In [56]:
```
# Import necessary libraries
import pandas as pd
```

In [58]:
```
# Load datasets into DataFrames
df_orders = pd.read_csv("C:/Users/suneh/Downloads/orders.csv")
df_order_products = pd.read_csv("C:/Users/suneh/Downloads/order_products.cs
```

In [59]:
```
# Display the first few rows of both DataFrames
print("First few rows of df_orders:")
display(df_orders.head())
```

```
First few rows of df_orders:
```

	order_id	customer_id	order_dow	order_hour_of_day	days_since_prior_orc
0	2OHUSWKKX9XC	J3MPRZEX	4.0	14.0	2
1	VBX3BU6LVY01	R69KH7XQ	4.0	14.0	1
2	POCGUIRDYNIZ	TKY3AG0E	5.0	13.0	1
3	BT1GH8NQ9DBF	5FS7WI9O	2.0	11.0	
4	D2RZ7J9N4OHN	6L0KF08H	5.0	14.0	

In [62]:
```
print("\nFirst few rows of df_order_products:")
display(df_order_products.head())
```

```
First few rows of df_order_products:
```

	order_id	product_id	quantity	unit_price	customer_id
0	TZYV9JHJX14K	0QQNAB5GK4	5	3.84	MRY7YWVY
1	0NBB6B7Y0ZVG	RD7LCX95LR	2	34.72	VC3CKQVW
2	T82MXR80NOYI	DAJQS7HHQW	1	17.10	VZ2MJ3GF
3	X3JKVCWSM9CV	V45HC2WDGA	5	7.66	1HYGFX9R
4	KCX24HB2PH9N	Z7XQNHH1OP	3	49.42	LMEVQXKO

Results Explanation

1. Review of `df_orders` and `df_order_products`:

- `df_orders.head()` shows the first few rows of the orders dataset, which contains information like `order_id`, `customer_id`, `order_dow` (day of the week), `order_hour_of_day`, and whether a coupon was used.

- `df_order_products.head()` reveals the first few rows of the product purchase records, showing `order_id`, `product_id`, `quantity`, and the price per unit of each product in the order.

2. Structure of DataFrames using `info()` function:

- `df_orders.info()` returns:
  - Number of entries (rows) and columns in the dataset.
  - Column names and their respective data types (like `int64`, `float64`, `object`).
  - Non-null values per column, which helps identify missing data if any.
- `df_order_products.info()` provides similar insights for the product-level dataset, confirming that multiple products can be linked to the same `order_id`.

By reviewing the `info()` results, we can determine:

- **Data Quality**: Are there any missing values that need handling?
- **Data Types**: Are the data types appropriate (e.g., `object` for categorical variables or `int` for numerical ones)?
- **Data Size**: How large the datasets are, which helps in planning the next analysis steps.

In [67]:
```
# Step 1: Find how many missing values each column contains
print("Missing values before cleaning:")
print(df_orders.isnull().sum())
```

```
Missing values before cleaning:
order_id                  64
customer_id               69
order_dow                 68
order_hour_of_day         55
days_since_prior_order    62
coupon_use                72
dtype: int64
```

In [69]:
```
# Step 2: Replace missing values in 'order_id' with 'unknown_order'
df_orders['order_id'].fillna('unknown_order', inplace=True)
```

```
C:\Users\suneh\AppData\Local\Temp\ipykernel_24824\2541313010.py:2: FutureWar
ning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df_orders['order_id'].fillna('unknown_order', inplace=True)
```

In [71]:
```
# Step 3: Replace missing values in 'customer_id' with 'unknown_customer'
df_orders['customer_id'].fillna('unknown_customer', inplace=True)
```

```
C:\Users\suneh\AppData\Local\Temp\ipykernel_24824\523455206.py:2: FutureWarn
ing: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df_orders['customer_id'].fillna('unknown_customer', inplace=True)
```

In [73]:
```
# Step 4: Replace missing values in 'days_since_prior_order' with the colum
mean_days = df_orders['days_since_prior_order'].mean()
df_orders['days_since_prior_order'].fillna(mean_days, inplace=True)
```

```
C:\Users\suneh\AppData\Local\Temp\ipykernel_24824\3124581164.py:3: FutureWar
ning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df_orders['days_since_prior_order'].fillna(mean_days, inplace=True)
```

In [75]:
```
# Step 5: Check missing values after replacements
print("Missing values after replacements:")
print(df_orders.isnull().sum())
```

```
Missing values after replacements:
order_id                   0
customer_id                0
order_dow                 68
order_hour_of_day         55
days_since_prior_order     0
coupon_use                72
dtype: int64
```

In [77]:
```python
# Step 6: Drop rows with any remaining missing values
df_orders.dropna(inplace=True)

# Verify if there are any remaining missing values
print("Missing values after dropping rows with missing values:")
print(df_orders.isnull().sum())
```

```
Missing values after dropping rows with missing values:
order_id                  0
customer_id               0
order_dow                 0
order_hour_of_day         0
days_since_prior_order    0
coupon_use                0
dtype: int64
```

Explanation of Code:

1. **Checking Missing Values:**
   We used `df_orders.isnull().sum()` to display the number of missing values in each column.

2. **Handling Missing Values:**

   - Replaced missing values in the `order_id` column with `'unknown_order'`.
   - Replaced missing values in the `customer_id` column with `'unknown_customer'`.
   - Replaced missing values in the `days_since_prior_order` column with the mean value of that column using the `mean()` function.

3. **Re-verifying Missing Values:**
   After the replacements, we checked again for missing values. If any are left, we proceeded to drop those rows using `dropna()`.

4. **Final Check:**
   After dropping rows with missing values, we verified that no missing values remain.

This process ensures that missing data is either filled with appropriate values or removed, depending on the context.

In [82]:
```
# Step 1: Create a DataFrame for orders with coupon use
df_orders_coupon = df_orders[df_orders['coupon_use'] == 'yes']
df_orders_coupon.head()
```

Out[82]:

	order_id	customer_id	order_dow	order_hour_of_day	days_since_prior_
2	POCGUIRDYNIZ	TKY3AG0E	5.0	13.0	
13	56ASBZ5F5DKQ	V3IYCE83	6.0	9.0	
16	AK1YW438GPZB	8AW3776P	0.0	19.0	
20	CQ6LVV3H007U	1M9MWA0U	4.0	20.0	
21	3616ESD8Q132	LIGF0U0Z	1.0	19.0	

In [84]:
```
# Step 2: Calculate the mean value of 'days_since_prior_order' for coupon u
mean_days_coupon = df_orders_coupon['days_since_prior_order'].mean()
print(f"Mean days since prior order (coupon used): {mean_days_coupon:.2f}")
```

```
Mean days since prior order (coupon used): 9.93
```

In [86]:
```
# Step 3: Create a DataFrame for orders without coupon use
df_orders_no_coupon = df_orders[df_orders['coupon_use'] == 'no']
df_orders_no_coupon.head()
```

Out[86]:

	order_id	customer_id	order_dow	order_hour_of_day	days_since_prior_c
0	2OHUSWKKX9XC	J3MPRZEX	4.0	14.0	
1	VBX3BU6LVY01	R69KH7XQ	4.0	14.0	
3	BT1GH8NQ9DBF	5FS7WI9O	2.0	11.0	
4	D2RZ7J9N4OHN	6L0KF08H	5.0	14.0	
5	WUPRXZY2OAYG	EEIOSKAP	6.0	20.0	

In [88]:
```
# Step 4: Calculate the mean value of 'days_since_prior_order' for non-coup
mean_days_no_coupon = df_orders_no_coupon['days_since_prior_order'].mean()
print(f"Mean days since prior order (no coupon used): {mean_days_no_coupon:
```

```
Mean days since prior order (no coupon used): 18.72
```

Explanation:

Selecting DataFrames by Coupon Use: We created two DataFrames: df_orders_coupon (where coupon_use is 'yes') and df_orders_no_coupon (where coupon_use is 'no').

Calculating Mean Days Since Prior Order: We used the mean() function to compute the average days_since_prior_order for both groups.

Analysis: Is the Use of Coupon Associated with Higher/Lower Order Frequency?

- **Mean Days (Coupon Used):** [Insert Value from Code Output]
- **Mean Days (No Coupon Used):** [Insert Value from Code Output]

Interpretation:

- If the **mean days since prior order** for coupon users is **lower** than that for non-coupon users, it suggests that offering coupons **increases order frequency** (customers place orders more frequently).
- Conversely, if the mean days are **higher** for coupon users, it implies that customers do not necessarily order more frequently despite the availability of coupons.

Based on the results:

- [Insert your conclusion based on the values]

Example: "Since the mean days since prior order are lower for coupon users, the data suggests that offering coupons is associated with **higher order frequency**."

In [94]:
```
# Group by 'order_dow' and count the total number of orders for each day
orders_per_day = df_orders.groupby('order_dow')['order_id'].count()

# Display the result as a Pandas Series
print("Total number of orders for each day of the week:")
print(orders_per_day)
```

```
Total number of orders for each day of the week:
order_dow
0.0    4669
1.0    1618
2.0     744
4.0    1561
5.0    2989
6.0    3851
Name: order_id, dtype: int64
```

Explanation: Grouping by 'order_dow': We use the groupby() function on the order_dow column to divide the orders based on the day of the week.

Counting Orders: For each group (day of the week), we use the count() function on the order_id column to count the total number of orders.

Output: The result is displayed as a Pandas Series showing the total number of orders for each day of the week, with the days represented by integers (0 for Sunday, 1–5 for Monday to Friday, and 6 for Saturday)

In [99]:
```
# Create the 'revenue' column by multiplying 'quantity' with 'unit_price'
df_order_products['revenue'] = df_order_products['quantity'] * df_order_prc
df_order_products.head()
```

Out[99]:

	order_id	product_id	quantity	unit_price	customer_id	revenue
0	TZYV9JHJX14K	0QQNAB5GK4	5	3.84	MRY7YWVY	19.20
1	0NBB6B7Y0ZVG	RD7LCX95LR	2	34.72	VC3CKQVW	69.44
2	T82MXR80NOYI	DAJQS7HHQW	1	17.10	VZ2MJ3GF	17.10
3	X3JKVCWSM9CV	V45HC2WDGA	5	7.66	1HYGFX9R	38.30
4	KCX24HB2PH9N	Z7XQNHH1OP	3	49.42	LMEVQXKO	148.26

In [101…
```
# Display the first few rows of the updated DataFrame
print("First few rows of the updated df_order_products DataFrame:")
display(df_order_products.head())
```

```
First few rows of the updated df_order_products DataFrame:
```

	order_id	product_id	quantity	unit_price	customer_id	revenue
0	TZYV9JHJX14K	0QQNAB5GK4	5	3.84	MRY7YWVY	19.20
1	0NBB6B7Y0ZVG	RD7LCX95LR	2	34.72	VC3CKQVW	69.44
2	T82MXR80NOYI	DAJQS7HHQW	1	17.10	VZ2MJ3GF	17.10
3	X3JKVCWSM9CV	V45HC2WDGA	5	7.66	1HYGFX9R	38.30
4	KCX24HB2PH9N	Z7XQNHH1OP	3	49.42	LMEVQXKO	148.26

In [103…
```
# Calculate the total revenue by summing the 'revenue' column
total_revenue = df_order_products['revenue'].sum()

# Display the total revenue
print(f"Total revenue: ${total_revenue:.2f}")
```

```
Total revenue: $8744724.59
```

Explanation: *Creating the 'revenue' Column:

The revenue for each row is calculated as: revenue = quantity unit_price. This new column is added directly to the DataFrame df_order_products. Displaying the First Few Rows:

*We used the head() function to verify that the new column has been added correctly.

*Calculating the Total Revenue:

The total revenue is calculated by summing the values in the revenue column using sum().

In [108…

```
# Select all rows related to the customer with id '0421MWMT'
df_cust_inquiry = df_order_products[df_order_products['customer_id'] == '04

# Display the content of the 'df_cust_inquiry' DataFrame
df_cust_inquiry.head()
```

Out[108…

	order_id	product_id	quantity	unit_price	customer_id	revenue
25733	SDSGL050UESG	HUW6839533	3	40.94	0421MWMT	122.82
45016	SDSGL050UESG	09DBUJNRUV	3	18.55	0421MWMT	55.65
76788	SDSGL050UESG	GFC4XNH9DI	4	3.59	0421MWMT	14.36
90021	SDSGL050UESG	IZRQFKDBMY	5	3.92	0421MWMT	19.60

In [110…

```
# Display the content of the 'df_cust_inquiry' DataFrame
print("Customer's purchase records (ID: 0421MWMT):")
display(df_cust_inquiry)
```

```
Customer's purchase records (ID: 0421MWMT):
```

	order_id	product_id	quantity	unit_price	customer_id	revenue
25733	SDSGL050UESG	HUW6839533	3	40.94	0421MWMT	122.82
45016	SDSGL050UESG	09DBUJNRUV	3	18.55	0421MWMT	55.65
76788	SDSGL050UESG	GFC4XNH9DI	4	3.59	0421MWMT	14.36
90021	SDSGL050UESG	IZRQFKDBMY	5	3.92	0421MWMT	19.60

In [112…

```
# Calculate the total purchase amount by summing the 'revenue' column
total_purchase_amount = df_cust_inquiry['revenue'].sum()
```

```
# Display the total purchase amount
print(f"Total purchase amount for customer '0421MWMT': ${total_purchase_amo
```

```
Total purchase amount for customer '0421MWMT': $212.43
```

Explanation:

*Selecting the Customer's Purchase Records:

*We use boolean indexing to filter all rows in df_order_products where customer_id matches '0421MWMT'. The filtered data is assigned to a new DataFrame called df_cust_inquiry. Displaying the Purchase Records:

*The display() function shows the relevant rows to verify the customer's purchase history. Calculating the Total Purchase Amount:

We sum the values in the revenue column to get the total dollar amount spent by the customer.

In [ ]:

In [ ]:

In [ ]: