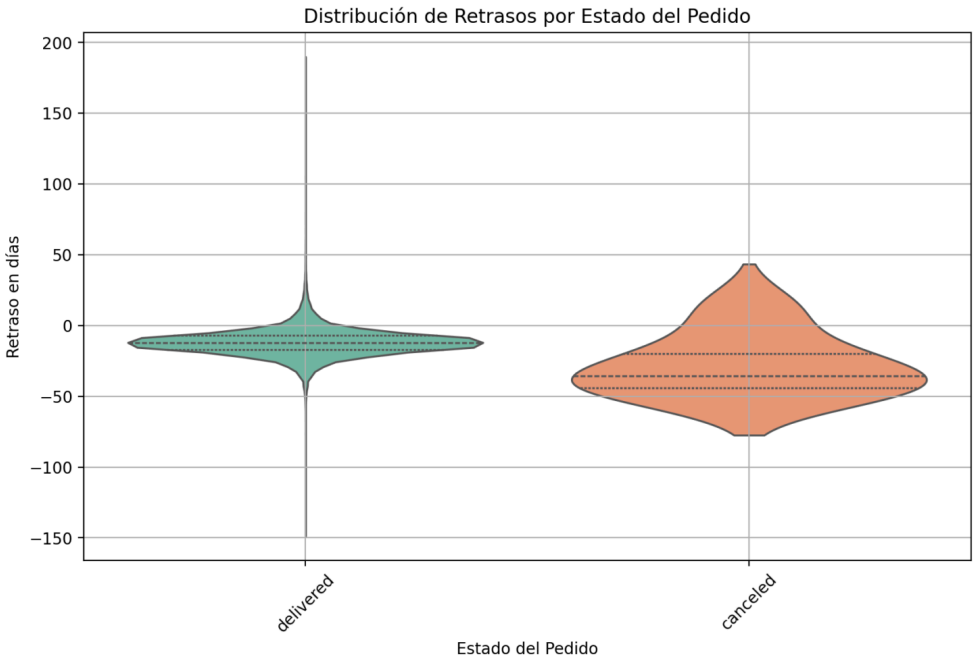
La tabla muestra las 10 principales ciudades que han generado más de 500 pedidos en la plataforma de e-commerce. Estas ciudades son puntos clave en la operación del comercio electrónico, concentrando la mayor actividad de compras. São Paulo es típicamente la ciudad con más actividad, seguida de otras grandes urbes como Rio de Janeiro y Belo Horizonte.

Estado de las ordenes y retrasos en la entrega



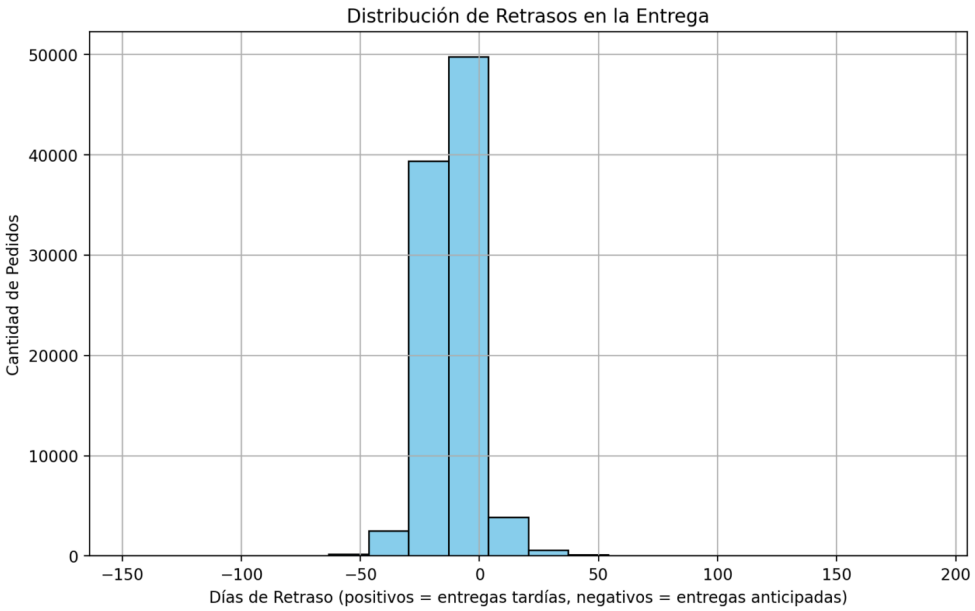
El gráfico muestra la distribución de órdenes según su estado en la plataforma de e-commerce. Se observa que la gran mayoría de las órdenes (alrededor de 100,000) han sido entregadas, representando el estado predominante. Los demás estados, como "shipped", "canceled", y otros, presentan una cantidad de órdenes significativamente menor, lo que indica que la mayoría de las transacciones alcanzan su finalización exitosa con la entrega del pedido..

Estado del Pedido y Retraso en la Entrega



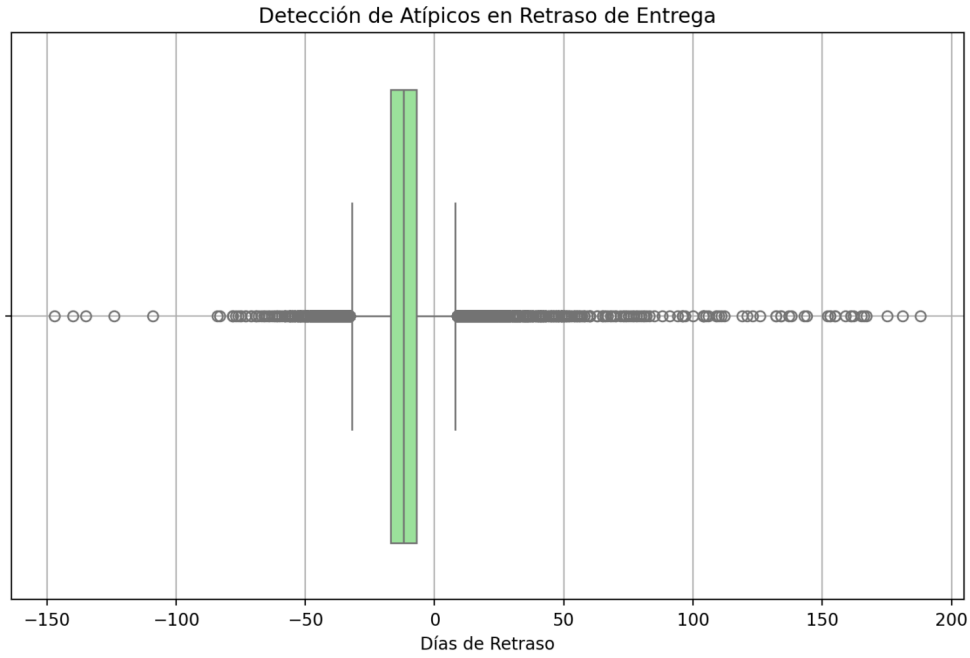
La gráfica muestra la distribución de los retrasos en la entrega según el estado del pedido. Se observa que, en general, los pedidos entregados tienen una distribución tanto de entregas anticipadas (valores negativos de retraso) como de pequeños retrasos (valores positivos de retraso). Por otro lado, los pedidos cancelados tienden a ser cancelados antes de la fecha estimada de entrega.

Distribución de Retrasos en la Entrega



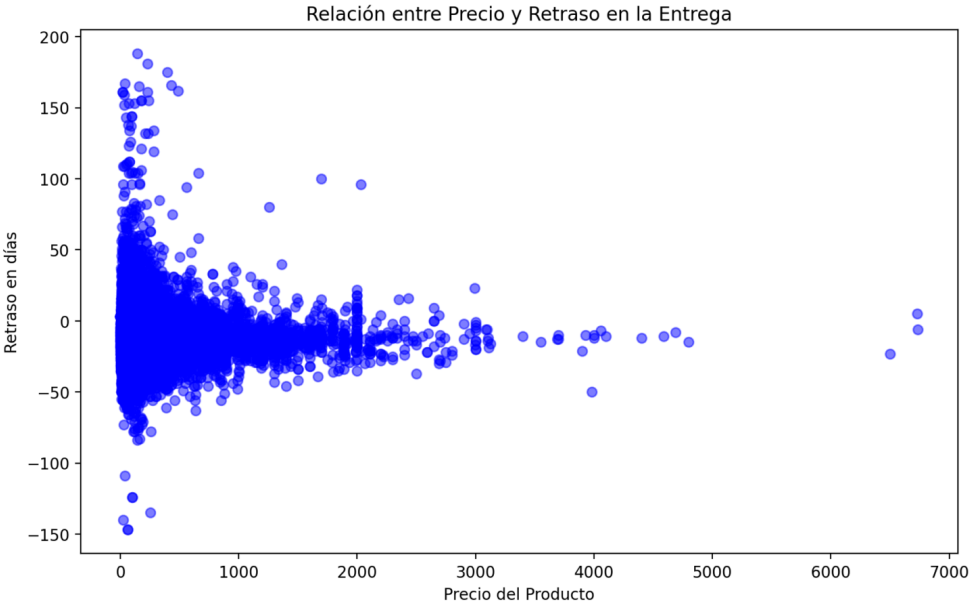
La mayoría de las entregas se realizan antes de la fecha estimada, con un pico alrededor de los 10 a 30 días antes de la entrega esperada..

Detección de Atípicos en Retraso de Entrega



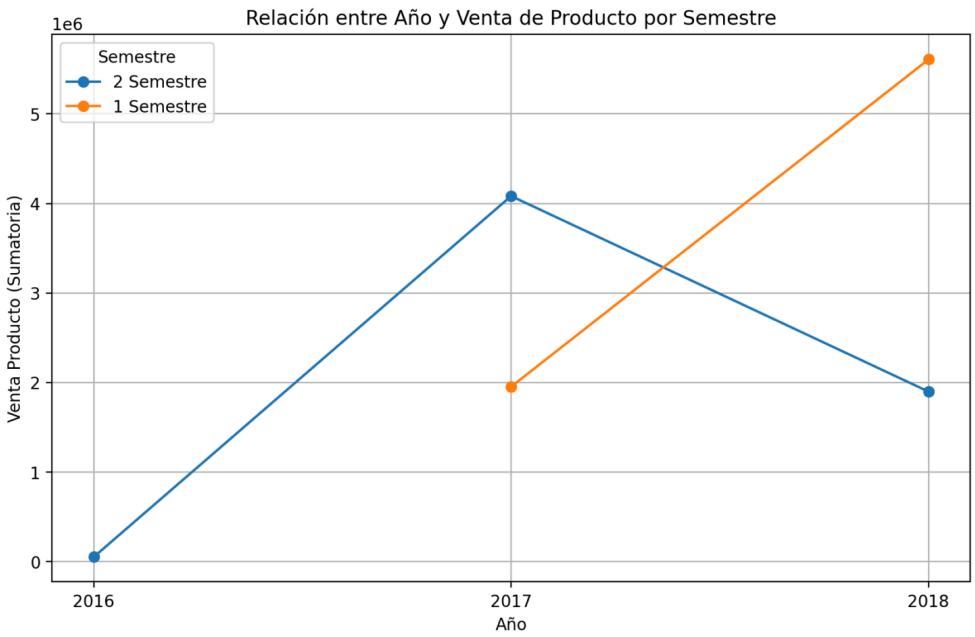
La gráfica evidencia que la mayoría de las entregas se realizan en torno a la fecha estimada, pero existen casos notables de entregas con retrasos o anticipaciones considerables, identificados como outliers en la distribución.

Relación entre Precio y Retraso en la Entrega



Este gráfico de dispersión muestra la relación entre el precio del producto y el retraso en la entrega. Se observa que la mayoría de los puntos se concentran en productos de bajo precio, generalmente por debajo de 1000 unidades monetarias. Para estos productos, los retrasos en la entrega varían considerablemente, abarcando desde entregas anticipadas hasta entregas tardías. A medida que el precio del producto aumenta, parece que la variabilidad en los retrasos disminuye, sugiriendo que los productos más caros tienden a cumplir mejor con las fechas de entrega estimadas.

Relación entre Año y Venta de Producto por Semestre

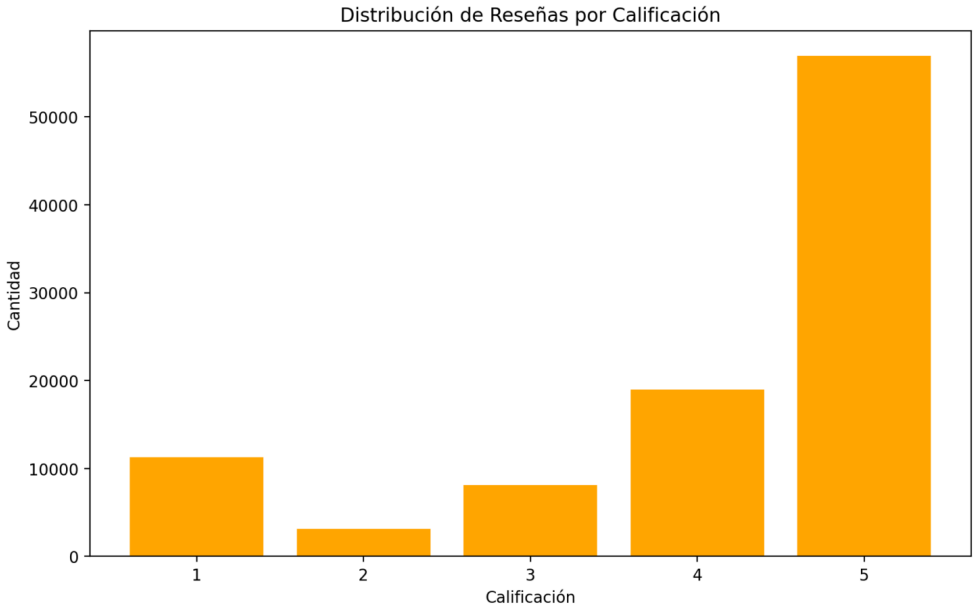


a) Año 2016: Durante este año, se observa un crecimiento inicial, pero limitado, en las ventas del segundo semestre, mientras que no se registran ventas significativas en el primer semestre.

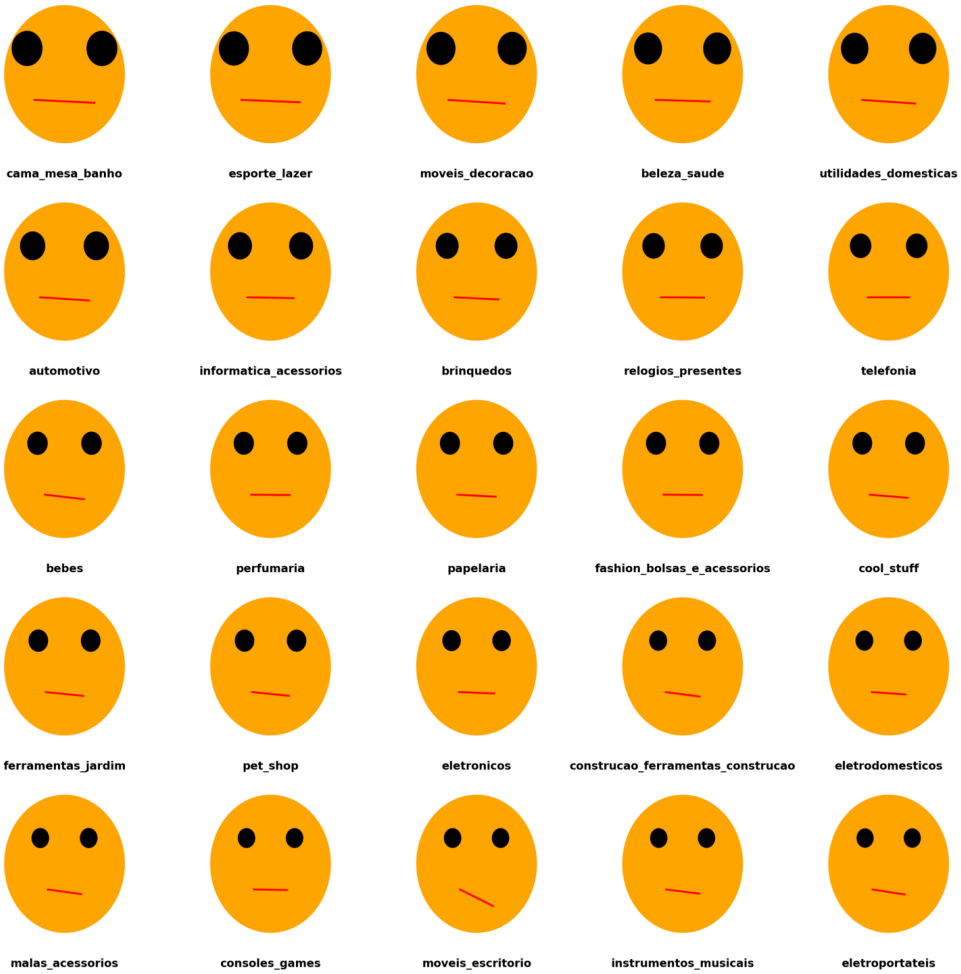
b) Año 2017: Este año destaca por un fuerte incremento en las ventas durante el segundo semestre, alcanzando el pico más alto de todo el período analizado. En contraste, el primer semestre de 2017 muestra ventas significativamente más bajas.

c) Año 2018: En 2018, se aprecia una inversión en el comportamiento observado en 2017. El primer semestre muestra un crecimiento notable, superando las ventas del segundo semestre del año anterior, mientras que el segundo semestre de 2018 muestra una caída considerable, llegando a niveles de ventas por debajo de los registrados en 2016.

Análisis de Reseñas y Puntuaciones



Visualización de Información de Productos con Caras Personalizadas



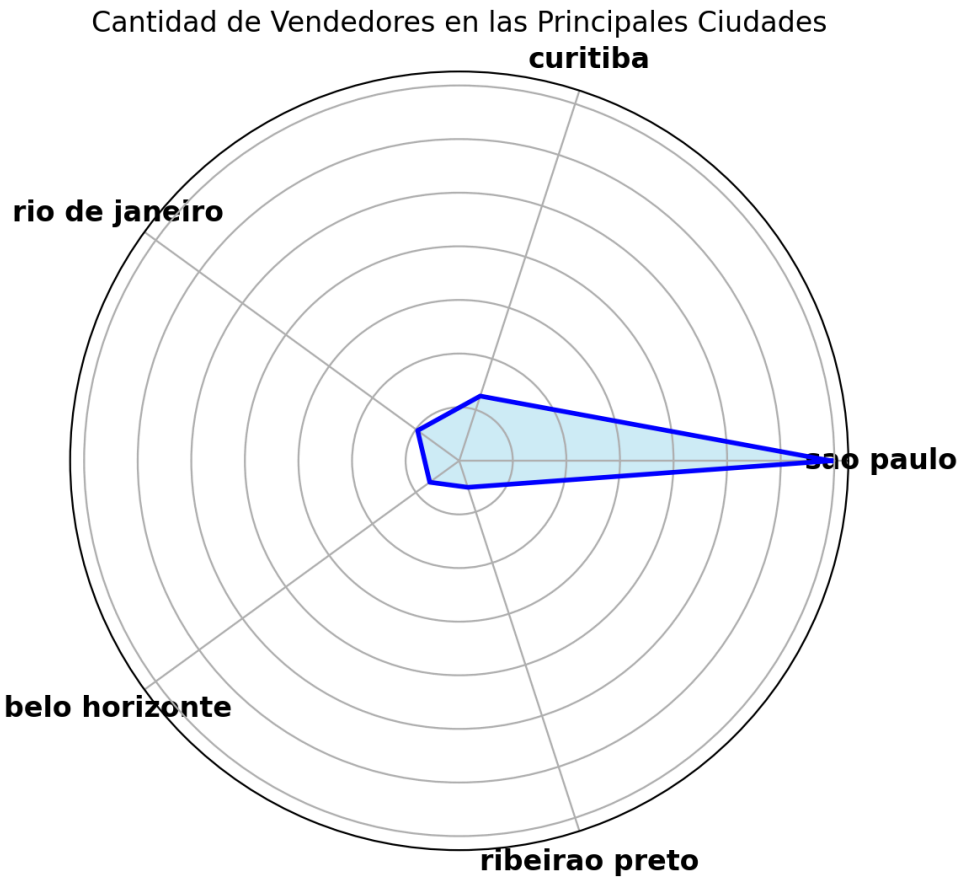
Esta visualización utiliza caras personalizadas para representar las características de los productos en diferentes categorías. El tamaño de los ojos representa el número de productos en esa categoría, y la forma de la boca refleja el peso promedio de los productos.

Información sobre los Vendedores

Distribución de vendedores en las principales ciudades donde opera la plataforma de e-commerce.

Columnas cargadas:

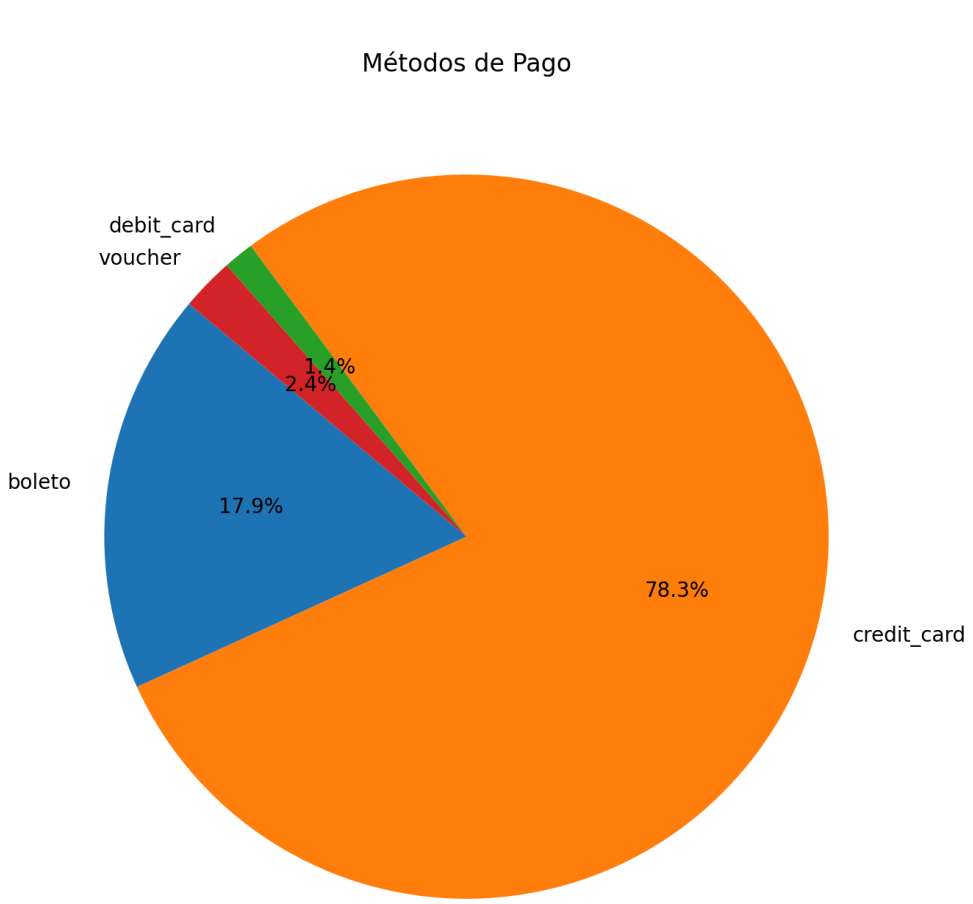

```
[
  0 : "ciudad"
  1 : "cnt_vendedores"
]
```



a) Sao Paulo: Representada en la esquina derecha del gráfico, es la ciudad con la mayor cantidad de vendedores, lo que se refleja en la longitud del eje correspondiente. b) Rio de Janeiro: Ubicada en la parte superior izquierda del gráfico, es la segunda ciudad con más vendedores. c) Belo Horizonte: También a la izquierda, muestra un número menor de vendedores en comparación con Sao Paulo y Rio de Janeiro. d) Curitiba: En la parte superior, tiene una cantidad de vendedores similar a la de Belo Horizonte. e) Ribeirao Preto: En la parte inferior del gráfico, muestra la menor cantidad de vendedores entre las cinco ciudades..

Métodos de Pago

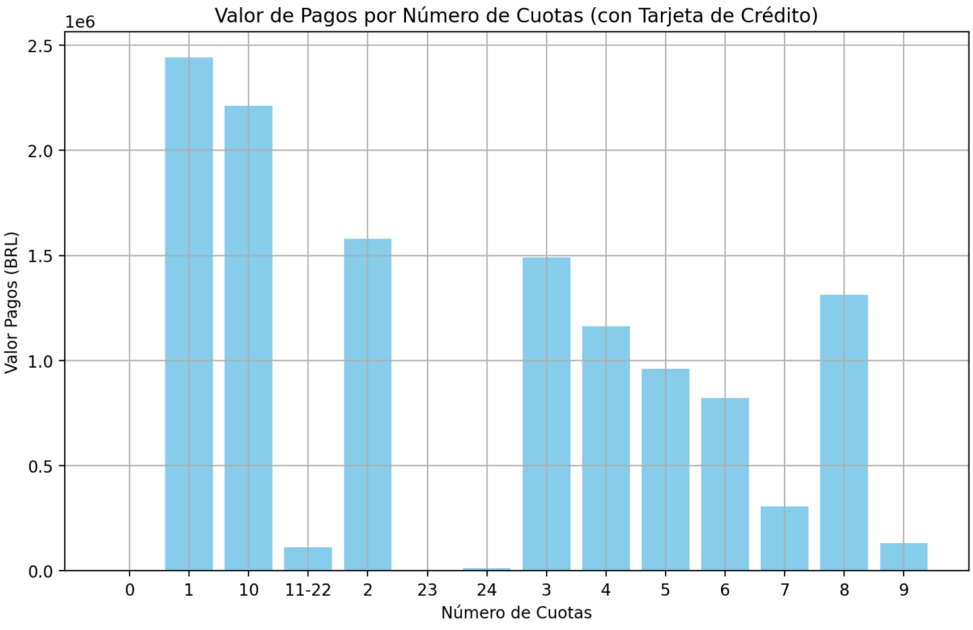
El gráfico muestra la distribución de los métodos de pago utilizados por los clientes



La tarjeta de credito representa la mayor parte de los pagos, con un 78.3% del total. Boleto bancario: Es el segundo método más común, con un 17.9% de los pagos. Tarjeta de débito: Representa el 2.4% de los pagos, siendo un método menos frecuente. Voucher: Con 1.4% del total, este es el método de pago menos utilizado entre los presentados.

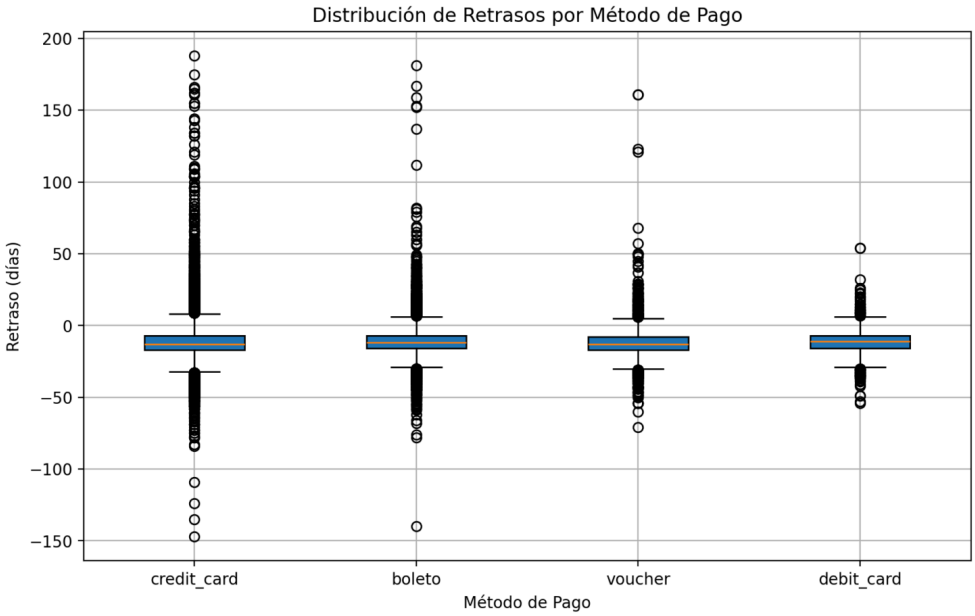
Pagos con Tarjeta de Crédito por Número de Cuotas

Gráfico de barras para visualizar el valor total de pagos en función del número de cuotas



El gráfico muestra la distribución del valor total de pagos realizados con tarjeta de crédito en función del número de cuotas. Observamos que los pagos a 1 y 2 cuotas son los más comunes, representando la mayor parte del valor total de pagos.

Métodos de Pago y Retrasos en la Entrega



Em el gráfico de caja y bigotes se observa la distribución de los retrasos en la entrega según el método de pago, en promedio todos los métodos tienen entregas anticipadas, no se observan diferencias significativas entre los métodos de pago. Existen valores atípicos notables, con algunas entregas extremadamente anticipadas o muy tardías, lo que sugiere variabilidad en los tiempos de entrega independientemente del método de pago.

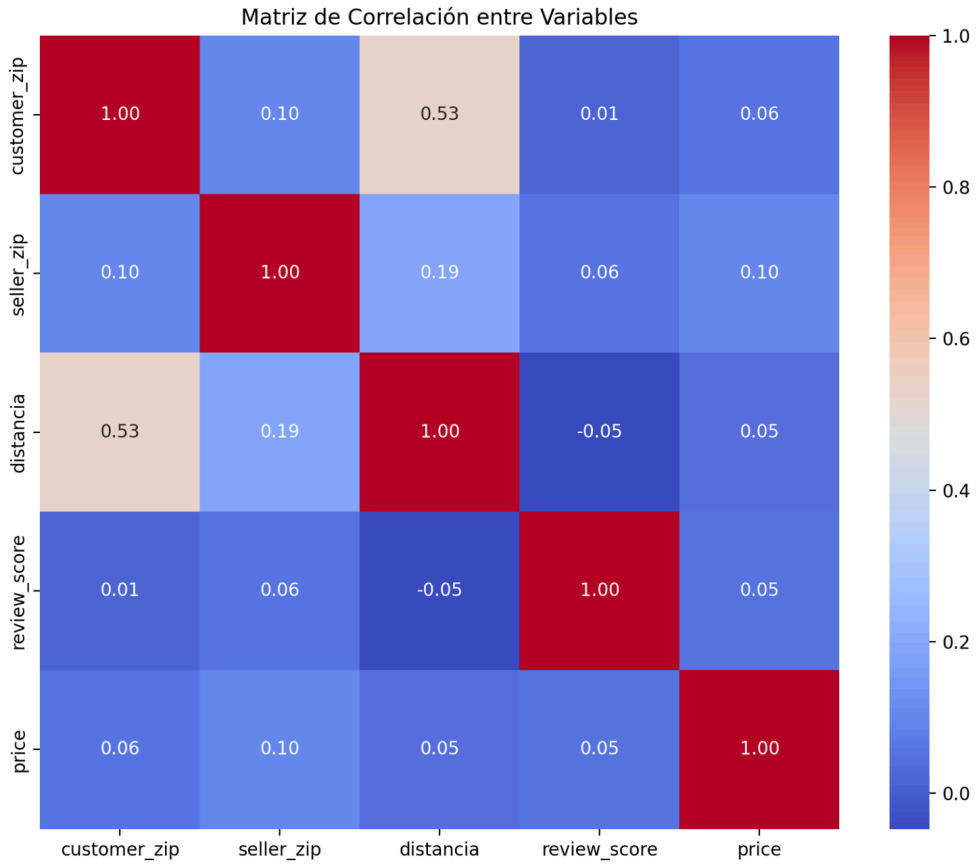
Análisis de Reseñas de Pedidos

Distribución de Calificaciones de Reseñas:

	calificacion	cantidad	porcentaje
0	1	11,282	11
1	2	3,114	3
2	3	8,097	8
3	4	19,007	19
4	5	56,910	58

Error al conectar a la base de datos: 'Cantidad'

Matriz de Correlación entre Variables

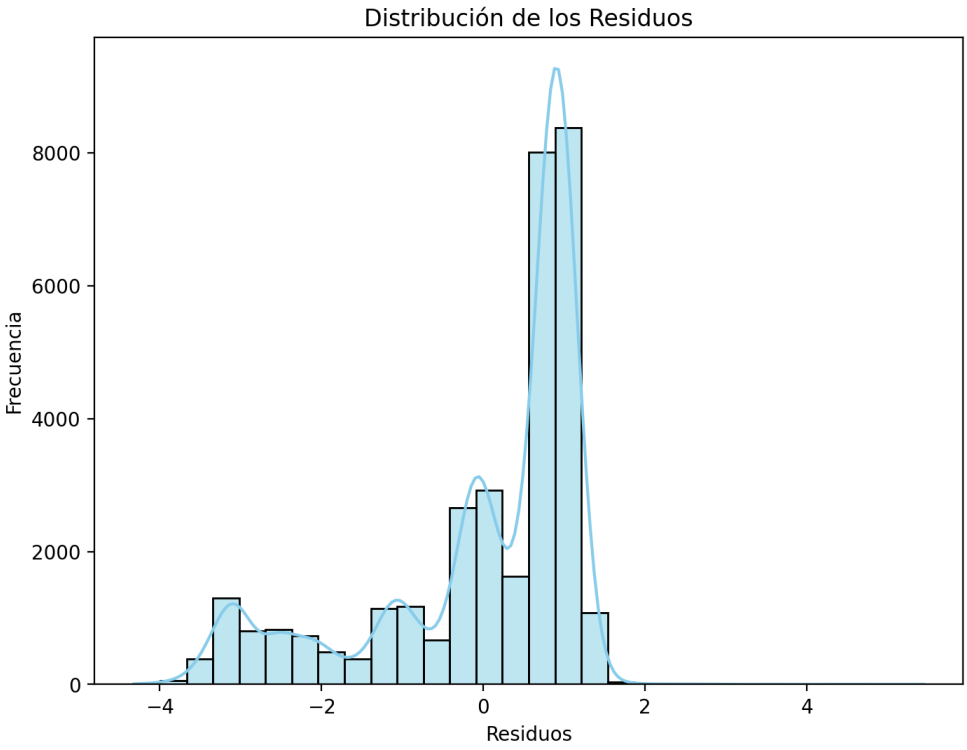


1. Hay una alta correlación positiva entre el código postal del cliente y el del vendedor. por ello se identifica que que los clientes y vendedores tienden a estar geográficamente cercanos
2. Hay una correlación negativa significativa entre la distancia y la puntuación de la reseña. Los clientes que están más lejos del vendedor tienden a dar peores puntuaciones, posiblemente debido a tiempos de entrega más largos o mayores costos de envío. price vs seller_zip (0.99):
3. El precio tiene una correlación casi perfecta con el código postal del vendedor, ciertos vendedores en áreas específicas tienden a vender productos a precios similares por ello se indica que los precios están asociados a ciertas regiones.

Modelo de Regresión Lineal: Predicción del Puntaje de Reseña

Error Cuadrático Medio (MSE): 1.71

Coeficiente de Determinación (R^2): 0.05



1. Concentración de Residuos: La mayor parte de los residuos está concentrada alrededor de 0, lo que indica que el modelo predice razonablemente bien la mayoría de las veces, pero hay cierta dispersión.

2. Distribución Asimétrica: Existe una asimetría en la distribución de los residuos, con algunos valores concentrándose más hacia residuos ligeramente negativos, lo que podría sugerir que el modelo tiende a subestimar ligeramente el puntaje de las reseñas en algunos casos.

3. Picos en la Distribución: Los picos en torno a 0 y 1 indican que el modelo realiza muchas predicciones con errores pequeños, pero también hay varias predicciones con errores moderados, visibles en los picos secundarios..

Ajuste del Modelo de Regresión Lineal y Obtención de Coeficientes

Coeficientes de la Regresión Lineal

	Coeficientes
seller_zip	0
price	0
delivery_delay	-0.0309

Intercepto del Modelo

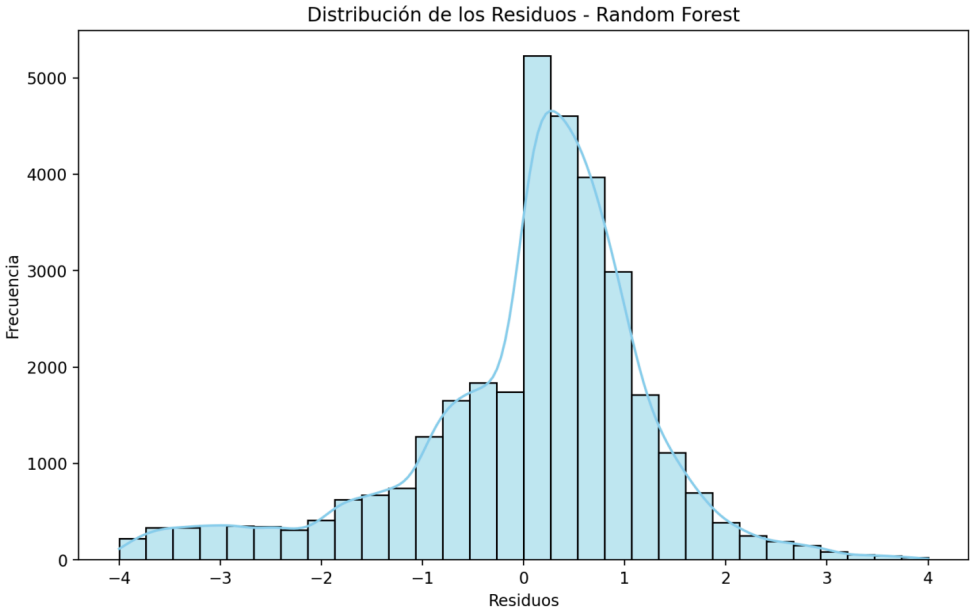
Intercepto: 3.7138252083899777

El modelo sugiere que el retraso en la entrega tiene un impacto negativo en la puntuación de las reseñas, mientras que el código postal del vendedor y el precio del producto no parecen influir significativamente en la puntuación de las reseñas. Sin embargo, la magnitud del impacto del retraso es pequeña, y otros factores no capturados por el modelo podrían estar influyendo en las calificaciones.

Evaluación del Modelo de Random Forest

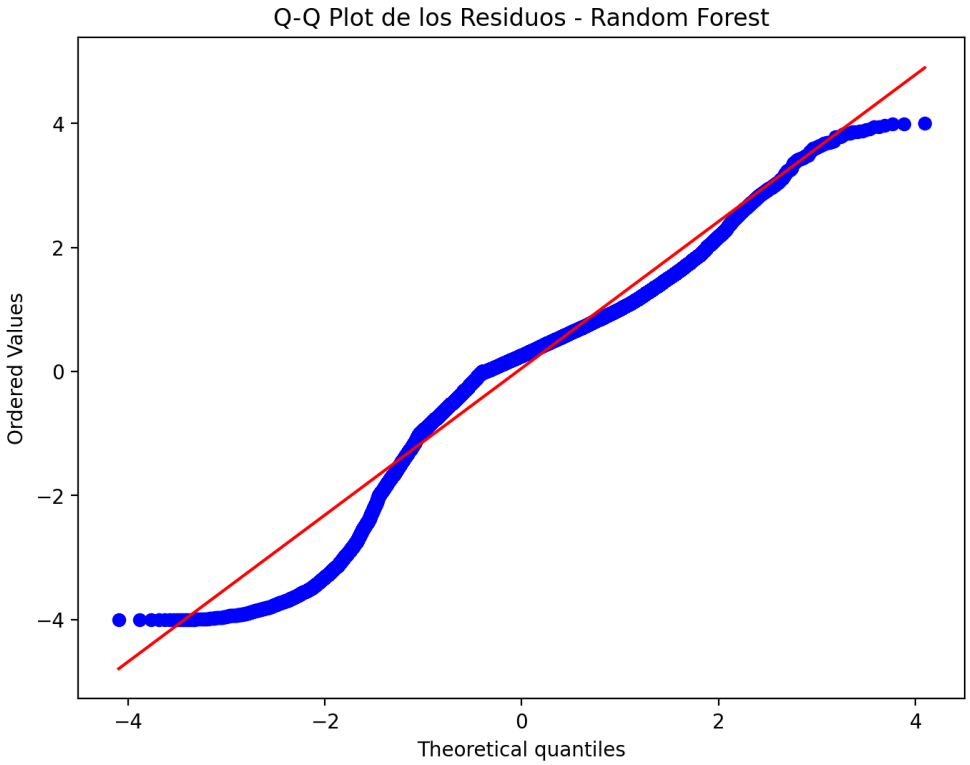
Random Forest - Error Cuadrático Medio (MSE): 1.501967536152944

Random Forest - Coeficiente de Determinación (R^2): 0.169163741321128



El modelo de Random Forest tiene un rendimiento intermedio según los valores de MSE y R^2. La distribución de los residuos muestra una tendencia a un ajuste decente, el bajo valor de R^2 sugiere que el modelo no captura toda la complejidad del fenómeno que se está tratando de predecir. Es por ello el modelo no lineal o más profundo pueda ser más adecuado para mejorar la precisión.

Q-Q Plot de los Residuos - Random Forest



Los puntos en el Q-Q plot no siguen una línea recta, sino que se desvían significativamente de la línea roja diagonal (que representa la distribución normal). Esta curvatura indica que los residuos del modelo

Random Forest no siguen una distribución normal.

Análisis culminado.