Libraries

In [347…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import tqdm
from unidecode import unidecode
from haversine import haversine, Unit
```

In [348…

```
import sys
import os

# File path to the src directory for both linux and windows
# workaround for the issue of relative imports in Jupyter notebooks to import mo
src_path = os.path.abspath("src")
if src_path not in sys.path:
    sys.path.insert(0, src_path)
```

In [349…

```
# Rerun this cell after making changes to the utils module
from the_team.utils import etl, viz
import importlib
importlib.reload(etl)
importlib.reload(viz)

# Set custom plot style for consistency
viz.set_plot_style()
```

```
INFO - Custom plot style set.
```

Before Cleaning

In [350…

```
RAW_DIR = Path("data") / "01_raw"
```

In [351…

```
# Load datasets
customers = etl.load_csv(RAW_DIR / "olist_customers_dataset.csv")
geolocation = etl.load_csv(RAW_DIR / "olist_geolocation_dataset.csv")
items = etl.load_csv(RAW_DIR / "olist_order_items_dataset.csv")
payments = etl.load_csv(RAW_DIR / "olist_order_payments_dataset.csv")
reviews = etl.load_csv(RAW_DIR / "olist_order_reviews_dataset.csv")
orders = etl.load_csv(RAW_DIR / "olist_orders_dataset.csv")
products = etl.load_csv(RAW_DIR / "olist_products_dataset.csv")
sellers = etl.load_csv(RAW_DIR / "olist_sellers_dataset.csv")
translation = etl.load_csv(RAW_DIR / "product_category_name_translation.csv")
```

Geolocation-related datasets [Jin Bin]

- customers.csv
- geolocation.csv
- sellers.csv

customers.csv

In [352...

```
customers.head()
```

Out[352...

	customer_id	customer_unique_id	customer_zi
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	

In [353...

```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 5 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   customer_id               99441 non-null  object
 1   customer_unique_id        99441 non-null  object
 2   customer_zip_code_prefix  99441 non-null  int64 
 3   customer_city             99441 non-null  object
 4   customer_state            99441 non-null  object
dtypes: int64(1), object(4)
memory usage: 3.8+ MB
```

In [354...

```
# Check for duplicates
etl.null_duplicate_check(customers)
```

```
INFO - No null values found.
INFO - No duplicates found.
```

In [355...

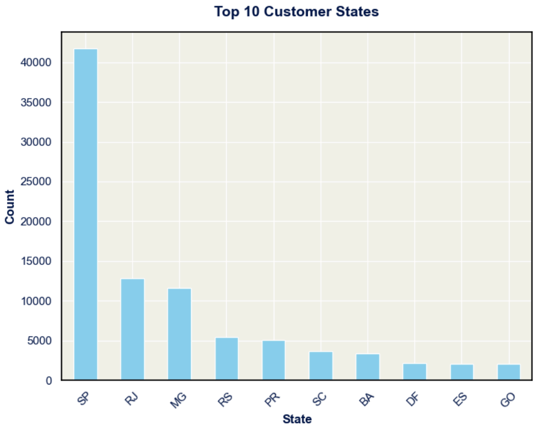
```
# Data formatting
formatted_customers = etl.format_customers(customers)

# Min seems low but its not a problem
formatted_customers.describe()
```

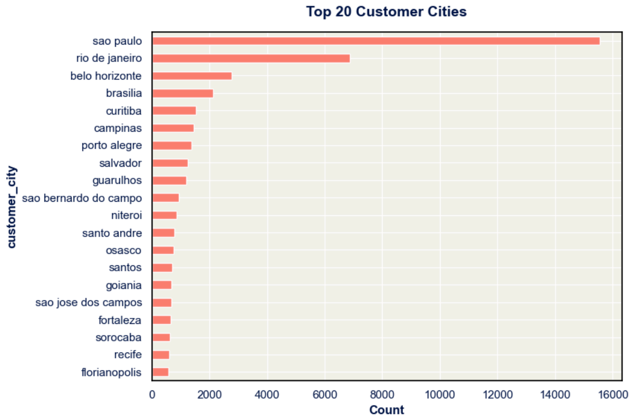
Out[355...

	customer_id	customer_unique_id	customer
count	99441	99441	
unique	99441	96096	
top	274fa6071e5e17fe303b9748641082c8	8d50f5eadf50201ccdcedfb9e2ac8455	
freq	1	17	

In [356...

```
viz.plot_top_locations(formatted_customers, title_prefix="Customer")
```





1. Strong Regional Concentration.(SP) dominates customer count with over 40,000 customers — nearly half of the data. This suggests regional market dependence so if Olist wants to target repeat buyers, SP should be a priority.
2. Urban Centers Drive Volume. Cities like São Paulo, Rio de Janeiro, and Belo Horizonte are far ahead of others. (Urban hubs = higher density = possibly faster repeat behavior.) --> we could analyze if urban customers reorder more frequently due to better delivery coverage or seller availability after taking sellers.csv into account.

geolocation.csv

In [357…

```
geolocation.head()
```

Out[357…

	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	

The geolocation dataset contains multiple similar latitude and longitude entries for the same zip code prefix. To simplify the data and enable efficient merging with customer and seller datasets, we averaged the latitude and longitude for each unique zip code prefix. While this reduces geographic precision, it preserves regional location context needed for distance-based analysis in later stages.

In [358…

```
geolocation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000163 entries, 0 to 1000162
Data columns (total 5 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   geolocation_zip_code_prefix  1000163 non-null  int64
 1   geolocation_lat              1000163 non-null  float64
 2   geolocation_lng              1000163 non-null  float64
 3   geolocation_city             1000163 non-null  object
 4   geolocation_state            1000163 non-null  object
dtypes: float64(2), int64(1), object(2)
memory usage: 38.2+ MB
```

In [359…

```
#checks for duplicates
etl.null_duplicate_check(geolocation, verbose=False)
```

```
INFO - No null values found.
WARNING - Duplicates found.
INFO - 26.18% or 261831 rows are complete duplicates.
```

In [360…

```
geolocation.describe()
```

Out[360…

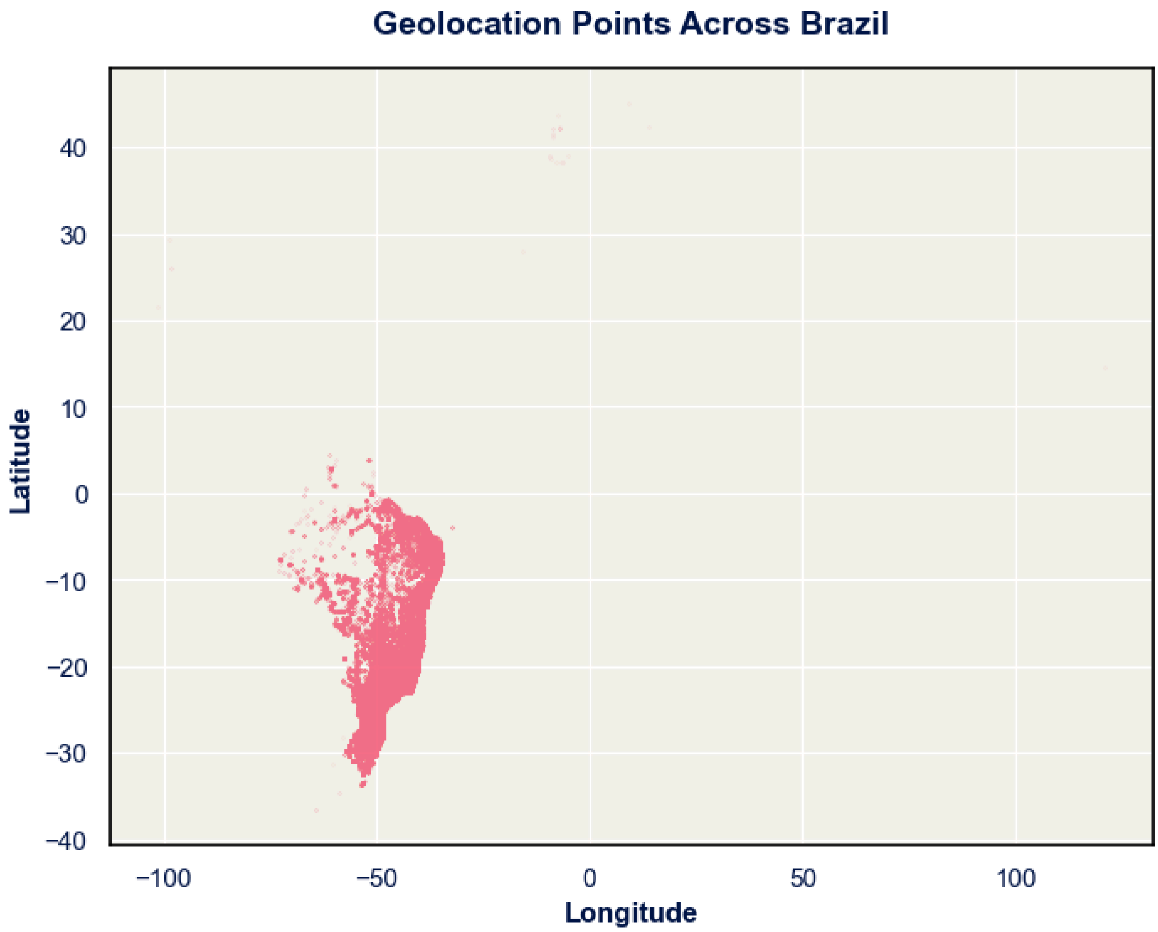
	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng
count	1.000163e+06	1.000163e+06	1.000163e+06
mean	3.657417e+04	-2.117615e+01	-4.639054e+01
std	3.054934e+04	5.715866e+00	4.269748e+00
min	1.001000e+03	-3.660537e+01	-1.014668e+02
25%	1.107500e+04	-2.360355e+01	-4.857317e+01
50%	2.653000e+04	-2.291938e+01	-4.663788e+01
75%	6.350400e+04	-1.997962e+01	-4.376771e+01
max	9.999000e+04	4.506593e+01	1.211054e+02

max lat/lng values are a little suspicious cause the borders of brazil are not that big

- lat range should be between 33.75116944 and 5.27438888
- lng range should be between -73.98283055 and -34.79314722

In [361…

```
viz.plot_geolocation_scatter(geolocation)
```



In [362…

```
# Removing the outlier Lat/Lng values
# Filter only valid rows
valid_lat_range = (-33.75116944, 5.27438888)
valid_lng_range = (-73.98283055, -34.79314722)

geolocation = geolocation[
    (geolocation["geolocation_lat"].between(*valid_lat_range)) &
    (geolocation["geolocation_lng"].between(*valid_lng_range))
]

# Check if any outliers remain
print(f"Remaining rows: {len(geolocation)}")
geolocation.describe()
```

```
Remaining rows: 1000121
```

Out[362...

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng
count	1.000121e+06	1.000121e+06	1.000121e+06
mean	3.657332e+04	-2.117779e+01	-4.639142e+01
std	3.054939e+04	5.707716e+00	4.260789e+00
min	1.001000e+03	-3.369262e+01	-7.293075e+01
25%	1.107500e+04	-2.360355e+01	-4.857322e+01
50%	2.653000e+04	-2.291941e+01	-4.663789e+01
75%	6.350400e+04	-1.997985e+01	-4.376810e+01
max	9.999000e+04	4.482242e+00	-3.479369e+01

In [363...

```
formatted_geolocation = etl.format_geolocation(geolocation)
formatted_geolocation.head()
```

Out[363...

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng
0	01001	-23.550271	-46.634047
1	01002	-23.547657	-46.634991
2	01003	-23.548991	-46.635653
3	01004	-23.549829	-46.634792
4	01005	-23.549487	-46.636650

In [364...

```
geolocation.drop_duplicates(inplace=True)
```

In [365...

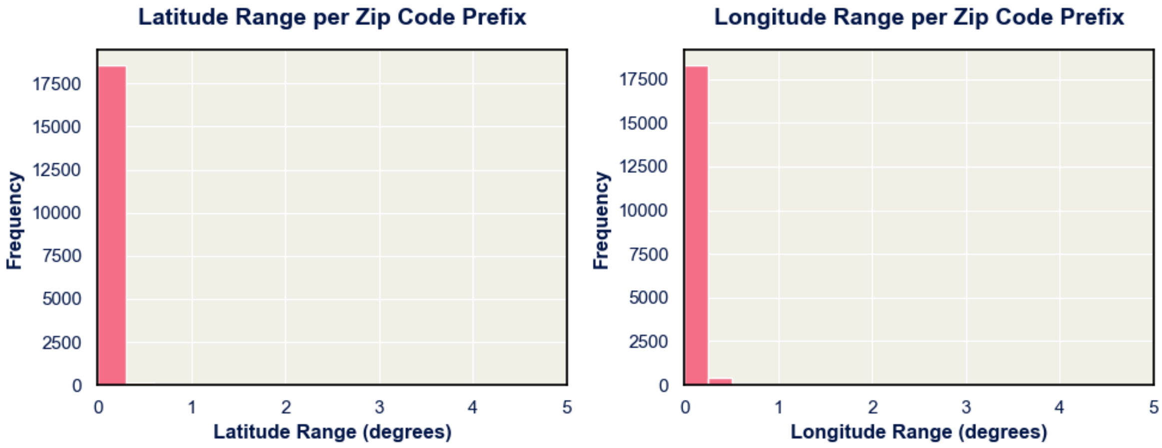
```
print(geolocation["geolocation_lat"].describe())
print(geolocation["geolocation_lng"].describe())
```

```
count    720463.000000
mean        -20.976145
std           5.909088
min         -33.692616
25%         -23.601834
50%         -22.861560
75%         -19.916176
max           4.482242
Name: geolocation_lat, dtype: float64
count    720463.000000
mean        -46.453543
std           4.408896
min         -72.930746
25%         -48.911812
50%         -46.645903
75%         -43.787556
max         -34.793685
Name: geolocation_lng, dtype: float64
```

In [366...

```
# Latitude and longitude range statistics per zip code prefix.
geo_range = etl.compute_geolocation_ranges(geolocation)
```

```
# Display the computed ranges
viz.plot_lat_lng_range_histograms(geo_range)
```



In [367…

```
noisy_zips = geo_range[
    (geo_range["lat_range"] > 0.5) | (geo_range["lng_range"] > 0.5)
]
print(f"{len(noisy_zips)} zip prefixes have high spatial variance.")
```

```
417 zip prefixes have high spatial variance.
```

In [368…

```
clean_geo = geolocation[~geolocation["geolocation_zip_code_prefix"].isin(noisy_z
```

will be merged with customers.csv and sellers.csv to plot a map for the distribution of sellers and customers. Could also be used to calculate distance between the 2 groups.

sellers.csv

In [369…

```
sellers.head()
```

Out[369…

	seller_id	seller_zip_code_prefix	seller_city	seller_state
0	3442f8959a84dea7ee197c632cb2df15	13023	campinas	SP
1	d1b65fc7debc3361ea86b5f14c68d2e2	13844	mogi guacu	SP
2	ce3ad9de960102d0677a81f5d0bb7b2d	20031	rio de janeiro	RJ
3	c0f3eea2e14555b6faeea3dd58c1b1c3	4195	sao paulo	SP
4	51a04a8a6bdcb23deccc82b0b80742cf	12914	braganca paulista	SP

In [370…

```
sellers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3095 entries, 0 to 3094
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   seller_id               3095 non-null   object
 1   seller_zip_code_prefix  3095 non-null   int64
 2   seller_city             3095 non-null   object
 3   seller_state            3095 non-null   object
dtypes: int64(1), object(3)
memory usage: 96.8+ KB
```

In [371…
```
#check for duplicates
etl.null_duplicate_check(sellers, verbose=False)
```

```
INFO - No null values found.
INFO - No duplicates found.
INFO - No duplicates found.
```

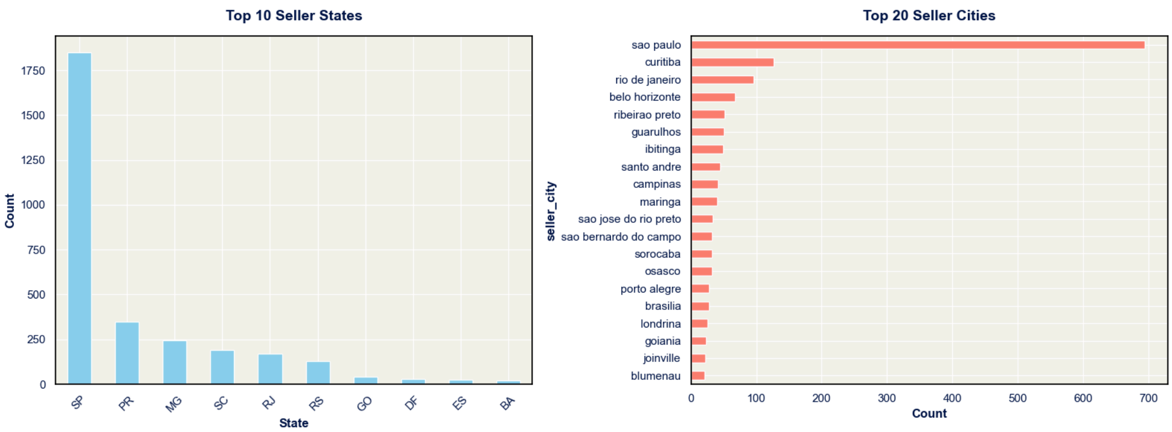
In [372…
```
formatted_sellers = etl.format_sellers(sellers)
```

In [373…
```
formatted_sellers.describe()
```

Out[373…

	seller_id	seller_zip_code_prefix	seller_city	seller_state
count	3095	3095	3095	3095
unique	3095	2246	609	23
top	9e25199f6ef7e7c347120ff175652c3b	14940	sao paulo	SP
freq	1	49	695	1849

In [374…
```
viz.plot_top_locations(formatted_sellers,
                       state_col="seller_state",
                       city_col="seller_city",
                       title_prefix="Seller")
```



Customer and seller geographic distribution shows strong overlap in urban regions such as São Paulo and Rio de Janeiro. This urban concentration, combined with higher seller density, could facilitate quicker deliveries and higher satisfaction—factors known to influence repeat buying behavior.

Review-related datasets [Habib]

- review.csv
- products.csv
- translation.csv

Review dataset

In [375…

```
reviews.head()
```

Out[375…

	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

In [376…

```
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99224 entries, 0 to 99223
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   review_id                99224 non-null  object
 1   order_id                 99224 non-null  object
 2   review_score             99224 non-null  int64 
 3   review_comment_title     11568 non-null  object
 4   review_comment_message   40977 non-null  object
 5   review_creation_date     99224 non-null  object
 6   review_answer_timestamp  99224 non-null  object
dtypes: int64(1), object(6)
memory usage: 5.3+ MB
```

In [377…

```
# .info shows missing entries
etl.null_duplicate_check(reviews, verbose=False)
```

```
WARNING - Null values found.
INFO - 90.08% or 89385 rows have 1 or more null values.
INFO - No duplicates found.
```

In [378…

```
# Step 1) Since many users about 90% did not give their comments a title I will
# there is no time aspect in this situation so that will be dropped as well. The

reviews.drop(columns=["review_id", "review_comment_title","review_creation_date"

reviews.head()

# Dataset is now less bloated and only relevant columns remain.
```

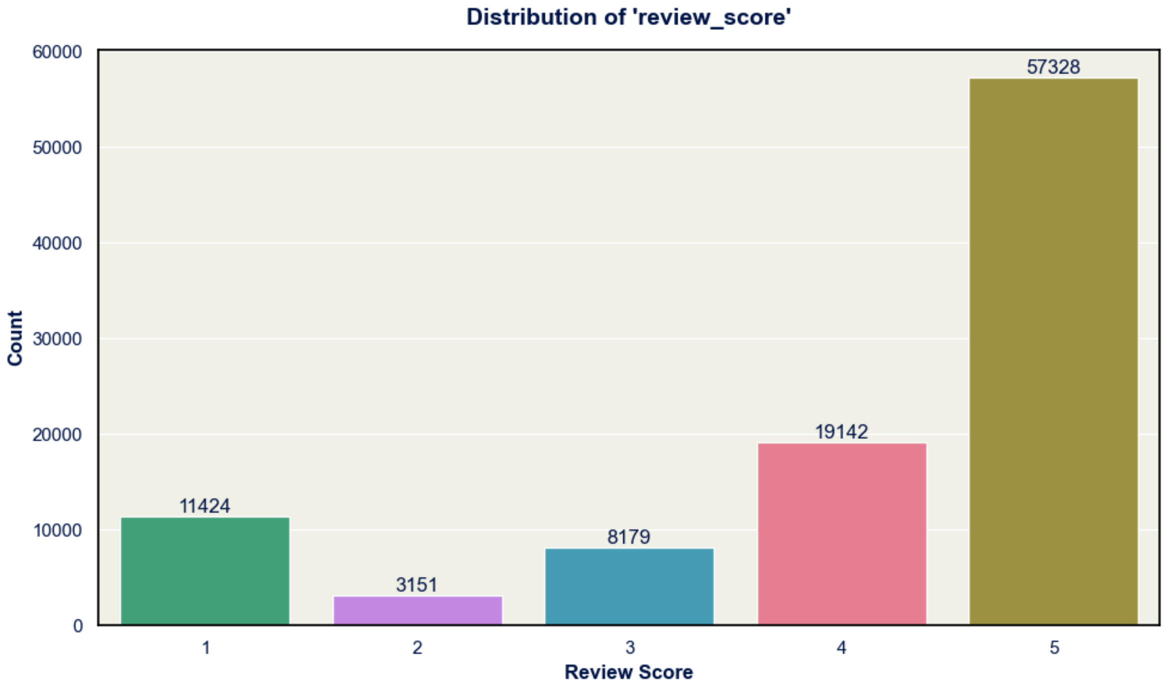
Out[378...

	order_id	review_score	review_comment_message
0	73fc7af87114b39712e6da79b0a377eb	4	NaN
1	a548910a1c6147796b98fdf73dbeba33	5	NaN
2	f9e4b658b201a9f2ecdecbb34bed034b	5	NaN
3	658677c97b385a9be170737859d3511b	5	Recebi bem antes do prazo estipulado.
4	8e6bfb81e283fa7e4f11123a3fb894f1	5	Parabéns lojas lannister adorei comprar pela I...

In [379...

```python
# Lets take a look at the spread of the review scores before any validation.

viz.plot_categorical_distribution(reviews, "review_score")
```

```
INFO - Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as numbers, cast to the appropriate data type before plotting.
INFO - Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as numbers, cast to the appropriate data type before plotting.
```



- We can see that there is a large bias towards 5 & 4 point reviews suggesting about 1 in 3 orders are satisfactory and leave a posititve impression on the end user.
- However there is a large amount of neutral reviews. To prevent non useful datapoints, this will be checked against the sentiment of the comments to verify all ratings are as intended.

Products dataset

In [380...

```python
products.head()
```

Out[380…

	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	
3	cef67bcfe19066a932b7673e239eb23d	bebes	27.0	
4	9dc1a7de274444849c219cff195d0b71	utilidades_domesticas	37.0	

In [381…

```
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32951 entries, 0 to 32950
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   product_id                  32951 non-null  object 
 1   product_category_name       32341 non-null  object 
 2   product_name_lenght         32341 non-null  float64
 3   product_description_lenght  32341 non-null  float64
 4   product_photos_qty          32341 non-null  float64
 5   product_weight_g            32949 non-null  float64
 6   product_length_cm           32949 non-null  float64
 7   product_height_cm           32949 non-null  float64
 8   product_width_cm            32949 non-null  float64
dtypes: float64(7), object(2)
memory usage: 2.3+ MB
```

In [382…

```
# .info shows small amount of missing entries
etl.null_duplicate_check(products, verbose=False)
```

```
WARNING - Null values found.
INFO - 1.85% or 611 rows have 1 or more null values.
INFO - No duplicates found.
```

In [383…

```
# 2) There is no possible way to feature engineer or substitute placeholder valu

products.dropna(inplace=True)

etl.null_duplicate_check(products)
```

```
INFO - No null values found.
INFO - No duplicates found.
```

In [384…

```
# 3) Convert float to int for certain columns

products = products.astype({
    "product_name_lenght": int,
    "product_description_lenght": int,
    "product_photos_qty": int
})

# Checking all dt conversions worked as intended
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 32340 entries, 0 to 32950
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   product_id                  32340 non-null  object 
 1   product_category_name       32340 non-null  object 
 2   product_name_lenght         32340 non-null  int64  
 3   product_description_lenght  32340 non-null  int64  
 4   product_photos_qty          32340 non-null  int64  
 5   product_weight_g            32340 non-null  float64
 6   product_length_cm           32340 non-null  float64
 7   product_height_cm           32340 non-null  float64
 8   product_width_cm            32340 non-null  float64
dtypes: float64(4), int64(3), object(2)
memory usage: 2.5+ MB
```

Translation Dataset

In [385…

```
translation.head()
# From what I can see, this is a small dataset for end users to translate the po
```

Out[385…

	product_category_name	product_category_name_english
0	beleza_saude	health_beauty
1	informatica_acessorios	computers_accessories
2	automotivo	auto
3	cama_mesa_banho	bed_bath_table
4	moveis_decoracao	furniture_decor

In [386…

```
translation.info()
# There are no missing or duplicate values. Nor any further visualisations, so I
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71 entries, 0 to 70
Data columns (total 2 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   product_category_name          71 non-null     object
 1   product_category_name_english  71 non-null     object
dtypes: object(2)
memory usage: 1.2+ KB
```

Order-related datasets [Min]

- items.csv
- payments.csv
- orders.csv

items.csv

In [387…

```
items.head()
```

Out[387…

	order_id	order_item_id	product_ic
0	00010242fe8c5a6d1ba2dd792cb16214	1	4244733e06e7ecb4970a6e2683c13e61
1	00018f77f2f0320c557190d7a144bdd3	1	e5f2d52b802189ee658865ca93d83a8
2	000229ec398224ef6ca0657da4fc703e	1	c777355d18b72b67abbeef9df44fd0fc
3	00024acbcdf0a6daa1e931b038114c75	1	7634da152a4610f1595efa32f14722fc
4	00042b26cf59d7ce69dfabb4e55b4fd9	1	ac6c3623068f30de03045865e4e10089

In [388…

```
items.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112650 entries, 0 to 112649
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   order_id             112650 non-null  object 
 1   order_item_id        112650 non-null  int64  
 2   product_id           112650 non-null  object 
 3   seller_id            112650 non-null  object 
 4   shipping_limit_date  112650 non-null  object 
 5   price                112650 non-null  float64
 6   freight_value        112650 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 6.0+ MB
```

In [389…

```
# df info says no Null but still; Null and duplicate check
etl.null_duplicate_check(items)
```

```
INFO - No null values found.
INFO - No duplicates found.
```

In [390…

```
# 1. shipping_limit_date has wrong data type.
items["shipping_limit_date"] = etl.to_datetime(items["shipping_limit_date"])
assert items["shipping_limit_date"].dtype == "datetime64[ns]", "shipping_limit_d

# order_item_id is categircal, so object type is fine

# Some duplicates are expected after dropping order_item_id, since there can be
# Left untreated as these duplicates are identifiable and workable with the orde
etl.null_duplicate_check(items, verbose=False)
```

```
INFO - No null values found.
INFO - No duplicates found.
```

In [391…

```
# Check distribution of numeric columns
items.describe()
```

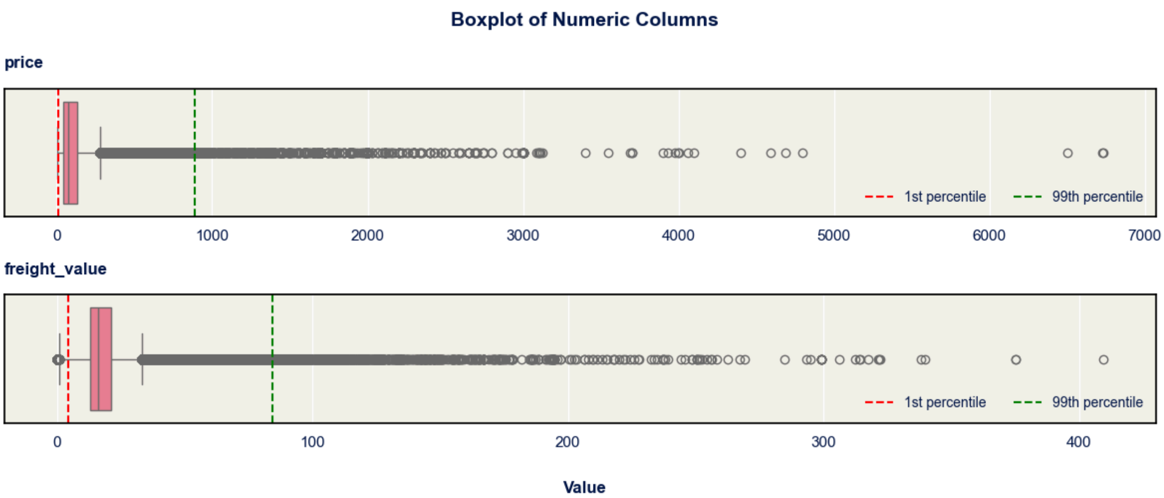
Out[391…

	order_item_id	shipping_limit_date	price	freight_value
count	112650.000000	112650	112650.000000	112650.000000
mean	1.197834	2018-01-07 15:36:52.192685312	120.653739	19.990320
min	1.000000	2016-09-19 00:15:34	0.850000	0.000000
25%	1.000000	2017-09-20 20:57:27.500000	39.900000	13.080000
50%	1.000000	2018-01-26 13:59:35	74.990000	16.260000
75%	1.000000	2018-05-10 14:34:00.750000128	134.900000	21.150000
max	21.000000	2020-04-09 22:35:08	6735.000000	409.680000
std	0.705124	NaN	183.633928	15.806405

- A large gap between the 75th percentile (Q3) and the maximum in prices indicates that there are high-value outliers.
- Olist dataset distributor mentioned (in Kaggle) that the data is from 2016 to 2018.

In [392…

```
# 2. Drop rows not within 2016-2018
items = items[items["shipping_limit_date"].dt.year.isin([2016, 2017, 2018])]

# Plot distribution of numeric columns
viz.plot_numeric_distribution(items.drop("order_item_id", axis=1))
```



- Both price & freight_value columns show strong right-skewed distributions with a large number of high-value outliers, which is expected as people mostly buy FMCGs (with low cost) from online, not high-end products.
- Right-skewed price means the item is expensive while right-skewed freight_value means either the customer lives far away from sellers or the item is physically huge; this can be cross-checked later with geolocation and product data.

In [393…

```
price_99p = items['price'].quantile(0.999)
print(f"99.9% of price is ${price_99p}.")
freight_99p = items['freight_value'].quantile(0.999)
print(f"99.9% of freight_value is ${freight_99p:.2f}.")
```

```
99.9% of price is $2110.0.
99.9% of freight_value is $175.59.
```

High values in both price and freight_value are important as they may suggest:

- People who bought expensive quality producuts are less likely to buy again.
- People who have to pay high freight_value are less likely to buy again. But, capturing 99.9% of current customers should be representative enough.

In [394…

```
# 3. Remove outliers
items['price'] = etl.cap_outliers(items['price'], min_cap=False, max_cap=.999)
items['freight_value'] = etl.cap_outliers(items['freight_value'], min_cap=False,
```

In [395…

```
# 4. Flag high values for the model to learn that a price/freight is unusually h
items['price_high'] = items['price'] > items['price'].quantile(0.75) * 1.5
items['freight_value_high'] = items['freight_value'] > items['freight_value'].qu
items.freight_value_high.value_counts()
```

Out[395…

```
freight_value_high
False    112646
Name: count, dtype: int64
```

orders.csv

In [396…

```
orders.head()
```

Out[396…

	order_id	customer_id	order_status
0	e481f51cbdc54678b7cc49136f2d6af7	9ef432eb6251297304e76186b10a928d	delivered
1	53cdb2fc8bc7dce0b6741e2150273451	b0830fb4747a6c6d20dea0b8c802d7ef	delivered
2	47770eb9100c2d0c44946d9cf07ec65d	41ce2a54c0b03bf3443c3d931a367089	delivered
3	949d5b44dbf5de918fe9c16f97b45f8a	f88197465ea7920adcdbec7375364d82	delivered
4	ad21c59c0840e6cb83a9ceb5573f8159	8ab97904e6daea8866dbdbc4fb7aad2c	delivered

In [397…

```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 8 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   order_id                       99441 non-null  object
 1   customer_id                    99441 non-null  object
 2   order_status                   99441 non-null  object
 3   order_purchase_timestamp       99441 non-null  object
 4   order_approved_at              99281 non-null  object
 5   order_delivered_carrier_date   97658 non-null  object
 6   order_delivered_customer_date  96476 non-null  object
 7   order_estimated_delivery_date  99441 non-null  object
dtypes: object(8)
memory usage: 6.1+ MB
```

In [398…

```python
# 1. Convert to datetime and clip to 2016-2018
for col in orders.columns[3:]:
    orders[col] = etl.clip_datetime(orders[col])
    assert orders[col].dtype == "datetime64[ns]", f"{col} should be datetime64[n
```

- order_delivered_carrier_date (which marks when the seller handed the package to the carrier) is mainly for logistic purposes and does not influence repeat buyer behaviour as comparied to the duration between purchase and delivery.

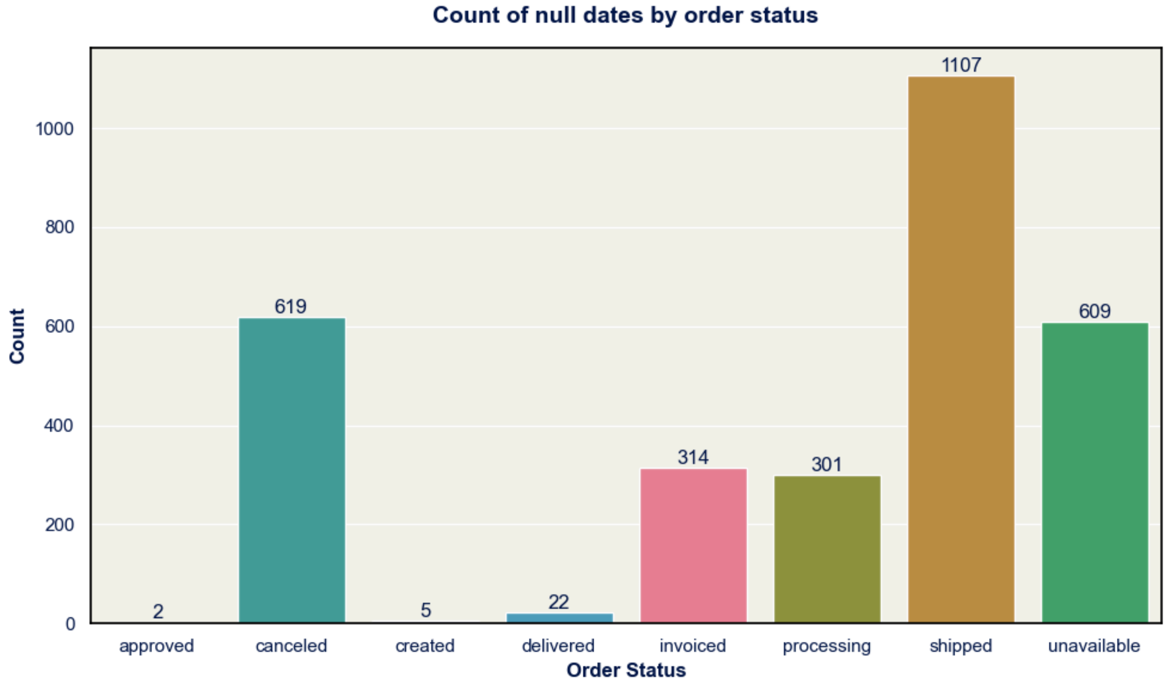
In [399…

```python
# 2. Drop order_delivered_carrier_date
orders.drop(columns=["order_delivered_carrier_date"], inplace=True)
assert "order_delivered_carrier_date" not in orders.columns, "order_delivered_ca

# Null values
etl.null_duplicate_check(orders, verbose=False)
```

```
WARNING - Null values found.
INFO - 3.00% or 2979 rows have 1 or more null values.
INFO - No duplicates found.
```

In [400…

```python
# ~3% of the data shows that the order_delivered_customer_date is null
# meaning that the order was not delivered or have not been delivered yet to the

# Check if null dates are related to different order statuses
null_dates_orders = orders[orders.isnull().any(axis=1)].copy()
viz.plot_categorical_distribution(null_dates_orders, "order_status", "Count of n
```

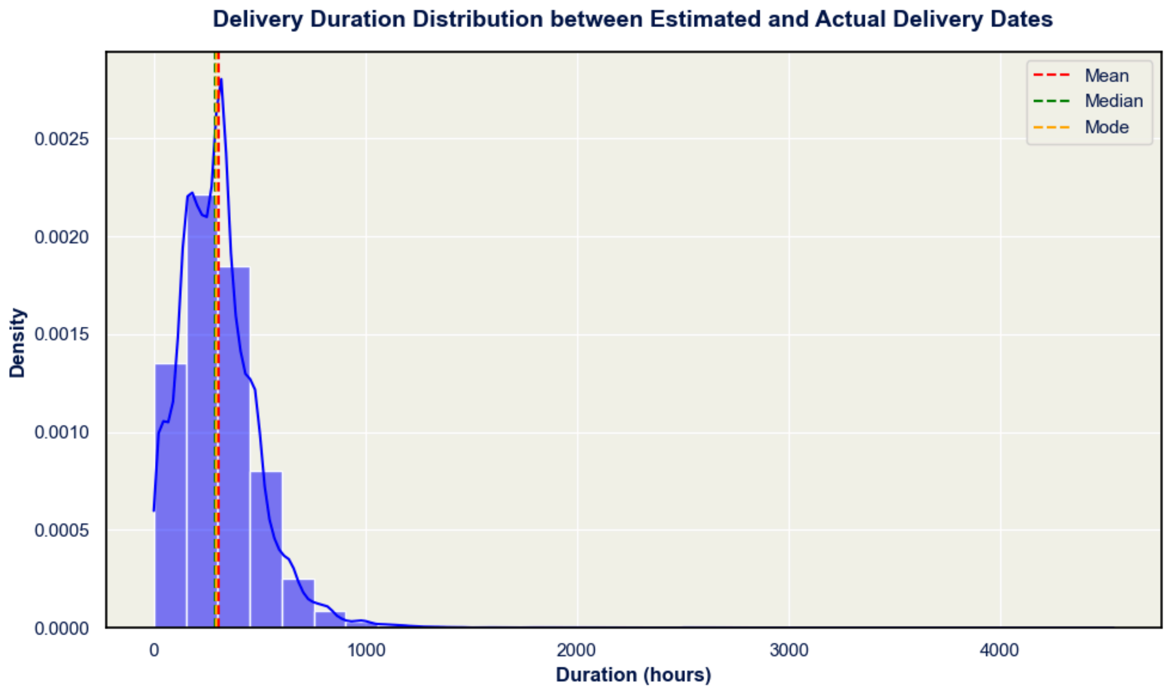


- Since the goal is to identify potential repeat buyer, we will focus on 'paid' statuses; dropping others while trying to impute the null values.

In [401…

```
# 3. Drop rows with [canceled, created, invoiced, unavailable] order_status
orders = orders[~orders["order_status"].isin(["canceled", "created", "invoiced",

# Vislualize the duration between order_approved_at and order_delivered_customer
mean, median, mode = viz.plot_duration_distribution(
                        df=orders,
                        column_x='order_delivered_customer_date',
                        column_y='order_estimated_delivery_date',
                        title='Delivery Duration Distribution between Estimated
                    )
```

```
WARNING - NaN values found in datetime columns. They will be ignored in duration calculation.
```



- The duration is right-skewed, and the mean is dragged towards right due to extreme outliers.

In [402…

```
# 4. Fill null values in order_delivered_customer_date with the median duration
orders["order_delivered_customer_date"] = orders["order_delivered_customer_date"
assert orders["order_delivered_customer_date"].isnull().sum() == 0, "There shoul

# For null order_approved_at,
orders[orders.order_approved_at.isna()]['order_status'].value_counts()
```

Out[402…

```
order_status
delivered    14
Name: count, dtype: int64
```

- order_status shows delivered, but order_approved_at (payment time) and order_delivered_customer_date is null, meaning these are mix-mactched; incorrect data (maybe the customer used points to exchange instead of payment?)

In [403…

```
# 5. drop NULL in order_approved_at
orders.dropna(inplace=True)
assert orders.isnull().sum().sum() == 0, "There should be no null values in the
```

payments.csv

In [404…

```
payments.head()
```

Out[404…

	order_id	payment_sequential	payment_type	payment_ins
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	

In [405…

```
payments.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103886 entries, 0 to 103885
Data columns (total 5 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   order_id              103886 non-null  object 
 1   payment_sequential    103886 non-null  int64  
 2   payment_type          103886 non-null  object 
 3   payment_installments  103886 non-null  int64  
 4   payment_value         103886 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 4.0+ MB
```

In [406…

```
# They are all in correct data types.
```

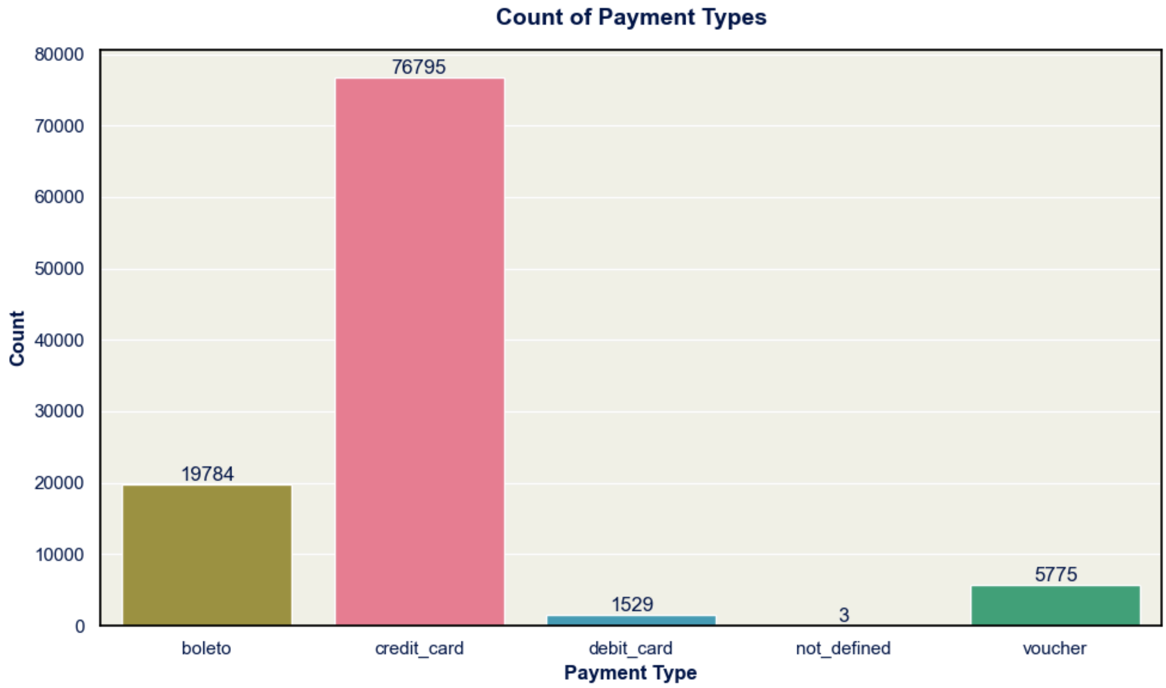
```
etl.null_duplicate_check(payments)
```

```
INFO - No null values found.
INFO - No duplicates found.
```

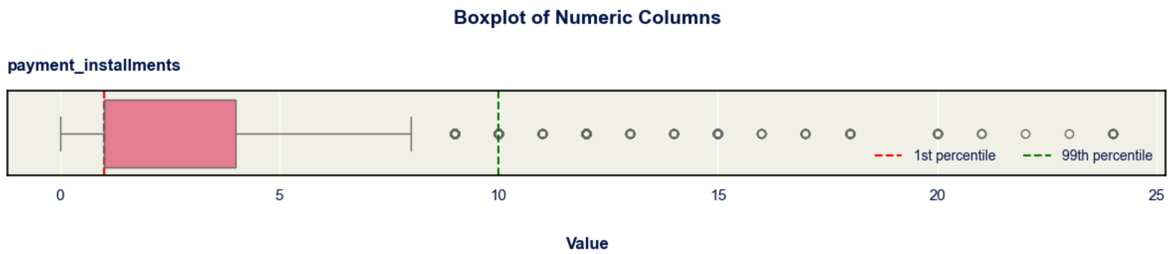
In [407…
```
# it seems the school intentionally screw the data btw ^^
payments.order_id.nunique()
```

Out[407…
```
99440
```

In [408…
```
# Which payment types are commonly used?
viz.plot_categorical_distribution(payments, "payment_type", "Count of Payment Ty
```



In [409…
```
# A glance at installments
viz.plot_numeric_distribution(payments.drop(columns=["order_id", "payment_sequen
```



- There is no info on not_defined payment type and logically does not make sense; dropped.
- payment_installments is right-skewed, and that many times of installments are not common either; capped.
- payment_sequential is based on customer preference and is redundant; we can work with groupby later if necessary; dropped.

In [410…
```
# 1. drop not_defined payment types
payments = payments[payments["payment_type"] != "not_defined"]
assert "not_defined" not in payments["payment_type"].unique(), "not_defined paym
```

```
# 2. cap installments to 99% quantile
payments["payment_installments"] = etl.cap_outliers(payments["payment_installmen

# 3. drop payment_sequential as it is not relevant for the analysis; payment_val
payments.drop(columns=["payment_sequential"], inplace=True)
assert "payment_sequential" not in payments.columns, "payment_sequential should
```

In [411…

```
df = payments.groupby("order_id")["payment_installments"].nunique()
multiple_installments = df[df > 2].index.values
payments[payments["order_id"].isin(multiple_installments[:1])]
```

Out[411…

	order_id	payment_type	payment_installments	paym
15259	1d251ab94983c4adb11e4b168abb1439	credit_card	4	
32551	1d251ab94983c4adb11e4b168abb1439	voucher	1	
93089	1d251ab94983c4adb11e4b168abb1439	credit_card	6	

- People are weird; for the same order, he chose to pay with a voucher, a credit card with 4 installments and another credit card with 6 installments.
- Since this is also a part of Olist's flexible payment, these installments will just be added together to reduce complexity while keeping info.