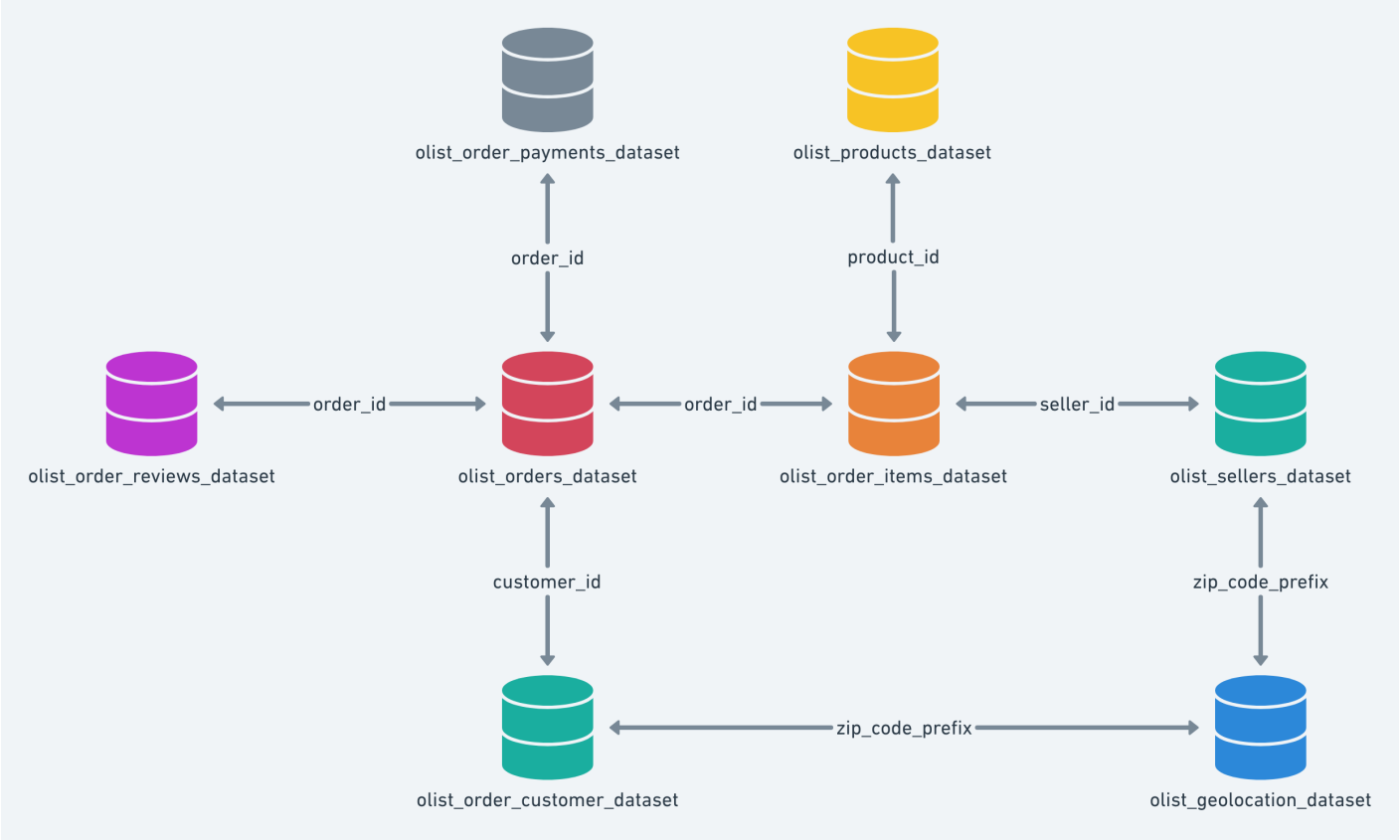
General Data Analysis

Available Data Description

This dataset encompasses **real commercial data** of orders made at the Olist Store - a public compilation detailing 100k orders placed from 2016 to 2018 across diverse marketplaces within Brazil. It's structured to offer a multi-dimensional view of an order, spanning attributes like order status, price, payment, freight performance, customer location, product characteristics, and even customer reviews. What's more, to bolster geographical analysis, a geolocation dataset has been included, mapping Brazilian zip codes to their respective lat/lng coordinates.

The structure of the dataset is meticulously organized, allowing for a holistic understanding of the relationships between various data attributes. This structure is comprehensively represented in the image provided:



The visual representation elucidates the connectivity and relationships between different datasets and their columns. By understanding this structure, we can adeptly navigate the data, ensuring that our analysis is not only accurate but also meaningful. The interconnected nature of this dataset allows us to derive insights from multiple perspectives, painting a comprehensive picture of the e-commerce landscape covered by Olist.

```
# Libraries

from IPython.display import display
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import seaborn as sns
import folium
from folium.plugins import HeatMap
from geopy.geocoders import Nominatim
import matplotlib.ticker as ticker
from statsmodels.tsa.stattools import adfuller


# Available datasets
datasets = [
    "/content/olist_customers_dataset.csv",
    "/content/olist_geolocation_dataset.csv",
    "/content/olist_order_items_dataset.csv",
    "/content/olist_order_payments_dataset.csv",
    "/content/olist_order_reviews_dataset.csv",
    "/content/olist_orders_dataset.csv",
    "/content/olist_products_dataset.csv",
    "/content/olist_sellers_dataset.csv",
    "/content/product_category_name_translation.csv"
]

#  Printig informations
for dataset in datasets:
    print(f"\n\nDataset: {dataset}")

    df = pd.read_csv(dataset)

    # Display
    display(df.head(10))

    # List columns
    print("\nColunas:", df.columns.tolist())
```

```
Dataset: /content/olist_customers_dataset.csv
```

	customer_id	customer_unique_id	customer_zip_co
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	
5	879864dab9bc3047522c92c82e1212b8	4c93744516667ad3b8f1fb645a3116a4	
6	fd826e7cf63160e536e0908c76c3f441	addec96d2e059c80c30fe6871d30d177	
7	5e274e7a0c3809e14aba7ad5aae0d407	57b2a98a409812fe9618067b6b8ebe4f	
8	5adf08e34b2e993982a47070956c5c65	1175e95fb47ddff9de6b2b06188f7e0d	
9	4b7139f34592b3a31687243a302fa75b	9afe194fb833f79e300e37e580171f22	

```
Colunas: ['customer_id', 'customer_unique_id', 'customer_zip_code_prefix', 'customer_ci
```

```
Dataset: /content/olist_geolocation_dataset.csv
```

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geo
0	1037	-23.545621	-46.639292	sao paulo	
1	1046	-23.546081	-46.644820	sao paulo	
2	1046	-23.546129	-46.642951	sao paulo	
3	1041	-23.544392	-46.639499	sao paulo	
4	1035	-23.541578	-46.641607	sao paulo	
5	1012	-23.547762	-46.635361	são paulo	
6	1047	-23.546273	-46.641225	sao paulo	
7	1013	-23.546923	-46.634264	sao paulo	
8	1029	-23.543769	-46.634278	sao paulo	
9	1011	-23.547640	-46.636032	sao paulo	

```
Colunas: ['geolocation_zip_code_prefix', 'geolocation_lat', 'geolocation_lng', 'geoloca
```

```
Dataset: /content/olist_order_items_dataset.csv
```

	order_id	order_item_id	product_id

0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	4
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32f14722fc	9
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e4e10089	
5	00048cc3ae777c65dbb7d2a0634bc1ea	1	ef92defde845ab8450f9d70c526ef70f	
6	00054e8431b9d7675808bcb819fb4a32	1	8d4f2bb7e93e6710a28f34fa83ee7d28	
7	000576fe39319847cbb9d288c5617fa6	1	557d850972a7d6f792fd18ae1400d9b6	
8	0005a1a1728c9d785b8e2b08b904576c	1	310ae3c140ff94b03219ad0adc3c778f	a
9	0005f50442cb953dcd1d21e1fb923495	1	4535b0e1091c278dfd193e5a1d63b39f	

```
Colunas: ['order_id', 'order_item_id', 'product_id', 'seller_id', 'shipping_limit_date'
```

```
Dataset: /content/olist_order_payments_dataset.csv
```

	order_id	payment_sequential	payment_type	payment_installm
0	b81ef226f3fe1789b1e8b2acac839d17	1	credit_card	
1	a9810da82917af2d9aefd1278f1dcfa0	1	credit_card	
2	25e8ea4e93396b6fa0d3dd708e76c1bd	1	credit_card	
3	ba78997921bbcdc1373bb41e913ab953	1	credit_card	
4	42fdf880ba16b47b59251dd489d4441a	1	credit_card	
5	298fcdf1f73eb413e4d26d01b25bc1cd	1	credit_card	
6	771ee386b001f06208a7419e4fc1bbd7	1	credit_card	
7	3d7239c394a212faae122962df514ac7	1	credit_card	
8	1f78449c87a54faf9e96e88ba1491fa9	1	credit_card	
9	0573b5e23cbd798006520e1d5b4c6714	1	boleto	

```
Colunas: ['order_id', 'payment_sequential', 'payment_type', 'payment_installments', 'pa
```

```
Dataset: /content/olist_order_reviews_dataset.csv
```

	review_id	order_id	review_score	re
0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	
1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	
2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	

3	e64fb393e7b32834bb789ff8bb30750e	658677c97b385a9be170737859d3511b	5
4	f7c4243c7fe1938f181bec41a392bdeb	8e6bfb81e283fa7e4f11123a3fb894f1	5
5	15197aa66ff4d0650b5434f1b46cda19	b18dcdf73be66366873cd26c5724d1dc	1
6	07f9bee5d1b850860defd761afa7ff16	e48aa0d2dcec3a2e87348811bcfdf22b	5
7	7c6400515c67679fbee952a7525281ef	c31a859e34e3adac22f376954e19b39d	5
8	a3f6f7f6f433de0aefbb97da197c554c	9c214ac970e84273583ab523dfafd09b	5
9	8670d52e15e00043ae7de4c01cc2fe06	b9bf720beb4ab3728760088589c62129	4

```
Colunas: ['review_id', 'order_id', 'review_score', 'review_comment_title', 'review_comm
```

```
Dataset: /content/olist_orders_dataset.csv
```

	order_id	customer_id	order_status	o
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered	
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered	
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered	
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered	
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daea8866dbdbc4fb7aad2c	delivered	
5	a4591c265e18cb1dcee52889e2d8acc3	503740e9ca751ccdda7ba28e9ab8f608	delivered	
6	136cce7faa42fdb2cefd53fdc79a6098	ed0271e0b7da060a393796590e7b737a	invoiced	
7	6514b8ad8028c9f2cc2374ded245783f	9bdf08b4b3b52b5526ff42d37d47f222	delivered	
8	76c6e866289321a7c93b82b54852dc33	f54a9f0e6b351c431402b8461ea51999	delivered	
9	e69bfb5eb88e0ed6a785585b27e16dbf	31ad1d1b63eb9962463f764d4e6e0c9d	delivered	

```
Colunas: ['order_id', 'customer_id', 'order_status', 'order_purchase_timestamp', 'order
```

```
Dataset: /content/olist_products_dataset.csv
```

	product_id	product_category_name	product_name_lenght	produc
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumaria	40.0	
1	3aa071139cb16b67ca9e5dea641aaa2f	artes	44.0	
2	96bd76ec8810374ed1b65e291975717f	esporte_lazer	46.0	
3	cef67bcfe19066a932b7673e239eb23d	bebes	27.0	
4	9dc1a7de274444849c219cff195d0b71	utilidades_domesticas	37.0	

5	41d3672d4792049fa1779bb35283ed13	instrumentos_musicais	60.0
6	732bd381ad09e530fe0a5f457d81becb	cool_stuff	56.0

General data pre-processing

9	8c92109888e8cdf9d66dc7e463025574	brinquedos	36.0

To better process our data, let's merge these datasets into a comprehensive dataframe named merged_data by aligning common identifiers like 'order_id' and 'seller_id'. After merging, several columns deemed extraneous are dropped to streamline the data.

Lastly, the top 15 rows of the refined merged_data dataset are displayed for quick inspection.

```
# Reading the datasets
customers = pd.read_csv("/content/olist_customers_dataset.csv")
orders = pd.read_csv("/content/olist_orders_dataset.csv")
order_items = pd.read_csv("/content/olist_order_items_dataset.cs
order_payments = pd.read_csv("/content/olist_order_payments_data
order_reviews = pd.read_csv("/content/olist_order_reviews_datase
sellers = pd.read_csv("/content/olist_sellers_dataset.csv")

# Merging the datasets
merged_data = (
    order_items
    .merge(order_payments, on='order_id', how='left')
    .merge(order_reviews, on='order_id', how='left')
    .merge(sellers, on='seller_id', how='left')
    .merge(orders[['order_id', 'customer_id']], on='order_id', h
    .merge(customers[['customer_id', 'customer_city', 'customer_
)
```

```
# Columns to be dropped
columns_to_drop = [
    'order_id', 'order_item_id', 'product_id', 'seller_id', 'shi
    'payment_sequential', 'payment_installments', 'review_commen
    'review_creation_date', 'review_answer_timestamp', 'seller_z
]

# Drop the columns
merged_data = merged_data.drop(columns=columns_to_drop)

# Display the first 15 rows of the updated merged_data dataset
from IPython.display import display
display(merged_data.head(15))
```

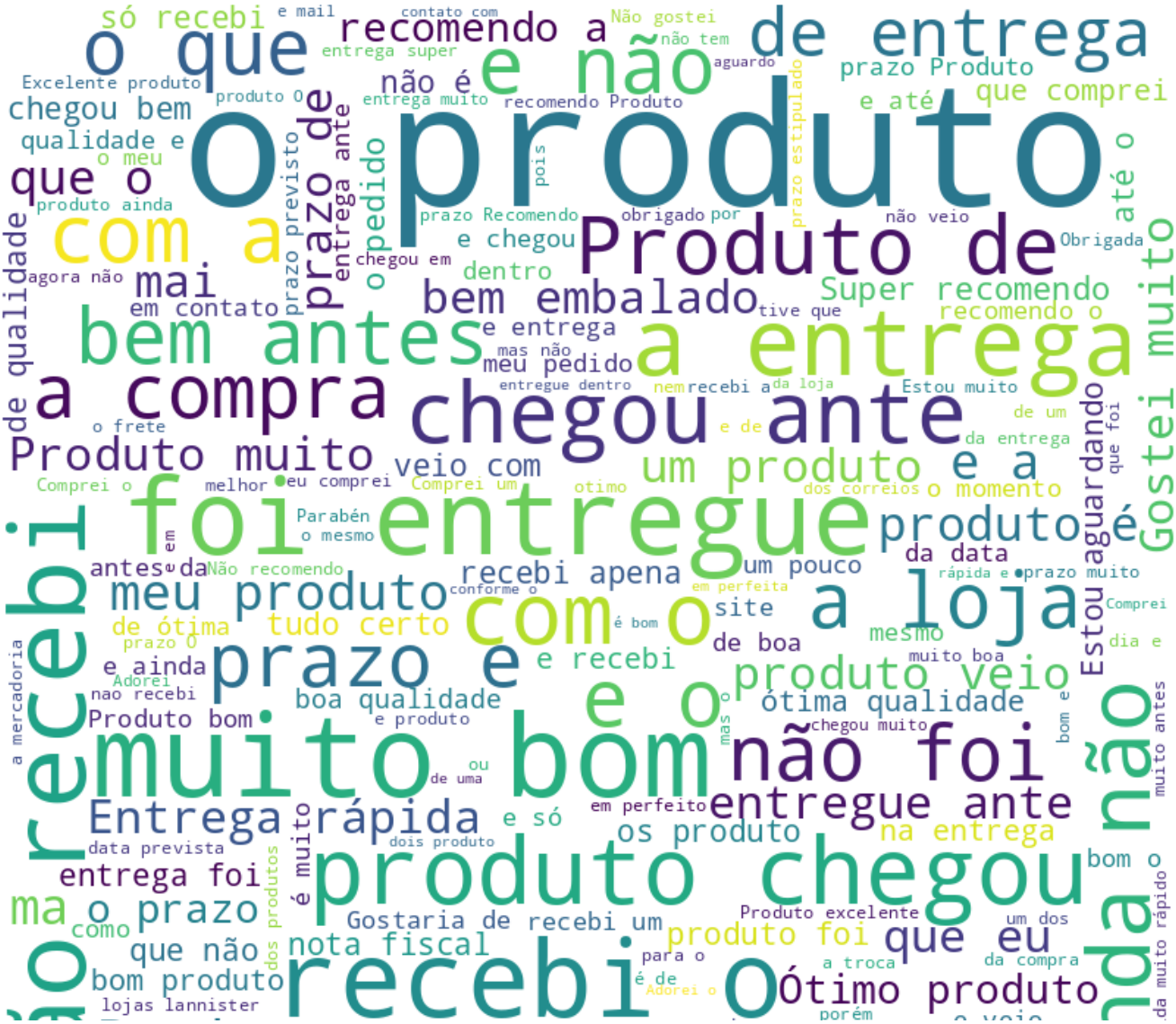
	price	freight_value	payment_type	payment_value	review_score	review_comment_mes
0	58.90	13.29	credit_card	72.19	5.0	Perfeito, produto ent antes do combi
1	239.90	19.93	credit_card	259.83	4.0	
2	199.00	17.87	credit_card	216.87	5.0	Chegou antes do previsto e o produto
3	12.99	12.79	credit_card	25.78	4.0	
4	199.90	18.14	credit_card	218.04	5.0	Gostei pois veio no determin
5	21.90	12.69	boleto	34.59	4.0	
6	19.90	11.85	credit_card	31.75	4.0	
7	810.00	70.75	credit_card	880.75	5.0	
8	145.95	11.65	credit_card	157.60	1.0	Na descrição do pr quando fui efetuar
9	53.99	11.40	credit_card	65.39	4.0	
10	59.99	8.88	credit_card	68.87	5.0	A caixa do produto com uma pec av

Customer's Review Data Visualization

Now let's generate a word cloud from customer review comments in the merged_data dataset. We need to combines all reviews into one string and creates a word cloud, omitting common stopwords.

The resulting word cloud visually emphasizes words mentioned more frequently in the reviews. When analyzing, the larger and bolder words in the word cloud represent common themes or sentiments in customer feedback. These dominant words provide insights into prevailing customer opinions and potential areas of focus for the business.

```
# Removing rows with NaN in the 'review_comment_message' column
merged_data_clean = merged_data.dropna(subset=['review_comment_m

# Combine all messages into a single string
all_reviews = ' '.join(merged_data_clean['review_comment_message

# Create the word cloud
wordcloud = WordCloud(width = 800, height = 800,
                      background_color ='white',
                      stopwords = set(['the', 'of', 'and', 'to',
                      min_font_size = 10).generate(all_reviews)

# Display the word cloud
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



Review Scores Analysis

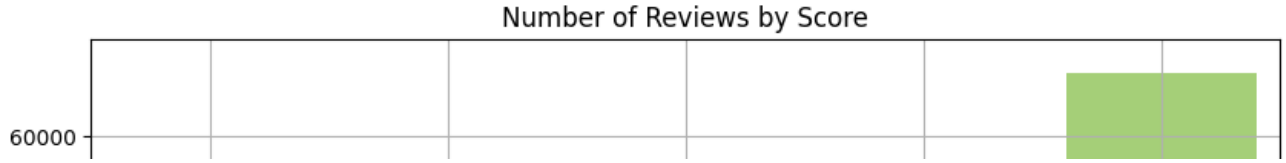
Let's visualize the distribution of review scores in the `merged_data` dataset. After cleaning the data to exclude missing review scores, we need to count the occurrences of each score and displays the results as a bar plot.

To analyze:

1. **Review Distribution**: Examine the height of each bar. The taller the bar, the more frequently that score was given by customers.
2. **Common Scores**: Bars that are notably taller signify scores that are more common. For instance, if the bar for score "5" is the tallest, most reviews are positive.
3. **Areas of Improvement**: If lower scores (e.g., 1 or 2) have significant counts, it indicates areas where the service or product might be lacking and needs attention.
4. **Overall Satisfaction**: An overall view can give a sense of customer satisfaction. A skewed distribution towards higher scores means general contentment.

By interpreting the bar plot, businesses can gauge customer satisfaction levels and identify areas needing improvement.

```
# Remove rows with NaN in the 'review_score' column
merged_data_clean = merged_data.dropna(subset=['review_score'])

# Count the number of occurrences of each review score
score_counts = merged_data_clean['review_score'].value_counts().

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x=score_counts.index, y=score_counts.values, alpha=0
plt.title('Number of Reviews by Score')
plt.ylabel('Number of Reviews', fontsize=12)
plt.xlabel('Review Score', fontsize=12)
plt.grid()
plt.show()
```




```
# Reading the datasets again
customers = pd.read_csv("/content/olist_customers_dataset.csv")
orders = pd.read_csv("/content/olist_orders_dataset.csv")
order_items = pd.read_csv("/content/olist_order_items_dataset.cs
order_payments = pd.read_csv("/content/olist_order_payments_data
order_reviews = pd.read_csv("/content/olist_order_reviews_datase
sellers = pd.read_csv("/content/olist_sellers_dataset.csv")

# Merging the datasets
merged_data = (
    order_items
    .merge(order_payments, on='order_id', how='left')
    .merge(order_reviews, on='order_id', how='left')
    .merge(sellers, on='seller_id', how='left')
    .merge(orders[['order_id', 'customer_id']], on='order_id', h
    .merge(customers[['customer_id', 'customer_city', 'customer_
)

# Columns to be dropped
columns_to_drop = [
    'order_id', 'order_item_id', 'product_id', 'seller_id', 'shi
    'payment_sequential', 'payment_installments', 'review_commen
    'review_creation_date', 'review_answer_timestamp', 'seller_z
]

# Drop the columns
merged_data = merged_data.drop(columns=columns_to_drop)

# Display the first 15 rows of the updated merged_data dataset
from IPython.display import display
display(merged_data.head(15))
```

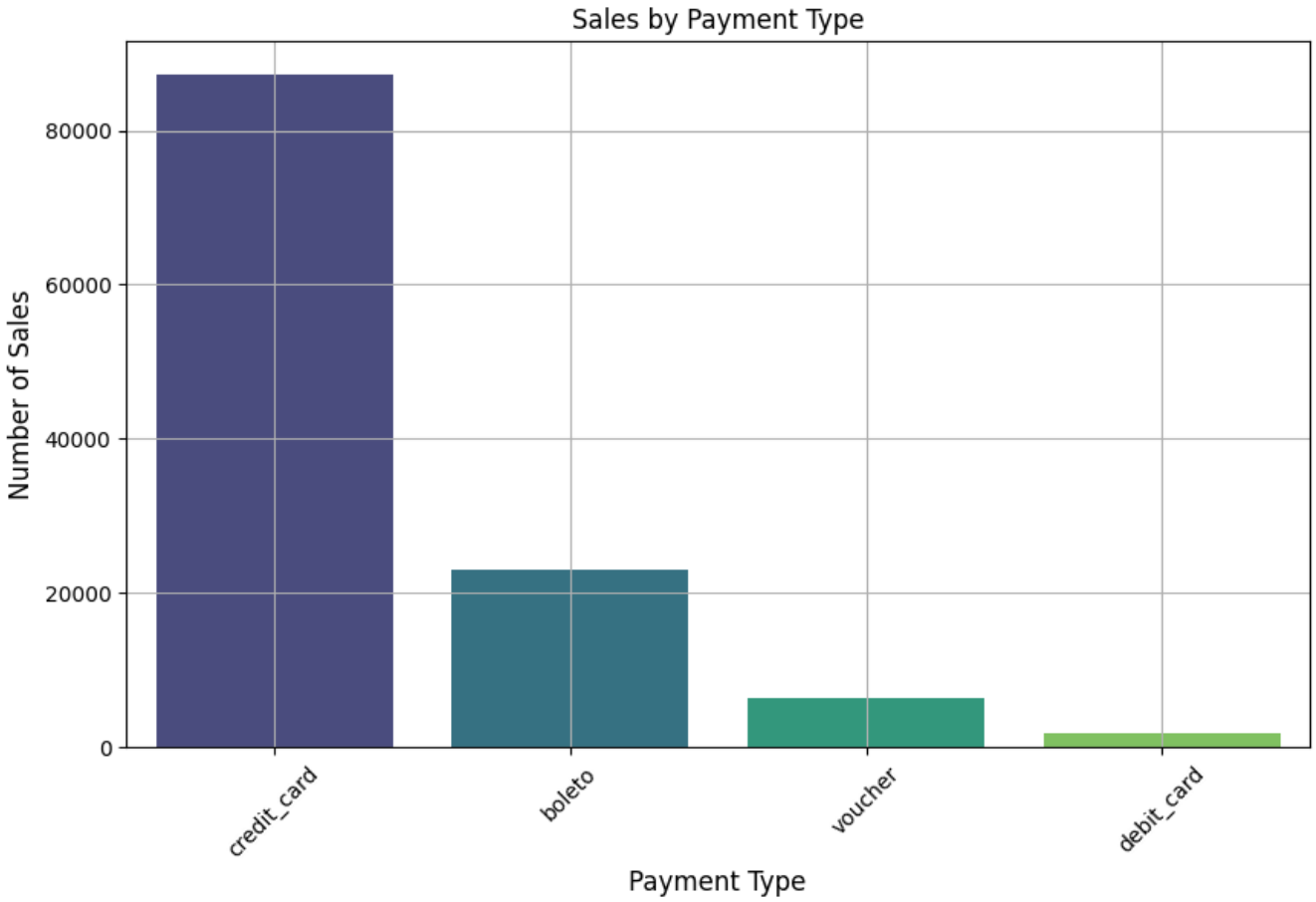
	price	freight_value	payment_type	payment_value	seller_city	seller_state	custo
0	58.90	13.29	credit_card	72.19	volta redonda	SP	ca g
1	239.90	19.93	credit_card	259.83	sao paulo	SP	sant
2	199.00	17.87	credit_card	216.87	borda da mata	MG	para
3	12.99	12.79	credit_card	25.78	franca	SP	
4	199.90	18.14	credit_card	218.04	loanda	PR	varze
5	21.90	12.69	boleto	34.59	ribeirao preto	SP	
6	19.90	11.85	credit_card	31.75	sao paulo	SP	g
7	810.00	70.75	credit_card	880.75	presidente prudente	SP	pra
8	145.95	11.65	credit_card	157.60	sao paulo	SP	
9	53.99	11.40	credit_card	65.39	sao paulo	SP	
10	59.99	8.88	credit_card	68.87	santo andre	SP	
11	45.00	12.98	credit_card	57.98	sao paulo	SP	

Payment Type Visualization

From this plot, businesses can understand customer payment preferences, potentially optimizing operations or sales strategies based on preferred payment methods.

1. **Popular Payment Methods**: Observe which bars are tallest. This indicates which payment methods are most commonly used by customers.
2. **Less Common Methods**: Shorter bars represent less frequently used payment methods. These might be areas where you could potentially increase marketing or offer incentives to boost usage if desired.
3. **Strategic Decisions**: If a less popular method is costly for the business to maintain, it may warrant reconsideration.

```
# Payment Typa Analysis
payment_type_counts = merged_data['payment_type'].value_counts()

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x=payment_type_counts.index, y=payment_type_counts.v
plt.title('Sales by Payment Type')
plt.ylabel('Number of Sales', fontsize=12)
plt.xlabel('Payment Type', fontsize=12)
plt.grid()
plt.xticks(rotation=45)
plt.show()
```

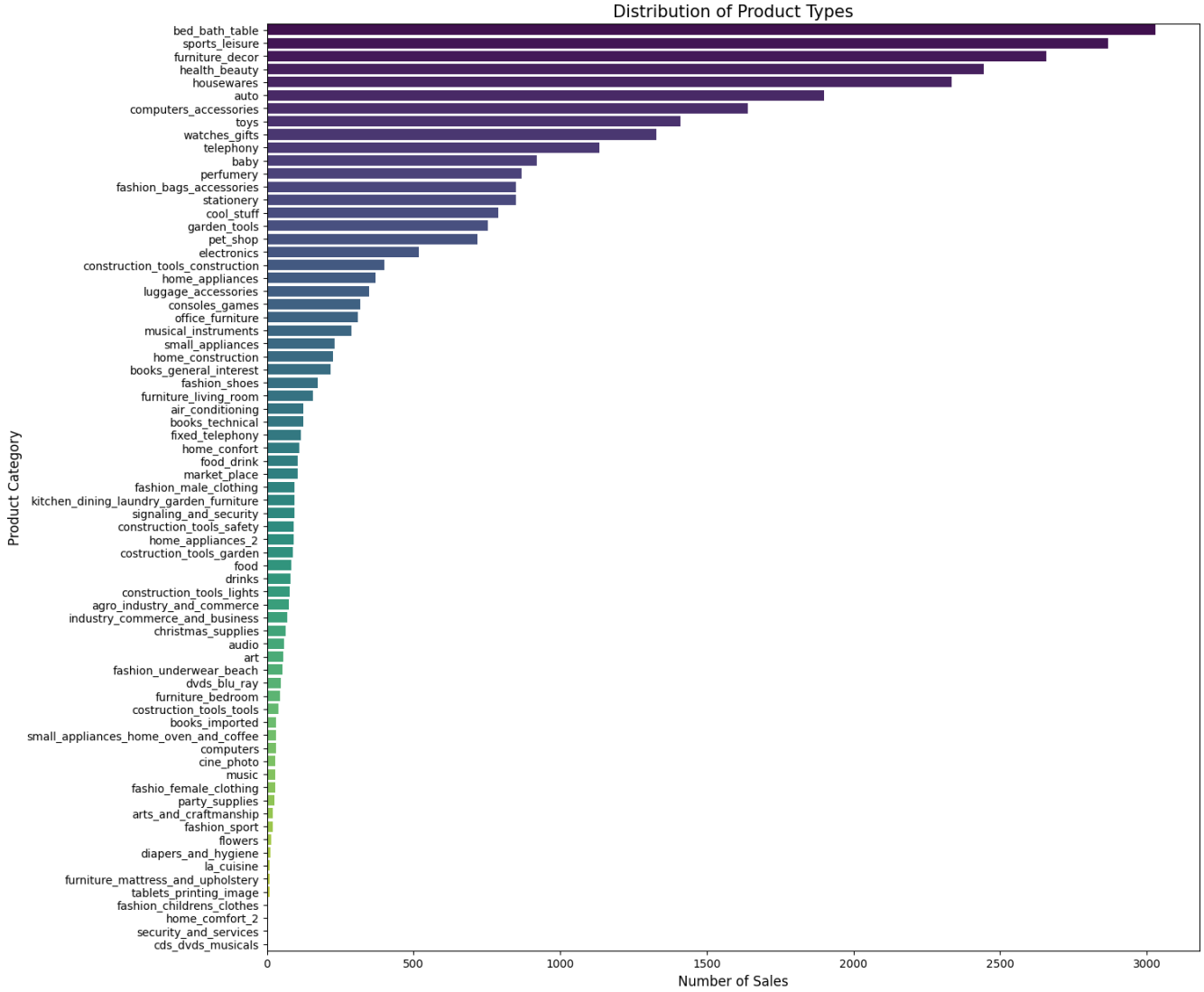


Sales and Revenue Data Analysis

Let's get some insights into the sales distribution and revenue generation of different product types in the Brazilian e-commerce dataset.

Distribution of Product Types: The first visualization showcases the sales distribution across various product categories. It essentially tells us how many times each product type was sold. Using this bar chart, businesses can identify which product types are most popular among their customers and which aren't.

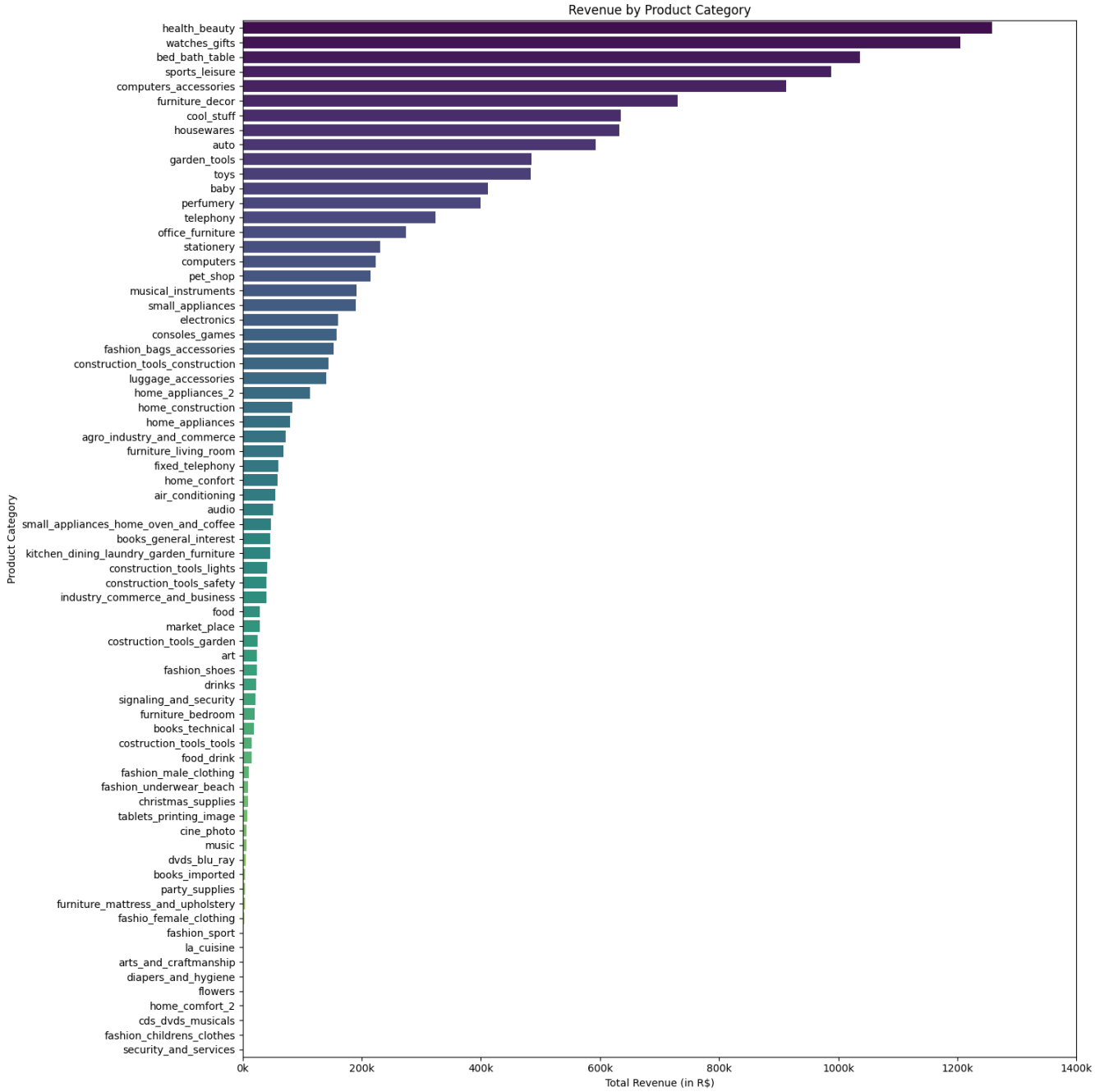
```
# Load datasets
products = pd.read_csv("/content/olist_products_dataset.csv")
category_translation = pd.read_csv("/content/product_category_na

# Merge data
merged_products = pd.merge(products, category_translation, on='p

# Drop the Portuguese category names
merged_products.drop(columns='product_category_name', inplace=Tr

# Count occurrences of each category
category_counts = merged_products['product_category_name_english

# Plot
plt.figure(figsize=(15,15))
sns.barplot(y=category_counts.index, x=category_counts.values, p
plt.xlabel('Number of Sales', fontsize=12)
plt.ylabel('Product Category', fontsize=12)
plt.title('Distribution of Product Types', fontsize=15)
plt.show()
```


Distribution of Product Types
bed_bath_table
sports_leisure
furniture_decor
health_beauty
housewares
auto
computers_accessories
toys
watches_gifts
telephony
baby
perfumery
fashion_bags_accessories
stationery
cool_stuff
garden_tools
pet_shop
electronics
construction_tools_construction
home_appliances
luggage_accessories
consoles_games
office_furniture
musical_instruments
small_appliances
home_construction
books_general_interest
fashion_shoes
furniture_living_room
air_conditioning
books_technical
fixed_telephony
home_confort
food_drink
market_place
fashion_male_clothing
kitchen_dining_laundry_garden_furniture
signaling_and_security
construction_tools_safety
home_appliances_2
costruction_tools_garden
food
drinks
construction_tools_lights
agro_industry_and_commerce
industry_commerce_and_business
christmas_supplies
audio
art
fashion_underwear_beach
dvds_blu_ray
furniture_bedroom
costruction_tools_tools
books_imported
small_appliances_home_oven_and_coffee
computers
cine_photo
music
fashio_female_clothing
party_supplies
arts_and_craftmanship
fashion_sport
flowers
diapers_and_hygiene
la_cuisine
furniture_mattress_and_upholstery
tablets_printing_image
fashion_childrens_clothes
home_comfort_2
security_and_services
cds_dvds_musicals
Product Category
0
500
1000
1500
2000
2500
3000
Number of Sales

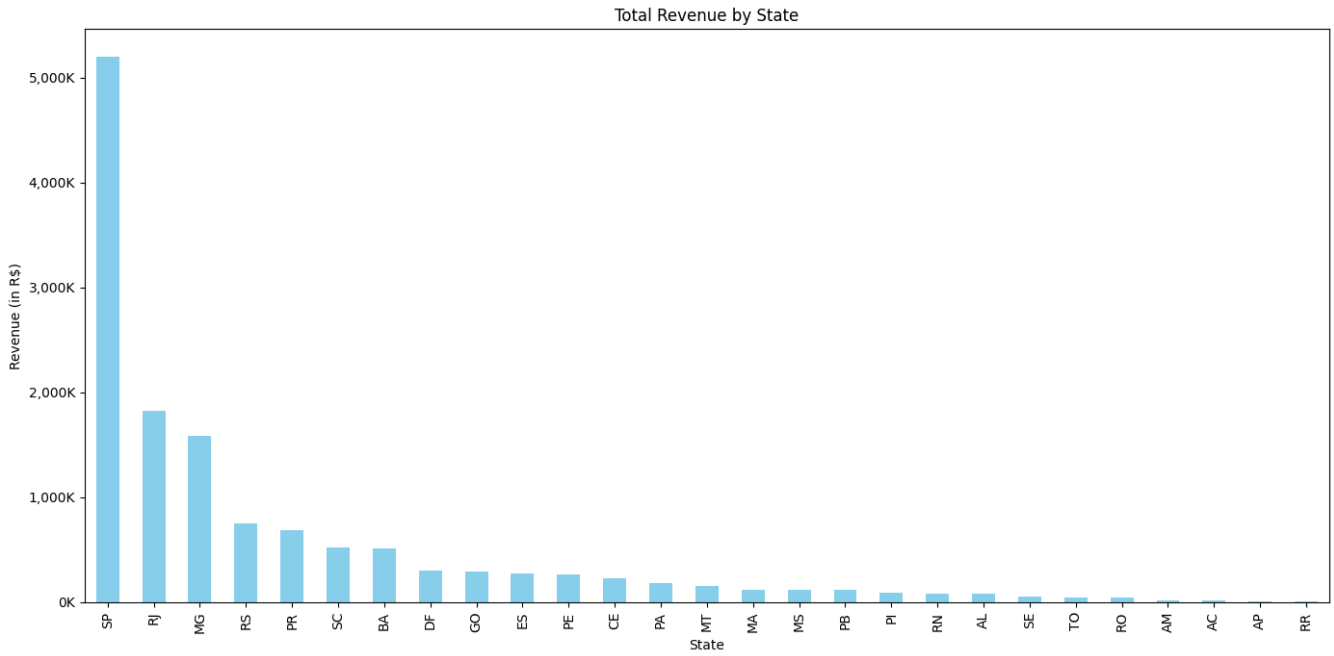
Revenue by Product Category: The second visualization illustrates the revenue generated by each product category. While the sales distribution gives an idea of quantity, the revenue visualization shows the financial impact. A product category might have fewer sales but could still generate high revenue if the product price is high. Conversely, a product type with high sales doesn't necessarily mean it's the top revenue generator.

The two charts together provide a comprehensive understanding of sales and revenue dynamics across different product categories. Businesses can use this information for inventory management, targeted marketing, and financial forecasting. Additionally, spotting a high-demand, high-revenue product can guide future procurement or production decisions.

```
# Revenue by Category Analysis

#Load Dataframes
products_df = pd.read_csv("/content/olist_products_dataset.csv")
translations_df = pd.read_csv("/content/product_category_name_tr
order_items_df = pd.read_csv("/content/olist_order_items_dataset

# Merge datasets
merged_products = pd.merge(order_items_df, products_df, on="prod
merged_products = pd.merge(merged_products, translations_df, on=

# Group by product category (in English) and sum the prices
revenue_by_category = merged_products.groupby("product_category_

# Plot
plt.figure(figsize=(15, 15))
sns.barplot(y=revenue_by_category.index, x=revenue_by_category.v
plt.title('Revenue by Product Category')
plt.xlabel('Total Revenue (in R$)')
plt.ylabel('Product Category')

# Adjust x-tick labels
ticks = plt.xticks()[0]
labels = [f"{int(tick/1000)}k" for tick in ticks]
plt.xticks(ticks, labels)

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


Revenue by Product Category
health_beauty
watches_gifts
bed_bath_table
sports_leisure
computers_accessories
furniture_decor
cool_stuff
housewares
auto
garden_tools
toys
baby
perfumery
telephony
office_furniture
stationery
computers
pet_shop
musical_instruments
small_appliances
electronics
consoles_games
fashion_bags_accessories
construction_tools_construction
luggage_accessories
home_appliances_2
home_construction
home_appliances
agro_industry_and_commerce
furniture_living_room
fixed_telephony
home_confort
air_conditioning
audio
small_appliances_home_oven_and_coffee
books_general_interest
kitchen_dining_laundry_garden_furniture
construction_tools_lights
construction_tools_safety
industry_commerce_and_business
food
market_place
costruction_tools_garden
art
fashion_shoes
drinks
signaling_and_security
furniture_bedroom
books_technical
costruction_tools_tools
food_drink
fashion_male_clothing
fashion_underwear_beach
christmas_supplies
tablets_printing_image
cine_photo
music
dvds_blu_ray
books_imported
party_supplies
furniture_mattress_and_upholstery
fashio_female_clothing
fashion_sport
la_cuisine
arts_and_craftmanship
diapers_and_hygiene
flowers
home_comfort_2
cds_dvds_musicals
fashion_childrens_clothes
security_and_services
Product Category
0k
200k
400k
600k
800k
1000k
1200k
1400k
Total Revenue (in R$)

```
orders_df = pd.read_csv('/content/olist_orders_dataset.csv')

# Merge order_items with orders to get the customer_id for each
merged_with_orders = pd.merge(order_items, orders_df, on='order_

# Merge the above result with the customers DataFrame using cust
final_merged_data = pd.merge(merged_with_orders, customers, on='

# Calculate total revenue by state
revenue_by_state = final_merged_data.groupby('customer_state')['

# Plot
plt.figure(figsize=(14,7))
revenue_by_state.plot(kind='bar', color='skyblue')

plt.title('Total Revenue by State')
plt.ylabel('Revenue (in R$)')
plt.xlabel('State')

# Adjust the y-axis ticks to display in terms of hundreds of tho
formatter = ticker.FuncFormatter(lambda x, pos: '{:,.0f}K'.forma
plt.gca().yaxis.set_major_formatter(formatter)

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




```
# Calculate the total revenue
total_revenue = merged_data['price'].sum()

# Calculate the total freight cost
total_freight_cost = merged_data['freight_value'].sum()

# Print the results
print(f"Total Revenue: R${total_revenue:,.2f}")
print(f"Total Freight Cost: R${total_freight_cost:,.2f}")
```

```
Total Revenue: R$14,273,699.65
Total Freight Cost: R$2,370,031.65
```

Review Score x Delivery Time Correlation Analysis

The analysis aims to investigate the relationship between delivery time and the review score given by customers.

Feature Engineering:

- The delivery duration, or how many days it took for a product to be delivered after it was approved, is calculated.

Correlation Calculation:

- A correlation coefficient is computed between delivery duration and review score. The correlation coefficient ranges from -1 to 1. If it's close to 1, it suggests that as delivery time increases, the review score also increases (which is unlikely). A correlation close to -1 suggests that as delivery time increases, the review score decreases (indicating customer dissatisfaction with longer delivery times). A correlation close to 0 indicates no strong linear relationship between the two variables.

In essence, this analysis seeks to answer: "Do longer delivery times influence customers to give lower review scores?"

The computed correlation value provides insight into this question.

```
# Load datasets
orders_data = pd.read_csv('/content/olist_orders_dataset.csv')
reviews_data = pd.read_csv('/content/olist_order_reviews_dataset

# Merge the datasets
merged_data = pd.merge(orders_data, reviews_data, on='order_id',

# Filter orders with status 'delivered'
delivered_orders = orders[orders['order_status'] == 'delivered']

# Convert date columns to datetime format
delivered_orders['order_approved_at'] = pd.to_datetime(delivered
delivered_orders['order_delivered_customer_date'] = pd.to_dateti

# Calculate the delivery duration in days
delivered_orders['delivery_duration'] = (delivered_orders['order

# Assuming you've read the olist_order_reviews_dataset.csv into
merged_data = delivered_orders.merge(reviews_data[['order_id', '

# Compute correlation
correlation = merged_data['delivery_duration'].corr(merged_data[

print(f"Correlation between delivery time and customer reviews:
```

```
Correlation between delivery time and customer reviews: -0.33
```

The correlation of -0.33 between delivery time and customer reviews suggests a moderate negative relationship between these two variables. In practical terms, this means that, on average, as the delivery time increases, the customer's rating tends to be lower. Conversely, faster deliveries tend to result in higher customer reviews.

This correlation can be interpreted as an indication that delivery efficiency is an important factor for customer satisfaction. The faster an order is delivered after its approval, the more satisfied a customer is likely to be, reflecting that in their review.

However, it's important to note that a correlation of -0.33, although suggesting a trend, does not indicate a perfectly inverse relationship.

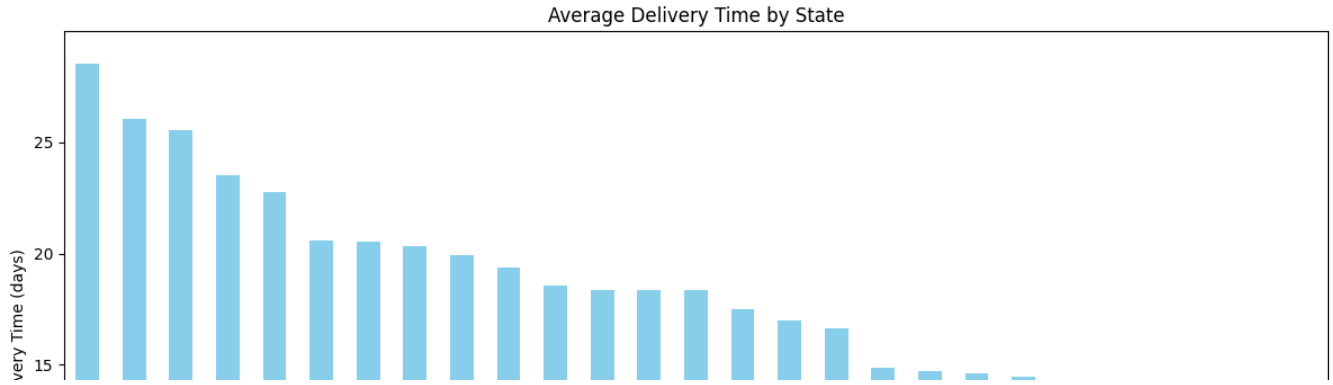
Avarage Delivery Time by Customer's State Visualization

From the "Average Delivery Time by State" bar chart, we can discern several insights about the distribution of delivery durations across different states:

1. **Variability in Delivery Time**: The delivery time varies significantly across states. Some states experience notably longer average delivery times, while others benefit from quicker delivery services.
2. **Geographic Implications**: Typically, states that are farther from distribution centers or main logistic routes may have longer delivery times. This could be due to logistical challenges such as transport availability, road infrastructure, or geographic barriers.
3. **Operational Efficiency**: Differences in delivery times might also be indicative of the operational efficiency of the sellers or shipping partners catering to particular regions. States with quicker delivery times could be serviced by more efficient or better-equipped sellers and shippers.
4. **Customer Experience**: States with prolonged delivery durations might have customers who are more prone to dissatisfaction due to the wait. Conversely, states with shorter delivery times could witness higher customer satisfaction rates, as we've seen from the negative correlation between delivery time and review scores.
5. **Potential for Business Strategy**: For e-commerce businesses, this chart can be invaluable. Recognizing which states have extended delivery durations can inform strategies for improvement. For instance, businesses might consider opening additional distribution centers in states with longer delivery times or partnering with local delivery services to optimize the delivery process.

In summary, the chart offers a clear visual representation of delivery performance across states. This information can be crucial for e-commerce platforms, sellers, and logistics partners aiming to enhance their service quality and customer experience.

```
# Load the dataset
customers_df = pd.read_csv('/content/olist_customers_dataset.csv

# Now, merge the datasets based on customer_id
merged_df = pd.merge(delivered_orders, customers_df[['customer_i

# Calculate the average delivery duration by state
avg_delivery_by_state = merged_df.groupby('customer_state')['del

# Plotting
plt.figure(figsize=(12, 7))
avg_delivery_by_state.plot(kind='bar', color='skyblue')
plt.title('Average Delivery Time by State')
plt.xlabel('State')
plt.ylabel('Average Delivery Time (days)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Temporal Trend Analysis

Analyzing patterns over time can shed light on past performance, indicate current status, and guide future strategies. Here's why it's particularly important:

1. **Identification of Patterns**:
    - Regular trends, seasonality, or cyclical patterns might emerge from temporal data. For instance, sales might spike during the holiday season every year or during particular hours of the day.
2. **Business Forecasting**:
    - By analyzing past trends, businesses can predict future behavior. If sales rise every summer, stock can be increased in preparation for the next year.
3. **Optimization of Operations**:
    - If freight costs spike during a particular month, businesses can negotiate better deals or find alternative suppliers or routes.
4. **Performance Evaluation**:
    - By comparing the revenue or sales quantity from one period to another (month-to-month or year-to-year), businesses can evaluate the effectiveness of their strategies and initiatives.
5. **Budgeting and Planning**:
    - Insights from temporal trends help in allocating resources, budgeting, and financial planning.
6. **Response to External Factors**:
    - Temporal analysis might reveal how external events (like a global pandemic or economic downturn) impact sales or costs. With this knowledge, businesses can develop contingency plans.

7. **Enhanced Customer Understanding**:
    - Analyzing when customers shop, and how often, helps businesses tailor their marketing strategies, sales promotions, and even operational hours.

For your specific case of analyzing Sales Quantity, Revenue, and Freight Cost:

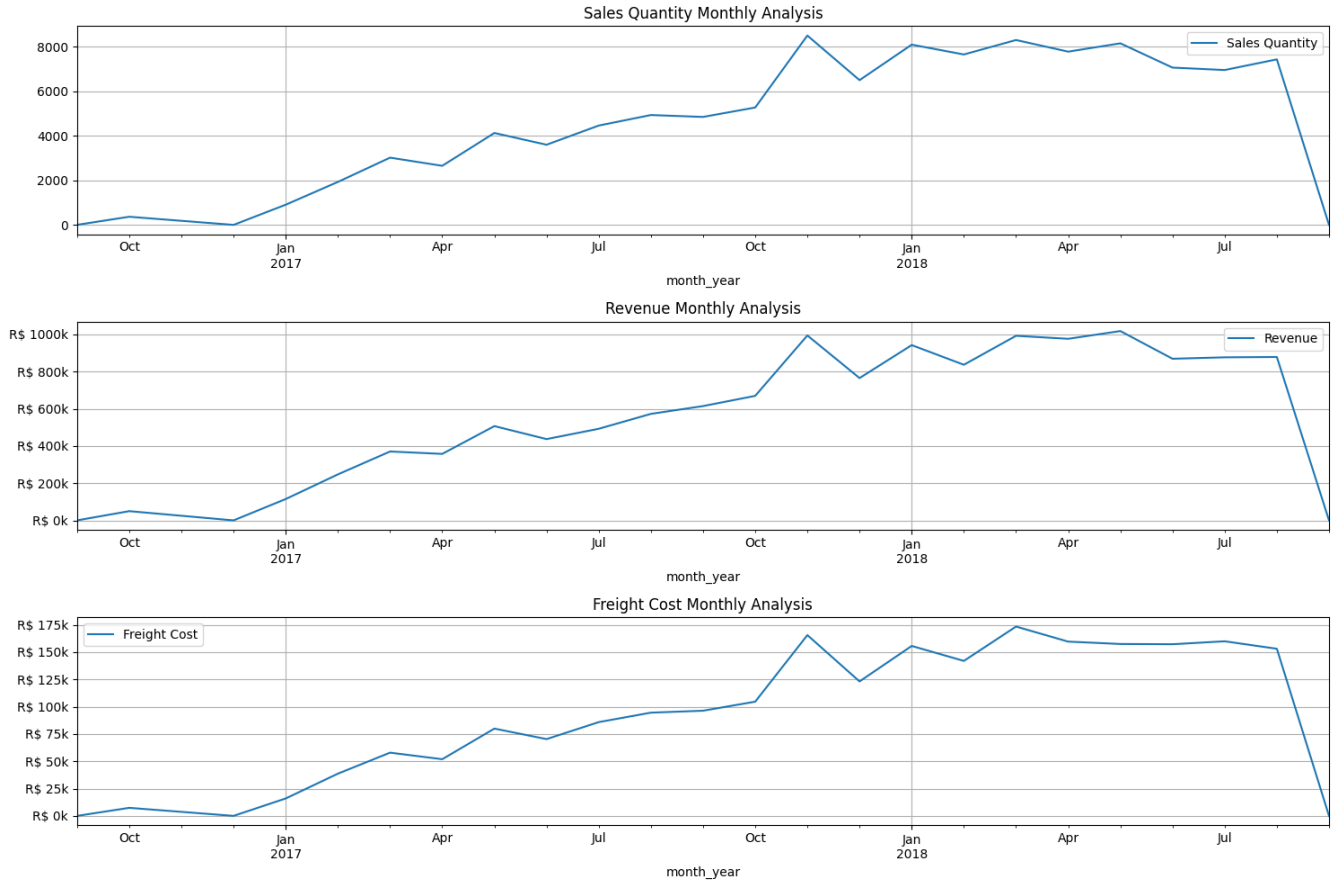
- **Monthly Analysis**: Helps capture short-term variations, which might be due to monthly promotions, events, or external factors like holidays.
- **Annual Analysis**: Provides a bigger picture, capturing year-to-year changes. It's ideal for spotting long-term trends, understanding growth rates, and comparing annual performances.
- **Day Period Analysis**: Reveals intra-day patterns. It's crucial for businesses that experience significant day-to-day variations, like restaurants that want to understand peak meal times.

In summary, temporal trend analysis provides a structured way to analyze time-bound data, uncovering insights that can drive business growth, efficiency, and customer satisfaction.

```
# Load datasets
orders_df = pd.read_csv('/content/olist_orders_dataset.csv')
order_items_df = pd.read_csv('/content/olist_order_items_dataset

# Merge the datasets
merged_df = pd.merge(orders_df, order_items_df, on='order_id', h

# Convert the 'order_approved_at' column to datetime
merged_df['order_approved_at'] = pd.to_datetime(merged_df['order
```

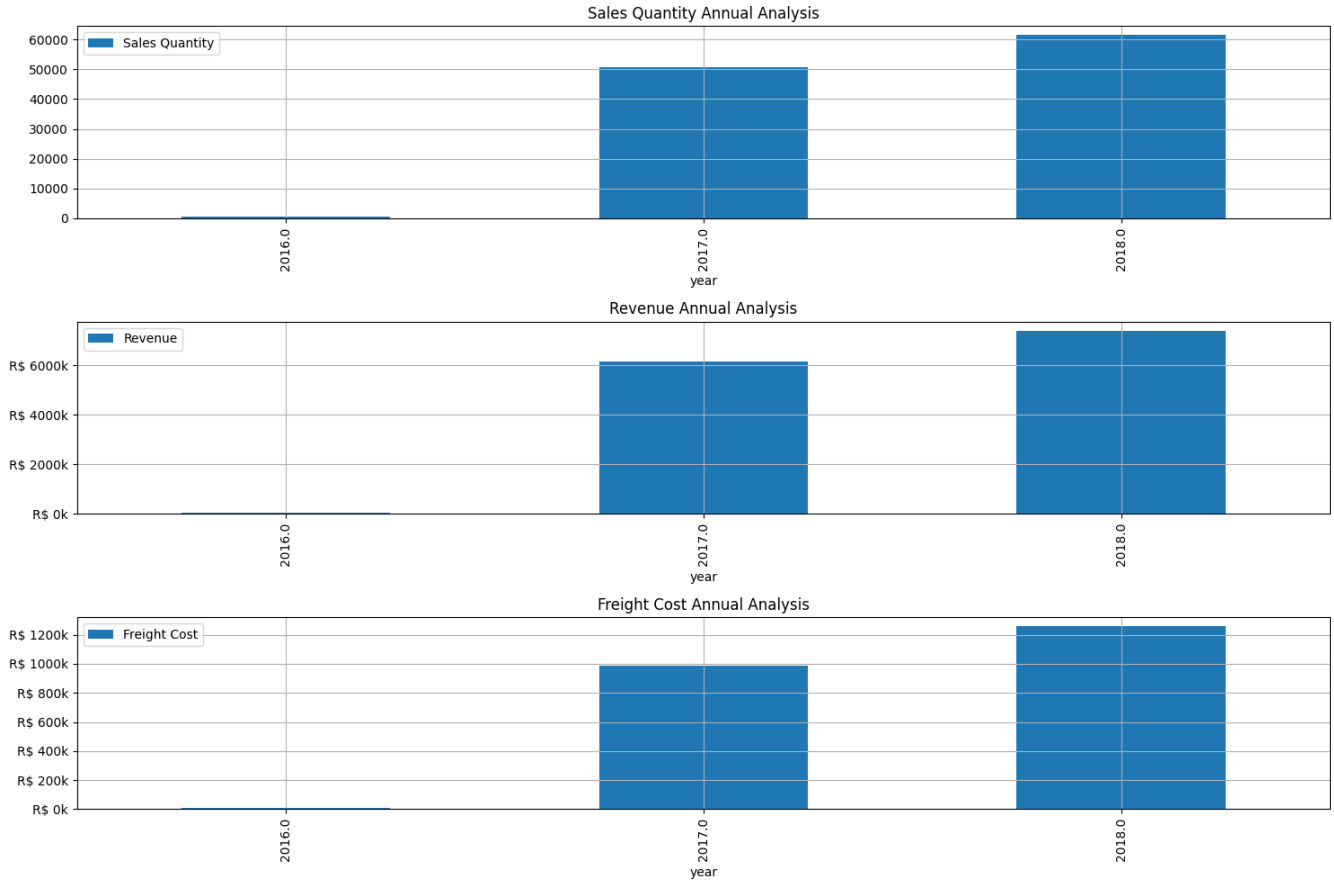
Monthly Analysis

```
# Ensure you have the month_year column
merged_df['month_year'] = merged_df['order_approved_at'].dt.to_p

# Rename columns
merged_df.rename(columns={'order_id': 'Sales Quantity','price':

# Group by month_year to get sales, revenue, and freight cost
monthly_data = merged_df.groupby('month_year').agg({
    'Sales Quantity': 'count',
    'Revenue': 'sum',
    'Freight Cost': 'sum'
}).reset_index()

# Plotting
fig, axes = plt.subplots(3, 1, figsize=(15, 10))
monthly_data.plot(x='month_year', y='Sales Quantity', ax=axes[0]
monthly_data.plot(x='month_year', y='Revenue', ax=axes[1], title
monthly_data.plot(x='month_year', y='Freight Cost', ax=axes[2],

# Adjust the y-axis labels
axes[1].yaxis.set_major_formatter(lambda x, _: f'R$ {x*1e-3:.0f}
axes[2].yaxis.set_major_formatter(lambda x, _: f'R$ {x*1e-3:.0f}

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



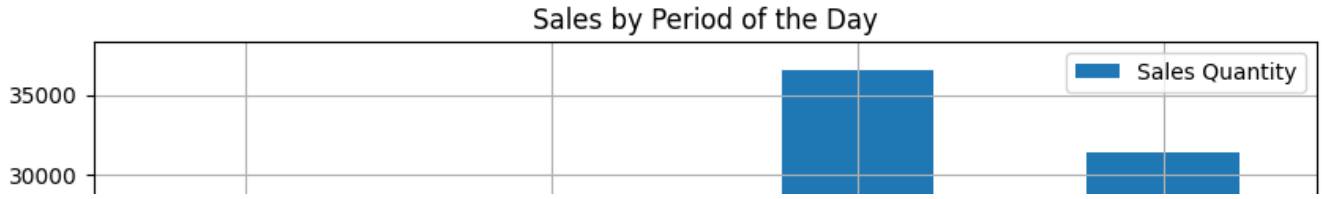
Annual Analysis

```
# Ensure the 'order_approved_at' column is in datetime format
merged_df['order_approved_at'] = pd.to_datetime(merged_df['order

# Create a 'year' column
merged_df['year'] = merged_df['order_approved_at'].dt.year

# Rename columns
merged_df.rename(columns={'order_id': 'Sales Quantity','price':

# Grouping by year
annual_data = merged_df.groupby('year').agg({
    'Sales Quantity': 'count',
    'Revenue': 'sum',
    'Freight Cost': 'sum'
}).reset_index()


# Plotting
fig, axes = plt.subplots(3, 1, figsize=(15, 10))
annual_data.plot(x='year', y='Sales Quantity', kind='bar', ax=ax
annual_data.plot(x='year', y='Revenue', kind='bar', ax=axes[1],
annual_data.plot(x='year', y='Freight Cost', kind='bar', ax=axes

# Adjusting the y-axis labels
axes[1].yaxis.set_major_formatter(lambda x, _: f'R$ {x*1e-3:.0f}
axes[2].yaxis.set_major_formatter(lambda x, _: f'R$ {x*1e-3:.0f}

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


Sales Quantity Annual Analysis
Sales Quantity
60000
50000
40000
30000
20000
10000
0
2016.0
2017.0
2018.0
year
Revenue Annual Analysis
Revenue
R$ 6000k
R$ 4000k
R$ 2000k
R$ 0k
2016.0
2017.0
2018.0
year
Freight Cost Annual Analysis
Freight Cost
R$ 1200k
R$ 1000k
R$ 800k
R$ 600k
R$ 400k
R$ 200k
R$ 0k
2016.0
2017.0
2018.0
year

```
# Determine period of the day
def period_of_day(hour):
    if 0 <= hour < 6:
        return 'Dawn'
    elif 6 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 18:
        return 'Afternoon'
    else:
        return 'Night'

merged_df['period'] = merged_df['order_approved_at'].dt.hour.app

# Group by period of the day
period_data = merged_df.groupby('period').agg({
    'Sales Quantity': 'count'
}).reindex(['Dawn', 'Morning', 'Afternoon', 'Night']).reset_inde

# Plotting
period_data.plot(x='period', y='Sales Quantity', kind='bar', fig
```

```
<Axes: title={'center': 'Sales by Period of the Day'}, xlabel='period'>
```



Customers Analysis

RFM (Recency, Frequency, Monetary) Analysis

The RFM analysis (Recency, Frequency, Monetary) is a pivotal technique in direct marketing and customer relationship management.

It segments customers based on three criteria: the recency of their last purchase, the frequency of their purchases, and the total amount spent. This segmentation allows businesses to pinpoint their most valuable customers and tailor communication strategies accordingly.

By understanding customer purchasing behavior, companies can optimize resources, enhance retention, and maximize the ROI of their campaigns, ensuring more pertinent offers for each segment.

The algorithm bellow loads data representing customers and orders.

After ensuring the 'order_approved_at' column is recognized as a datetime object, it merges the datasets together using order and customer IDs. The merged dataset's 'order_approved_at' column is converted to a datetime format.

To perform the RFM (Recency, Frequency, Monetary) analysis, it identifies the most recent order date, calculates the days since a customer's last purchase (Recency), counts the unique number of orders by a customer (Frequency), and totals the monetary value spent by each customer (Monetary). The results are then consolidated into a single DataFrame.

```
# Load datasets
customers_df = pd.read_csv('/content/olist_customers_dataset.csv
order_items_df = pd.read_csv('/content/olist_order_items_dataset
orders_df = pd.read_csv('/content/olist_orders_dataset.csv')

# Ensure the 'order_approved_at' column is a datetime object
orders_df['order_approved_at'] = pd.to_datetime(orders_df['order

# Merging the datasets
merged_df = pd.merge(orders_df, order_items_df, on='order_id', h
merged_df = pd.merge(merged_df, customers_df, on='customer_id',

# Convert order_approved_at to datetime format
merged_df['order_approved_at'] = pd.to_datetime(merged_df['order

# Get the most recent date in the dataset + 1 day (to use as a r
last_date = merged_df['order_approved_at'].max() + pd.Timedelta(

# Calculate Recency
recency_df = merged_df.groupby('customer_unique_id').agg(last_pu
recency_df['Recency'] = (last_date - recency_df['last_purchase']

# Calculate Frequency
frequency_df = merged_df.groupby('customer_unique_id').agg(Frequ

# Calculate Monetary value
monetary_df = merged_df.groupby('customer_unique_id').agg(Moneta

# Merge all dataframes together
rfm_df = pd.concat([recency_df, frequency_df, monetary_df], axis

# Drop the last_purchase column as it was an intermediate step t
rfm_df.drop(columns='last_purchase', inplace=True)
```

With the combined RFM table:

By displaying the top rows, we get a glimpse of the RFM values for individual customers.

Describe the RFM table provides a statistical summary. This can give insights into averages, distribution, and potential outliers in Recency, Frequency, and Monetary metrics.

```
# Define the 'now' point as one day after the latest order in th
now = merged_df['order_approved_at'].max() + pd.Timedelta(days=1

# Recency: Days since last purchase
recency = merged_df.groupby('customer_unique_id')['order_approve
recency.columns = ['customer_unique_id', 'LastPurchaseDate']
recency['Recency'] = (now - recency['LastPurchaseDate']).dt.days

# Frequency: Number of purchases
frequency = merged_df.groupby('customer_unique_id')['order_id'].
frequency.columns = ['customer_unique_id', 'Frequency']

# Monetary: Total money spent
monetary = merged_df.groupby('customer_unique_id')['price'].sum(
monetary.columns = ['customer_unique_id', 'Monetary']

# Merge recency, frequency and monetary dataframes
rfm = pd.merge(recency, frequency, on='customer_unique_id')
rfm = pd.merge(rfm, monetary, on='customer_unique_id')

# Display the top rows
print(rfm.head())
print(rfm.describe())
```

```
                customer_unique_id    LastPurchaseDate  Recency  Frequency  \
0  0000366f3b9a7992bf8c76cfdf3221e2 2018-05-10 11:11:18    117.0          1
1  0000b849f77a49e4a4ce2b2a4ca5be3f 2018-05-07 18:25:44    119.0          1
2  0000f46a3911fa3c0805444483337064 2017-03-10 21:05:03    542.0          1
3  0000f6ccb0745a6a4b88665a16c9f078 2017-10-12 20:49:17    326.0          1
4  0004aac84e0df4da2b147fca70cf8255 2017-11-14 20:06:52    293.0          1

   Monetary
0    129.90
1     18.90
2     69.00
3     25.99
4    180.00
            Recency     Frequency      Monetary
count  95997.000000  96096.000000  96096.000000
```

mean	244.250904	1.180330	141.438184
std	153.282280	0.620748	217.215904
min	1.000000	1.000000	0.000000
25%	120.000000	1.000000	45.990000
50%	225.000000	1.000000	89.000000
75%	354.000000	1.000000	154.000000
max	719.000000	24.000000	13440.000000

Histograms Visualization for RFM analysis complemented by a Kernel Density Estimate (KDE) overlay.

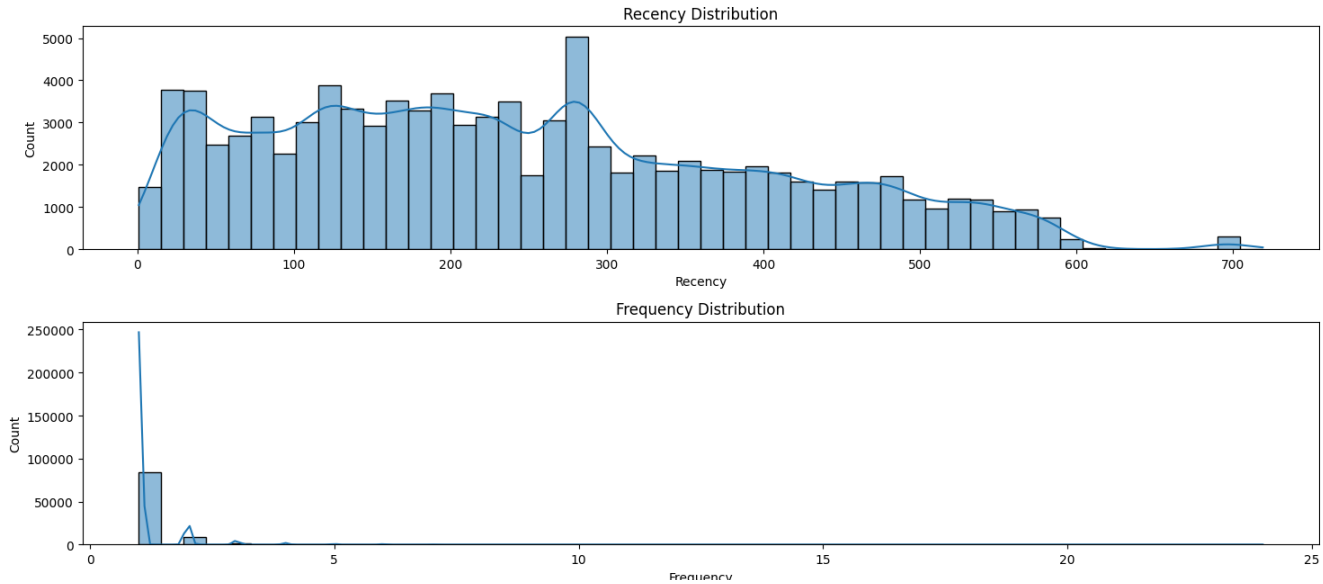
From the plots we can get some insight:

1. **Recency Distribution**:
    - Insights about how recently most of the customers have made purchases can be drawn.
    - Peaks indicate the most common recency intervals.
    - If a significant peak is towards lower values, it indicates that a substantial portion of customers have purchased recently.
2. **Frequency Distribution**:
    - Shows the number of times most customers have made purchases.
    - Peaks highlight the most typical purchase frequencies.
    - If most customers have low frequencies, it might indicate a large pool of one-time purchasers. High frequencies signal a group of loyal customers.
3. **Monetary Distribution**:
    - Represents the total monetary value contributed by customers.
    - Peaks show the most common spending brackets.
    - If the distribution skews towards higher values, it suggests that many customers spend a significant amount.

The KDE lines help in understanding the probability density of the data at different values, making it easier to identify the most common patterns in each metric.

In summary, these visualizations assist businesses in understanding their customer behavior distribution, helping them refine marketing strategies, customer retention programs, and sales forecasts.

```
# Set up the matplotlib figure
f, axes = plt.subplots(3, 1, figsize=(15, 10))

# Plot the distributions
sns.histplot(rfm['Recency'], bins=50, ax=axes[0], kde=True).set_
sns.histplot(rfm['Frequency'], bins=50, ax=axes[1], kde=True).se
sns.histplot(rfm['Monetary'], bins=50, ax=axes[2], kde=True).set

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



Churn Rate

The churn rate, in an e-commerce context, can be determined based on the number of customers who made only one purchase versus the total number of customers. The churn rate calculation can be simplified as:

$$\text{Churn Rate} = \frac{\text{Number of Customers who Purchased Only Once}}{\text{Total Number of Customers}} \times 100\%$$

Let's compute the churn rate using the RFM dataframe that we already have:

```
# Identify customers who made only one purchase
one_time_buyers = rfm[rfm['Frequency'] == 1].shape[0]

# Total customers
total_customers = rfm.shape[0]

# Calculate churn rate
churn_rate = (one_time_buyers / total_customers) * 100

print(f"The Churn Rate is: {churn_rate:.2f}%")
```

```
The Churn Rate is: 87.57%
```

If the churn rate is high, it indicates many customers are making just one purchase and aren't returning for subsequent purchases. This could be a sign that there's a need to invest in customer retention strategies. If the rate is low, it suggests customers are satisfied and are coming back for more purchases, indicating a good business health in terms of customer retention.

Predictive Analysis

Sales Forecast:

Using time series models to forecast future sales.

Predicting sales using time series models is a common practice in data analysis and data science. Here's a basic approach using the ARIMA (Auto Regressive Integrated Moving Average) model.

```
# Preparing the Data
sales_ts = merged_df.groupby('order_approved_at').agg({'price':
```

Checking Stationarity:

Time series models require the data to be stationary. A time series is stationary if its statistical properties do not change over time.

```
def test_stationarity(timeseries):
    result = adfuller(timeseries['price'])
    if result[1] <= 0.05:
        print("The series is stationary")
    else:
        print("The series is not stationary")

test_stationarity(sales_ts)
```

```
The series is not stationary
```

Differencing is one of the most common methods to make a time series stationary. You subtract the current value from the previous value.

```
sales_ts_diff = sales_ts.diff().dropna()
test_stationarity(sales_ts_diff)
```

```
The series is stationary
```