In [ ]:
```
# importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import zscore
import contextily as ctx
```

Data cleaning

In [ ]:
```
#defining the file paths for the datasets
customers = "X:/data/olist_customers_dataset.csv"
geolocation = "X:/data/olist_geolocation_dataset.csv"
order_items = "X:/data/olist_order_items_dataset.csv"
order_payments = "X:/data/olist_order_payments_dataset.csv"
order_reviews = "X:/data/olist_order_reviews_dataset.csv"
orders = "X:/data/olist_orders_dataset.csv"
products = "X:/data/olist_products_dataset.csv"
sellers = "X:/data/olist_sellers_dataset.csv"
product_category_name_translation = "X:/data/product_category_name_translation.c
```

In [ ]:
```
customers_df = pd.read_csv(customers, on_bad_lines='skip')
geolocation_df = pd.read_csv(geolocation, on_bad_lines='skip')
order_items_df = pd.read_csv(order_items, on_bad_lines='skip')
order_payments_df = pd.read_csv(order_payments, on_bad_lines='skip')
order_reviews_df = pd.read_csv(order_reviews, on_bad_lines='skip')
orders_df = pd.read_csv(orders, on_bad_lines='skip')
products_df = pd.read_csv(products, on_bad_lines='skip')
sellers_df = pd.read_csv(sellers, on_bad_lines='skip')
product_category_name_translation_df = pd.read_csv(product_category_name_transla
```

In [ ]:
```
# defining functions to clean and preprocess the data
def checkingforduplinull(df, name="DataFrame"):
    print(f"Checking for duplicates and null values in `{name}`...")
    print(f"Duplicates: {df.duplicated().sum()}")
    print(f"Null values: {df.isnull().sum().sum()}")
    print("\n")
```

In [ ]:
```
dataset = {
    'customers_df': customers_df, 'geolocation_df': geolocation_df, 'order_items
    'order_payments_df': order_payments_df, 'order_reviews_df': order_reviews_df
    'products_df': products_df, 'sellers_df': sellers_df, 'product_category_name
}
```

In [ ]:
```
#checking for duplicates and null values in each DataFrame
checkingforduplinull(customers_df, "customers_df")
checkingforduplinull(geolocation_df, "geolocation_df")
checkingforduplinull(order_items_df, "order_items_df")
checkingforduplinull(order_payments_df, "order_payments_df")
checkingforduplinull(order_reviews_df, "order_reviews_df")
checkingforduplinull(orders_df, "orders_df")
checkingforduplinull(products_df, "products_df")
checkingforduplinull(sellers_df, "sellers_df")
checkingforduplinull(product_category_name_translation_df, "product_category_nam
```

```
Checking for duplicates and null values in `customers_df`...
Duplicates: 0
Null values: 0


Checking for duplicates and null values in `geolocation_df`...
Duplicates: 261831
Null values: 0


Checking for duplicates and null values in `order_items_df`...
Duplicates: 0
Null values: 0


Checking for duplicates and null values in `order_payments_df`...
Duplicates: 0
Null values: 0


Checking for duplicates and null values in `order_reviews_df`...
Duplicates: 0
Null values: 145903


Checking for duplicates and null values in `orders_df`...
Duplicates: 0
Null values: 4908


Checking for duplicates and null values in `products_df`...
Duplicates: 0
Null values: 2448


Checking for duplicates and null values in `sellers_df`...
Duplicates: 0
Null values: 0


Checking for duplicates and null values in `product_category_name_translation_df
`...
Duplicates: 0
Null values: 0
```

duplicates are normal as one place can have multiple orders.

Order_reviews

In [ ]:
```
order_reviews_df.isnull().sum()
```

Out[ ]:
```
review_id                     0
order_id                      0
review_score                  0
review_comment_title      87656
review_comment_message    58247
review_creation_date          0
review_answer_timestamp       0
dtype: int64
```

review_id column

In [ ]:
```
order_reviews_df[order_reviews_df['review_id'].duplicated()].head(3)
```

Out[ ]:

	review_id	order_id	review_s
3317	3242cc306a9218d0377831e175d62fbf	9c5bfba7de6a4abbb6ba0baab78d1622	
5719	308316408775d1600dad81bd3184556d	3fe4dbcdb046a475dbf25463c1ca78bd	
7213	8ee90ac383cf825bb7f4756130d4e74a	75d5d3d16567a27eefc5752aeb063072	

In [ ]:
```
# Remove duplicates based on 'review_id' and keeping the first occurrence
order_reviews_df = order_reviews_df.drop_duplicates(subset='review_id', keep='fi
```

order_id column

In [ ]:
```
order_reviews_df[order_reviews_df['order_id'].duplicated()].head()
```

Out[ ]:

	review_id	order_id	review_s
1119	46abf3ea0b2710ad41390fdb79c32d84	5040757d4e06a4be96d3827b860b4e7c	
3109	aa193e76d35950c4ae988237bb36ed2b	cf73e2cb1f4a9480ed70c154da3d954a	
8108	40294ea5a778dc62080d6b3f55d361ce	e1bc1083cd7acd30d0576335373b907d	
9064	32e2c7e889f7a185d462265398ee3631	c7cfea0c153e6382e32e84c2a9dd7d2e	
9795	95a3135743556b117d888cc8c6e12e11	f9c78e6e58306dc81efbbada1ac11f24	

In [ ]:
```
# Remove duplicates based on 'order_id' and keeping the first occurrence
order_reviews_df = order_reviews_df.drop_duplicates(subset='order_id', keep='fir
```

review_score column

In [ ]:
```
# Find rows where review_score is not between 1 and 5
out_of_range_reviews = order_reviews_df[(order_reviews_df['review_score'] < 1) |
out_of_range_reviews
```

Out[ ]:

review_id	order_id	review_score	review_comment_title	review_comment_message	re

Comments columns

In [ ]:
```
order_reviews_df.isnull().sum() # finding which column null values appear in the
```

Out[ ]:
```
review_id                      0
order_id                       0
review_score                   0
review_comment_title       86654
review_comment_message     57585
review_creation_date           0
review_answer_timestamp        0
dtype: int64
```

In [ ]:
```
order_reviews_df.head()# order_reviews_df dataset
```

Out[ ]:

	review_id	order_id	review_score
0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4
1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5
2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5
3	e64fb393e7b32834bb789ff8bb30750e	658677c97b385a9be170737859d3511b	5
4	f7c4243c7fe1938f181bec41a392bdeb	8e6bfb81e283fa7e4f11123a3fb894f1	5

we will also leave order_reviews_df alone as review comment titles and message are not crucial.

Geolocation_df

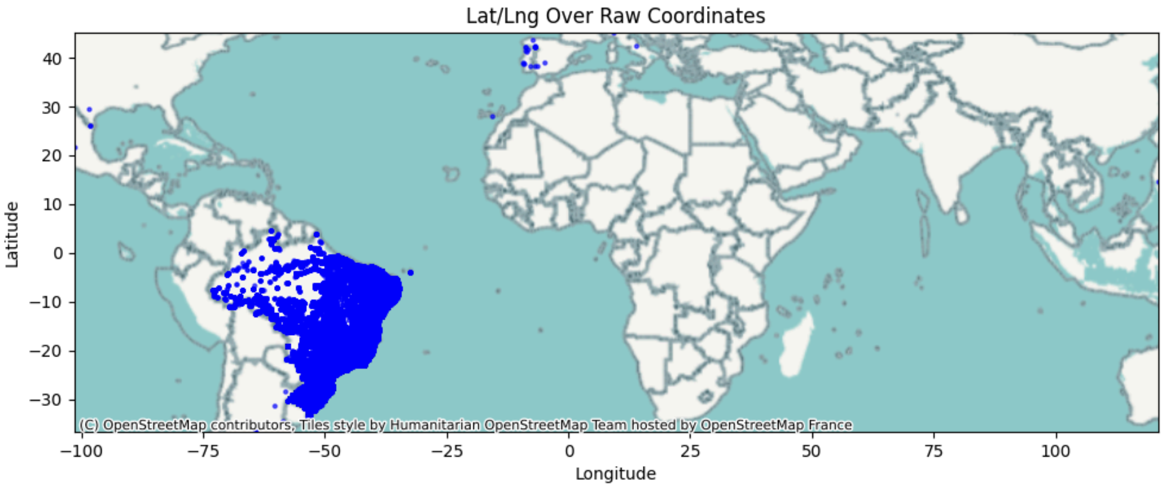
In [ ]:
```
geolocation_df.head()
```

Out[ ]:

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolo
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	

In [ ]:
```
fig, ax = plt.subplots(figsize=(10, 10))# Create a blank plot with lat/lng
ax.scatter(geolocation_df["geolocation_lng"], geolocation_df["geolocation_lat"],
```

```
            s=5, color='blue', alpha=0.6)
ax.set_xlim(geolocation_df["geolocation_lng"].min(), geolocation_df["geolocation
ax.set_ylim(geolocation_df["geolocation_lat"].min(), geolocation_df["geolocation
ax.set_aspect('equal')# Adjust the aspect ratio to be equal so that latitude and
try:# Try to overlay map (may not align perfectly without reprojecting)
    ctx.add_basemap(ax, crs='EPSG:4326')  # using raw lat/lng coords
except Exception as e:
    print("Map overlay failed:", e)
ax.set_title("Lat/Lng Over Raw Coordinates")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.tight_layout()
plt.show()
```



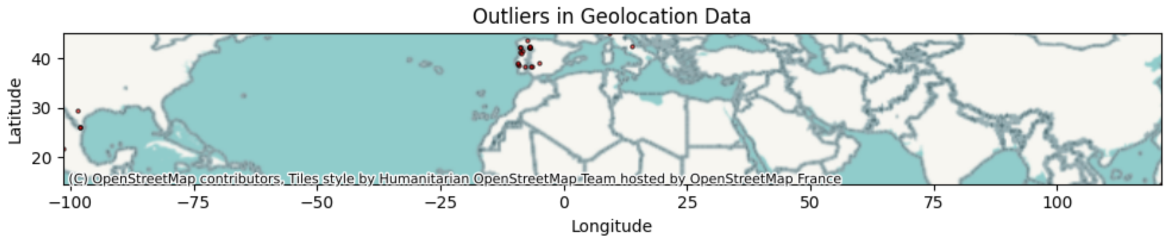
In [ ]:
```
# Calculate z-scores for latitude and longitude
geolocation_df["lat_z"] = zscore(geolocation_df["geolocation_lat"])
geolocation_df["lng_z"] = zscore(geolocation_df["geolocation_lng"])
# Set a threshold (e.g. 3 standard deviations from the mean)
threshold = 10
# Identify rows where either lat or lng z-score is above the threshold
outliers = geolocation_df[(geolocation_df["lat_z"].abs() > threshold) | (geoloca
# Drop the z-score columns if not needed
geolocation_df.drop(columns=["lat_z", "lng_z"], inplace=True)
# Display the outliers
outliers.head(3)
```

Out[ ]:

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city
513631	28165	41.614052	-8.411675	vila nova de campos
513754	28155	42.439286	13.820214	santa maria
514429	28333	38.381672	-6.328200	raposo

In [ ]:
```
# Create a blank plot with lat/lng for outliers
fig, ax = plt.subplots(figsize=(10, 10))
ax.scatter(outliers["geolocation_lng"], outliers["geolocation_lat"],
           s=5, color='red', edgecolors='k', alpha=0.6)
# Set correct bounds for map tiles based on outliers data
ax.set_xlim(outliers["geolocation_lng"].min(), outliers["geolocation_lng"].max()
ax.set_ylim(outliers["geolocation_lat"].min(), outliers["geolocation_lat"].max()
# Adjust the aspect ratio to be equal so that latitude and longitude scales matc
```

```
ax.set_aspect('equal')
# Try to overlay the map for outliers (may not align perfectly without reproject
try:
    ctx.add_basemap(ax, crs='EPSG:4326')  # using raw lat/lng coords
except Exception as e:
    print("Map overlay failed:", e)
# Title and labels
ax.set_title("Outliers in Geolocation Data")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.tight_layout()
plt.show()
```



In [ ]:
```
# removing outliers from the geolocation DataFrame
geolocation_df = geolocation_df.drop(outliers.index)
```

Orders_df

In [ ]:
```
# finding the number of null values in each column of the orders DataFrame
orders_df.isnull().sum()
```

Out[ ]:
```
order_id                            0
customer_id                         0
order_status                        0
order_purchase_timestamp            0
order_approved_at                 160
order_delivered_carrier_date     1783
order_delivered_customer_date    2965
order_estimated_delivery_date       0
dtype: int64
```

order_id column

In [ ]:
```
orders_df[orders_df['order_id'].duplicated()].head()
```

Out[ ]:

order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	or

customer_id column

In [ ]:
```
orders_df[orders_df['customer_id'].duplicated()].head()
```

Out[ ]:

order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	or

Unfilled empty data

In [ ]:
```
# rows that do not have "delivered" in the order_status column
non_delivered = orders_df[orders_df['order_status'] != 'delivered']
non_delivered.head(3)
```

Out[ ]:

	order_id	customer_id	order_statu
6	136cce7faa42fdb2cefd53fdc79a6098	ed0271e0b7da060a393796590e7b737a	invoice
44	ee64d42b8cf066f35eac1cf57de1aa85	caded193e8e47b8362864762a83db3c5	shippe
103	0760a852e4e9d89eb77bf631eaaf1c84	d2a79636084590b7465af8ab374a8cf5	invoice

In [ ]:
```
# unfilled/null rows even with "delivered" status
delivered_with_nulls = orders_df[(orders_df['order_status'] == 'delivered') &(or
delivered_with_nulls.head(3)
```

Out[ ]:

	order_id	customer_id	order_sta
3002	2d1e2d5bf4dc7227b3bfebb81328c15f	ec05a6d8558c6455f0cbbd8a420ad34f	delive
5323	e04abd8149ef81b95221e88f6ed9ab6a	2127dc6603ac33544953ef05ec155771	delive
16567	8a9adc69528e1001fc68dd0aaebbb54a	4c1ccc74e00993733742a3c786dc3c1f	delive

In [ ]:
```
# Dropping unfilled rows even with "delivered" status
orders_df = orders_df.drop(delivered_with_nulls.index)
```

the other null values in "orders_df" are normal due to their respective "order_status"

products_df

In [ ]:
```
# checking for null values in the products DataFrame columns
products_df.isnull().sum()
```

Out[ ]:
```
product_id                      0
product_category_name         610
product_name_lenght           610
product_description_lenght    610
product_photos_qty            610
product_weight_g                2
product_length_cm               2
product_height_cm               2
product_width_cm                2
dtype: int64
```

product_id column

In [ ]:
```
products_df[products_df['product_id'].duplicated()].head()# finding duplicate pr
```

Out[ ]:

product_id	product_category_name	product_name_lenght	product_description_lenght

product_category_name column

In [ ]:
```
empty_product_name = products_df[products_df['product_category_name'].isnull()]#
product_ids_to_remove = empty_product_name['product_id']
product_ids_to_remove.head()
```

Out[ ]:
```
105    a41e356c76fab66334f36de622ecbd3a
128    d8dee61c2034d6d075997acef1870e9b
145    56139431d72cd51f19eb9f7dae4d1617
154    46b48281eb6d663ced748f324108c733
197    5fb61f482620cb672f5e586bb132eae9
Name: product_id, dtype: object
```

In [ ]:
```
def remove_product_ids(df):
    df = df[~df['product_id'].isin(product_ids_to_remove)] # removing the rows w
    return df
```

In [ ]:
```
remove_product_ids(products_df) # removing rows with null values in 'product_cat
products_df.head(3)
```

Out[ ]:

	product_id	product_category_name	product_name_lenght	p
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumaria	40.0	
1	3aa071139cb16b67ca9e5dea641aaa2f	artes	44.0	
2	96bd76ec8810374ed1b65e291975717f	esporte_lazer	46.0	

outliers, (missing last 4 columns)

In [ ]:
```
# finding the 2 outliers, 2 null values in product_weight_g, product_length_cm,
empty = products_df[products_df['product_weight_g'].isnull()]
empty.head()
```

Out[ ]:

	product_id	product_category_name	product_name_lengh
8578	09ff539a621711667c43eba6a3bd8466	bebes	60.(
18851	5eb564652db742ff8f28759cd8d2652a	NaN	NaN

In [ ]:
```
# Add a new product_id to the product_ids_to_remove series using concat
product_ids_to_remove = pd.concat([product_ids_to_remove, pd.Series(['09ff539a62
```

In [ ]:
```
# removing the row with null values in product_weight_g, product_length_cm, prod
remove_product_ids(products_df)
products_df.head(3)
```

Out[ ]:

	product_id	product_category_name	product_name_lenght	p
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumaria	40.0	
1	3aa071139cb16b67ca9e5dea641aaa2f	artes	44.0	
2	96bd76ec8810374ed1b65e291975717f	esporte_lazer	46.0	

In [ ]:
```
#removing the row with product_id = 5eb564652db742ff8f28759cd8d2652a
products_df = remove_product_ids(products_df)
order_items_df = remove_product_ids(order_items_df)
```

customers_df

In [ ]:
```
# checking for null values in the customers_df DataFrame columns
customers_df.isnull().sum()
```

Out[ ]:
```
customer_id                 0
customer_unique_id          0
customer_zip_code_prefix    0
customer_city               0
customer_state              0
dtype: int64
```

customer_id column

In [ ]:
```
customers_df[customers_df['customer_id'].duplicated()] #checking for duplicate c
```

Out[ ]:

customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_s

customer_unique_id

In [ ]:
```
customers_df[customers_df['customer_unique_id'].duplicated()].head(3) # checking
```

Out[ ]:

	customer_id	customer_unique_id	custome
679	c57b4b6f3719475543b721e720a526ad	b6c083700ca8c135ba9f0f132930d4e8	
1463	9f6f3da49e2d46e3a7529f5e3c25ecce	a40096fc0a3862e9e12bc55b5f8e6ab2	
1607	299f7b5125c8fbe1761a1b320c34fc7d	b8b3c435a58aebd788a477bed8342910	

Having duplicates on this "customer_unique_id" is normal as this mean that one customer has repeatedly shopped at Olist

order_items_df

In [ ]:
```
# checking for null values in the order_items DataFrame columns
order_items_df.isnull().sum()
```

Out[ ]:
```
order_id               0
order_item_id          0
product_id             0
seller_id              0
shipping_limit_date    0
price                  0
freight_value          0
dtype: int64
```

order_id, order_item_id, product_id column

In [ ]:
```
# checking for duplicate order_id in order_items_df
order_items_df[order_items_df['order_id'].duplicated()].head(3)
```

Out[ ]:

	order_id	order_item_id	product_i
14	0008288aa423d2a3f00fcb17cd7d8719	2	368c6c730842d78016ad823897a372c
33	00143d0f86d6fbd9f9b38ab440ac16f5	2	e95ee6822b66ac6058e2e4aff656071
34	00143d0f86d6fbd9f9b38ab440ac16f5	3	e95ee6822b66ac6058e2e4aff656071

In [ ]:
```
# Group by 'order_id' and 'product_id' and get the row with the highest 'order_i
order_items_df = order_items_df.loc[order_items_df.groupby(['order_id', 'product
# Display the cleaned DataFrame
order_items_df.head(3)
```

Out[ ]:

	order_id	order_item_id	product_ic
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fc

In [ ]:
```
# Rename 'order_item_id' to 'quantity'
order_items_df = order_items_df.rename(columns={'order_item_id': 'quantity'})
order_items_df.head(3)
```

Out[ ]:

	order_id	quantity	product_id	
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b

order_payments_df

In [ ]:
```
# checking for null values in the order_payments DataFrame columns
order_payments_df.isnull().sum()
```

Out[ ]:
```
order_id                0
payment_sequential      0
payment_type            0
payment_installments    0
payment_value           0
dtype: int64
```

In [ ]:
```
# checking for duplicate order_id in order_payments_df
order_payments_df[order_payments_df['order_id'].duplicated()].head(3)
```

Out[ ]:

	order_id	payment_sequential	payment_type	payment
1456	683bf306149bb869980b68d48a1bd6ab	1	credit_card	
2324	e6a66a8350bb88497954d37688ab123e	2	voucher	
2393	8e5148bee82a7e42c5f9ba76161dc51a	1	credit_card	

Revamping the dataset to fit for our use.

In [ ]:
```
# Add a new column to identify if the payment_type is 'voucher'
order_payments_df['is_voucher'] = order_payments_df['payment_type'] == 'voucher'
# Count the number of vouchers for each 'order_id'
voucher_counts = order_payments_df[order_payments_df['is_voucher']].groupby('ord
# Perform aggregation and keep 'payment_type' as well
orderpaymentmerge = order_payments_df.groupby('order_id').agg({
    'payment_value': 'sum',         # Total payment for the order
    'is_voucher': 'any',            # Whether any voucher was used in the order
    'payment_type': 'first'         # Keep the first 'payment_type' for each 'or
}).reset_index()
# Rename columns
orderpaymentmerge.rename(columns={'payment_value': 'total_payment',  'is_voucher
# Merge with voucher counts to get the number of vouchers used per order
orderpaymentmerge = pd.merge(orderpaymentmerge, voucher_counts, on='order_id', h
# Fill missing 'voucher_count' values with 0 and convert to integer
orderpaymentmerge['voucher_count'] = orderpaymentmerge['voucher_count'].fillna(0
# Display the final result
order_payments_df = orderpaymentmerge
order_payments_df.head(3)
```

Out[ ]:

	order_id	total_payment	voucher_used	payment_type	vou
0	00010242fe8c5a6d1ba2dd792cb16214	72.19	False	credit_card	
1	00018f77f2f0320c557190d7a144bdd3	259.83	False	credit_card	
2	000229ec398224ef6ca0657da4fc703e	216.87	False	credit_card	

product_category_name_translation_df

In [ ]:
```
# checking for null values in the product_category_name_translation_df columns
product_category_name_translation_df.isnull().sum()
```

Out[ ]:
```
product_category_name            0
product_category_name_english    0
dtype: int64
```

In [ ]:
```
#checking for duplicate product_category_name in product_category_name_translati
product_category_name_translation_df[product_category_name_translation_df['produ
```

Out[ ]:

product_category_name	product_category_name_english

In [ ]:
```
# checking for duplicate product_category_name_english in product_category_name_
product_category_name_translation_df[product_category_name_translation_df['produ
```

Out[ ]:

product_category_name	product_category_name_english

Merging the Datasets:

In [ ]:
```
#Merge Customer_df with Orders_df
customer_orders_df = pd.merge(customers_df, orders_df, on='customer_id', how="in
customer_orders_df.head(3)
```

Out[ ]:

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	

In [ ]:
```
#merge customer, orders df with payment df
customer_orders_payment_df = pd.merge(customer_orders_df, order_payments_df, on=
customer_orders_payment_df.head(3)
```

Out[ ]:

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	

In [ ]:
```
#Ensure that geolocation zipcodes are in customer dataset
filtered_customer_orders_payment_df = customer_orders_payment_df[customer_orders
```

```
filtered_customer_orders_payment_df.head(3)
```

Out[ ]:

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	

In [ ]:

```
filtered_customer_orders_payment_reviews_df = pd.merge(filtered_customer_orders_
filtered_customer_orders_payment_reviews_df.head(3)
#merge the geolocation-filtered customer + order + payment dataset with reviews
```

Out[ ]:

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	

3 rows × 22 columns

In [ ]:

```
# Before we merge the rest, merge Product cateogry name translated with Product_
products_df = products_df.merge(
    product_category_name_translation_df,
    on='product_category_name',
    how='left'
)
# Replace the original column with the English version
products_df['product_category_name'] = products_df['product_category_name_englis
# Drop the now redundant English translation column
products_df.drop(columns=['product_category_name_english'], inplace=True)
products_df.head(3)
```

Out[ ]:

	product_id	product_category_name	product_name_lenght	p
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumery	40.0	
1	3aa071139cb16b67ca9e5dea641aaa2f	art	44.0	
2	96bd76ec8810374ed1b65e291975717f	sports_leisure	46.0	

In [ ]:

```
products_df = products_df[['product_id', 'product_category_name']]
products_df.head(3)
#the only columns we need.
```

Out[ ]:

	product_id	product_category_name
0	1e9e8ef04dbcff4541ed26657ea517e5	perfumery
1	3aa071139cb16b67ca9e5dea641aaa2f	art
2	96bd76ec8810374ed1b65e291975717f	sports_leisure

In [ ]:

```
products_order_items_df = order_items_df.merge(products_df, on='product_id', how
products_order_items_df.head(3)
#merge order items with product id-get product category for order items
```

Out[ ]:

	order_id	quantity	product_id	
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61	48
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8f	dd
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fd	5b

In [ ]:

```
# I want to merge the orders together, though they include different products, s
products_order_items_df_grouped_Version1 = products_order_items_df.groupby('orde
    'product_category_name': lambda x: ', '.join(sorted(set(x.dropna()))), 'pric
    'freight_value': 'sum', 'quantity': 'sum'})
products_order_items_df_grouped_Version1.head(3)
```

Out[ ]:

	order_id	product_category_name	price	freight_value	qua
0	00010242fe8c5a6d1ba2dd792cb16214	cool_stuff	58.9	13.29	
1	00018f77f2f0320c557190d7a144bdd3	pet_shop	239.9	19.93	
2	000229ec398224ef6ca0657da4fc703e	furniture_decor	199.0	17.87	

In [ ]:

```
merged_dataset = filtered_customer_orders_payment_reviews_df.merge(products_orde
merged_dataset.head(3)
```

Out[ ]:

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	

3 rows × 26 columns

In [ ]:

```
#exporting the final merged dataset to a CSV file
merged_dataset.to_csv('merged_dataset.csv', index=False)
```

Feature

In [ ]:
```
df = pd.read_csv('merged_dataset.csv')
df.head(3)
```

Out[ ]:

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	

3 rows × 26 columns

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Convert datetime columns to pandas datetime format
df['order_purchase_timestamp'] = pd.to_datetime(df['order_purchase_timestamp'])
df['order_approved_at'] = pd.to_datetime(df['order_approved_at'])
df['order_delivered_carrier_date'] = pd.to_datetime(df['order_delivered_carrier_
df['review_creation_date'] = pd.to_datetime(df['review_creation_date'])

# Extract useful time features from the datetime columns
df['order_purchase_hour'] = df['order_purchase_timestamp'].dt.hour
df['order_purchase_day'] = df['order_purchase_timestamp'].dt.day
df['order_purchase_weekday'] = df['order_purchase_timestamp'].dt.weekday
df['order_purchase_month'] = df['order_purchase_timestamp'].dt.month

df['order_to_approval_time'] = (df['order_approved_at'] - df['order_purchase_tim
df['approval_to_delivery_time'] = (df['order_delivered_carrier_date'] - df['orde
df['delivery_to_review_time'] = (df['review_creation_date'] - df['order_delivere
```

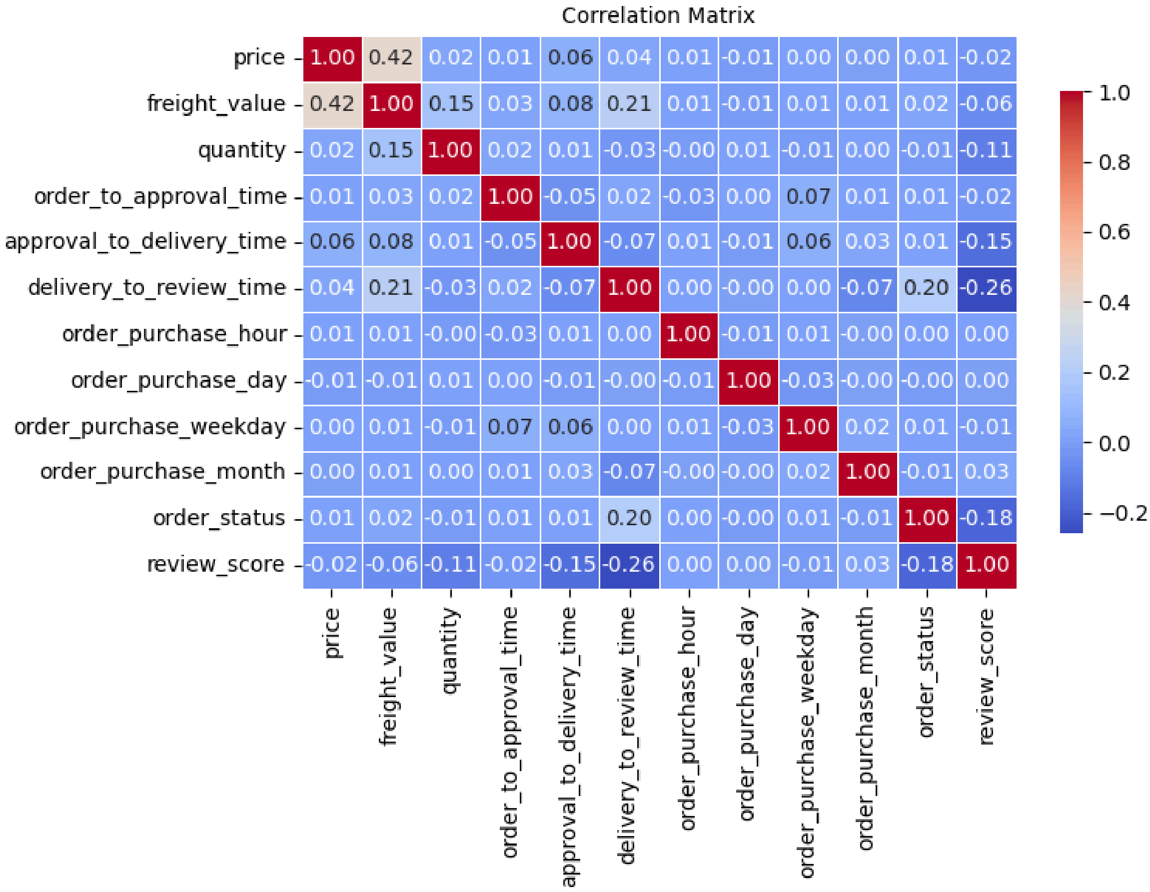
In [ ]:
```
# Label Encoding for Ordinal Categories (e.g., 'order_status', 'review_score')
label_encoder = LabelEncoder()
df['order_status'] = label_encoder.fit_transform(df['order_status'])  # Ordinal
df['review_score'] = label_encoder.fit_transform(df['review_score'])  # Ordinal

# One-Hot Encoding for Nominal Categories (e.g., 'product_category_name', 'custo
df = pd.get_dummies(df, columns=['product_category_name', 'customer_state'], drc

# Step 1: Select relevant numeric columns for correlation analysis
df1 = df[['price', 'freight_value', 'quantity', 'order_to_approval_time',
          'approval_to_delivery_time', 'delivery_to_review_time', 'order_purchas
          'order_purchase_day', 'order_purchase_weekday', 'order_purchase_month'
          'order_status', 'review_score']]
```

In [ ]:
```
# Step 2: Compute the correlation matrix for the selected numeric columns
correlation_matrix = df1.corr()
```

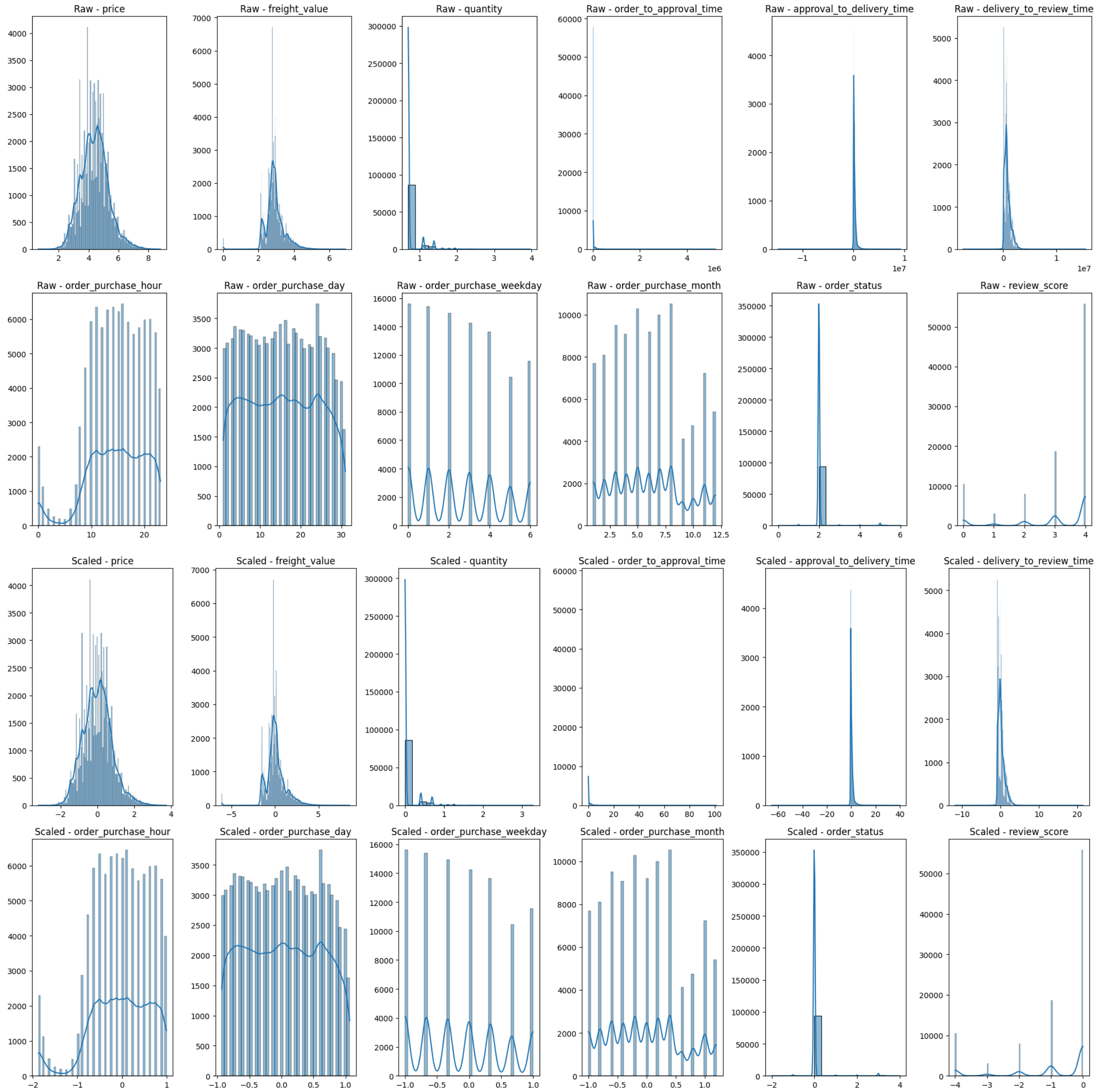
```
# Step 3: Plot the correlation matrix with better readability
plt.figure(figsize=(8, 6))  # Reduce size for readability
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidt
plt.title('Correlation Matrix', fontsize=10)
plt.tight_layout()  # Ensure the plot fits within the figure
plt.show()
```

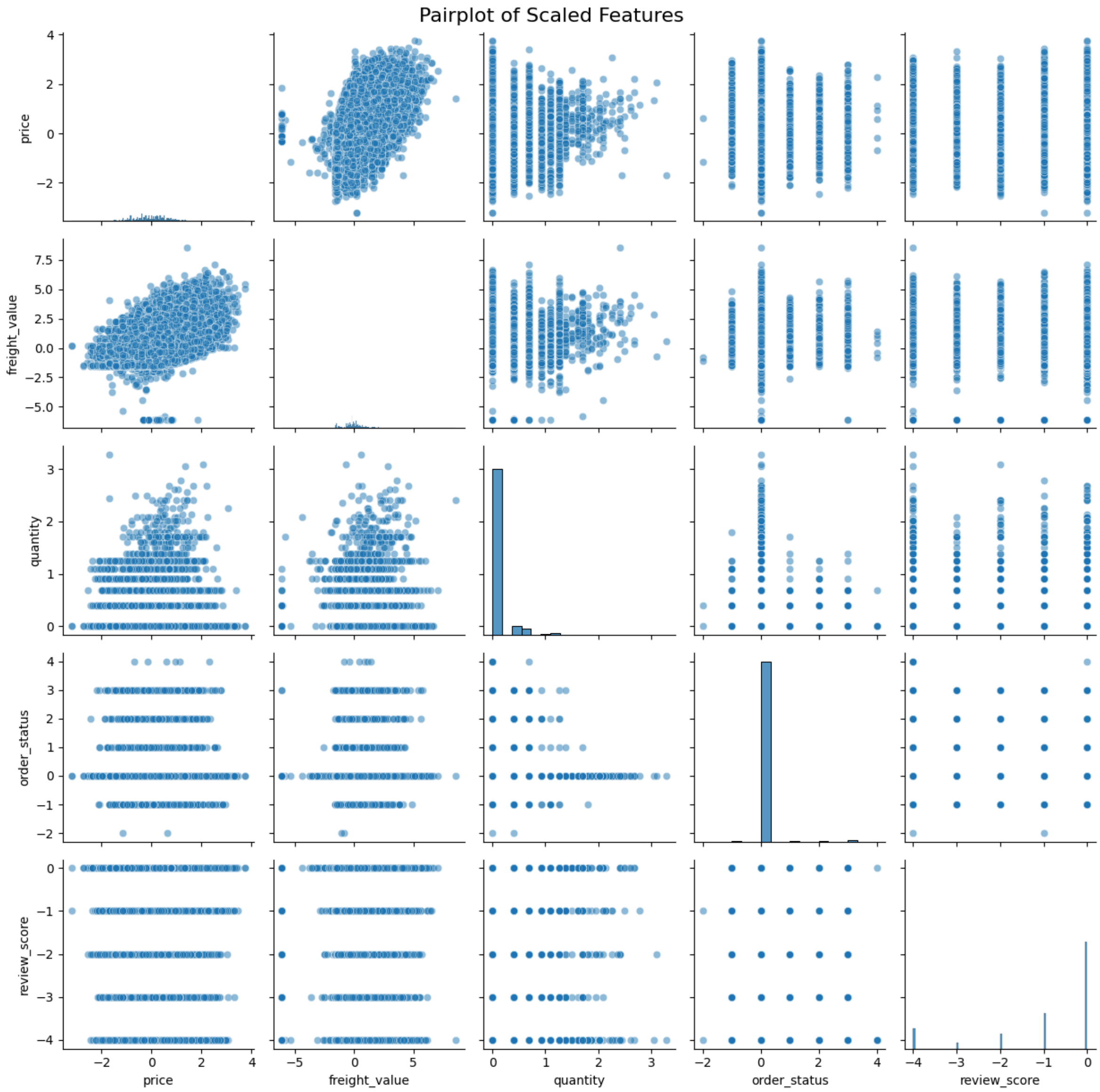


In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.impute import SimpleImputer
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
# Load the cleaned CSV file
df = pd.read_csv('merged_dataset.csv')
# Step 1: Convert datetime columns to pandas datetime format
df['order_purchase_timestamp'] = pd.to_datetime(df['order_purchase_timestamp'])
df['order_approved_at'] = pd.to_datetime(df['order_approved_at'])
df['order_delivered_carrier_date'] = pd.to_datetime(df['order_delivered_carrier_
df['review_creation_date'] = pd.to_datetime(df['review_creation_date'])
# Step 2: Extract useful time features from the datetime columns
df['order_purchase_hour'] = df['order_purchase_timestamp'].dt.hour
df['order_purchase_day'] = df['order_purchase_timestamp'].dt.day
df['order_purchase_weekday'] = df['order_purchase_timestamp'].dt.weekday
df['order_purchase_month'] = df['order_purchase_timestamp'].dt.month
df['order_to_approval_time'] = (df['order_approved_at'] - df['order_purchase_tim
df['approval_to_delivery_time'] = (df['order_delivered_carrier_date'] - df['orde
df['delivery_to_review_time'] = (df['review_creation_date'] - df['order_delivere
# Step 3: Label Encoding for Ordinal Categories (e.g., 'order_status', 'review_s
label_encoder = LabelEncoder()
```

```
df['order_status'] = label_encoder.fit_transform(df['order_status'])  # Ordinal
df['review_score'] = label_encoder.fit_transform(df['review_score'])  # Ordinal
# Step 4: One-Hot Encoding for Nominal Categories (e.g., 'product_category_name'
df = pd.get_dummies(df, columns=['product_category_name', 'customer_state'], drc
# Step 5: Select relevant numeric columns for analysis
df1 = df[['price', 'freight_value', 'quantity', 'order_to_approval_time',
          'approval_to_delivery_time', 'delivery_to_review_time', 'order_purchas
          'order_purchase_day', 'order_purchase_weekday', 'order_purchase_month'
          'order_status', 'review_score']]
# Step 6: Handle missing values
imputer = SimpleImputer(strategy='mean')  # Fill missing values with the column
df1 = pd.DataFrame(imputer.fit_transform(df1), columns=df1.columns)
# Step 7: Apply Log Transformation to Skewed Features (price, freight_value, qua
# We use np.log1p to safely handle zero or negative values by applying log(1+x)
df1['price'] = np.log1p(df1['price'])
df1['freight_value'] = np.log1p(df1['freight_value'])
df1['quantity'] = np.log1p(df1['quantity'])
# Step 8: Apply RobustScaler to handle outliers
scaler = RobustScaler()
df1_scaled = scaler.fit_transform(df1)
# Convert the scaled data back to a DataFrame
df1_scaled = pd.DataFrame(df1_scaled, columns=df1.columns)
# Step 9: Plot the distribution of raw and scaled features (selected subset for
fig, axes = plt.subplots(2, 6, figsize=(20, 10))  # Increase the figure size
axes = axes.flatten()
# Plot raw (before scaling) distributions
for i, column in enumerate(df1.columns):
    sns.histplot(df1[column], kde=True, ax=axes[i])
    axes[i].set_title(f'Raw - {column}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
# Adjust layout to fit the plots
plt.tight_layout()
plt.show()
# Step 10: Plot scaled (after scaling) distributions
fig, axes = plt.subplots(2, 6, figsize=(20, 10))  # Increase the figure size
axes = axes.flatten()
for i, column in enumerate(df1.columns):
    sns.histplot(df1_scaled[column], kde=True, ax=axes[i])
    axes[i].set_title(f'Scaled - {column}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
plt.tight_layout()
plt.show()
# Step 11: Scatter Plot Matrix (Pairplot) to visualize relationships between sel
# We'll reduce the variables for clarity
sns.pairplot(df1_scaled[['price', 'freight_value', 'quantity', 'order_status', '
plt.suptitle('Pairplot of Scaled Features', size=16)
plt.tight_layout()
plt.show()
```


Raw - price
Raw - freight_value
Raw - quantity
Raw - order_to_approval_time
Raw - approval_to_delivery_time
Raw - delivery_to_review_time
Raw - order_purchase_hour
Raw - order_purchase_day
Raw - order_purchase_weekday
Raw - order_purchase_month
Raw - order_status
Raw - review_score
Scaled - price
Scaled - freight_value
Scaled - quantity
Scaled - order_to_approval_time
Scaled - approval_to_delivery_time
Scaled - delivery_to_review_time
Scaled - order_purchase_hour
Scaled - order_purchase_day
Scaled - order_purchase_weekday
Scaled - order_purchase_month
Scaled - order_status
Scaled - review_score



In [ ]:
```
df1_scaled.head(5)
```

Out[ ]:

	price	freight_value	quantity	order_to_approval_time	approval_to_delivery_time
0	0.375897	0.546860	0.0	-0.004693	1.875785
1	1.083313	2.095298	0.0	-0.012236	0.381013
2	0.471046	0.129150	0.0	1.664650	7.574440
3	0.529213	0.679803	0.0	0.072174	4.679188
4	0.890300	0.580885	0.0	-0.002327	-0.234406