

**A CASE STUDY REPORT SUMMITTED AS A PART OF EXPERIENTIAL LEARNING ON**

**“ANALYZING BRAZILIAN E-COMMERCE DATA FROM O-LIST” SUBMITTED BY:-**

|  |  |
| --- | --- |
| **NAME** | **REGISTRATION NUMBER** |
| **Aditya Sinha** | **CL20250106018930112** |
| **Jaijeet Paul** | **CL2025010601897533** |
| **Chandan Choudhury** | **CL2025010601911046** |
| **Aditya Nayak** | **CL2025010601911349** |
| **Biswojit Lenka** | **CL20250106018928110** |
| **Ch. Jagan Subudhi** | **CL20250106018923105** |
| **Aman Verma** | **CL2025010601911147** |

**Under the Guidance of CRANES VARSITY**

# DECLARATION

We hereby declare that the case study report entitled **"ANALYZING BRAZILIAN E-COMMERCE DATA FROM O-LIST"** is an original work completed by us as part of our academic curriculum. This case study report is submitted to the Department of Computer Science Engineering,

C.V. Raman Global University, Bhubaneswar, Odisha, under the guidance of **Cranes Varsity.** All the information, data, designs, and analysis included in this report are authentic and reflect our own efforts, except where specified otherwise. Any references or resources used have been properly acknowledged in the report's references section.

This project has been conducted solely for academic purposes, and any resemblance to existing products or systems is purely coincidental. We affirm that this work has not been submitted, either fully or in part, for any other course or assessment.

Signature of Students

Signature of Supervisor Signature of HOD

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Cranes Varsity** for their unwavering guidance and support throughout the development of this project on **Analyzing Brazilian E-Commerce Data from Olist**. Their mentorship and technical expertise played a pivotal role in shaping the direction and depth of this analytical study.

A special thanks to my instructors and mentors at Cranes Varsity, whose consistent encouragement and insightful feedback significantly contributed to my growth in the field of data science and analytics. Their expertise in areas such as data engineering, machine learning, and business intelligence has been instrumental in enabling me to conduct a thorough and meaningful analysis.

I am also deeply thankful to my peers and family members for their continuous motivation and support. Their belief in me has been a constant source of inspiration throughout this journey.

Lastly, I am grateful for the opportunity to apply my knowledge of programming, data analysis, and visualization techniques in a practical setting. This project has allowed me to develop a data-driven solution that offers valuable insights into Brazil’s e-commerce landscape and the business operations of Olist.

# ABSTRACT

In the rapidly expanding landscape of e-commerce, understanding customer behavior, operational efficiency, and market dynamics is critical for business growth and sustainability. This project presents a comprehensive analysis of the **Brazilian E-Commerce Public Dataset by Olist**, one of the most robust and richly structured datasets available for retail analytics.

The dataset consists of multiple interconnected tables that capture the entire customer journey—from order placement and payment to delivery and customer feedback. The primary objective of this project is to explore and extract actionable insights that can help enhance business strategies related to marketing, logistics, and customer satisfaction.

To achieve this, extensive **Exploratory Data Analysis (EDA)** was conducted using Python libraries such as **pandas**, **matplotlib**, **seaborn**, and **plotly**. Data preprocessing involved cleaning null values, merging datasets based on key relationships, and performing feature engineering on date and geographic fields. Special attention was given to delivery time performance, product categories, payment preferences, customer reviews, and geographic trends across Brazil.

In addition to static visualizations, a dynamic and interactive **Streamlit dashboard** was developed to present the findings in an intuitive format, enabling real-time exploration of sales metrics, top-performing product categories, seller activity, and customer distribution across states. Geographic visualizations using **Folium** helped to identify regional disparities in customer base and delivery times.

Key insights from the analysis revealed trends such as the dominance of credit card payments, the impact of delivery delays on customer reviews, and the correlation between product category and return rates. These insights can directly inform logistics planning, customer relationship management, and product positioning.

Overall, the project demonstrates the power of data-driven decision-making in the e-commerce sector and offers a scalable blueprint for similar analytical efforts in retail businesses globally.

# TABLE OF CONTENTS

1. **INTRODUCTION**
2. **DATASET DESCRIPTION**
3. **TOOLS AND TECHOLOGIES USED**
4. **DATA PREPROCESSING**
5. **EXPLORATORY DATA ANALYSIS (EDA)**
6. **DASHBOARD**
7. **SOURCE CODE**
8. **OUTPUT ANALYSIS**
9. **DASHBOARD IMPLEMENT AND ANALYSIS**
10. **KEY INSIGHTS**
11. **CHALLENGES FACED**
12. **FUTURE SCOPE**
13. **CONCLUSION**

# INTRODUCTION

In the digital era, **e-commerce has emerged as a cornerstone of global trade**, revolutionizing the way consumers purchase goods and services. With millions of transactions taking place every day, e-commerce platforms generate vast amounts of data that, when properly analyzed, can yield powerful insights into customer behavior, market trends, logistics performance, and business opportunities. This has given rise to the field of **e-commerce analytics**, which enables companies to make data-driven decisions to enhance profitability, improve customer satisfaction, and streamline operations.

**E-commerce analytics** involves the examination of metrics related to online sales, user interactions, product performance, and delivery logistics. It empowers businesses to:

* + Identify top-selling products and underperforming categories.
  + Understand customer preferences and segment their audience.
  + Optimize supply chain and delivery processes.
  + Enhance marketing strategies based on consumer behavior.
  + Reduce churn by proactively addressing pain points in the customer journey.

To delve into this domain, this project utilizes the **Brazilian E-Commerce Public Dataset provided by Olist**, a well-known multi-category marketplace operating in Brazil. Olist acts as an intermediary between small to medium-sized sellers and customers across the country, providing logistics and customer service support.

The dataset is extensive and includes:

* + **Order data** (status, purchase timestamps, delivery details)
  + **Customer demographics and location**
  + **Product attributes and categories**
  + **Seller performance**
  + **Payment and review information**

# DATASET DESCRIPTION

The analysis conducted in this project is based on the **Brazilian E-Commerce Public Dataset provided by Olist**, a comprehensive and real-world dataset available on Kaggle. It contains detailed transactional records from Olist, a Brazilian e-commerce marketplace that connects multiple small and medium- sized businesses with customers across Brazil. The dataset captures a wide range of interactions in the e-commerce pipeline, offering rich opportunities for exploratory analysis and business insight generation. There 9 structured CSV files and one product category translation file

## Key Components of the Dataset

The dataset is composed of the following core tables:

## orders.csv

* + Contains unique order records including order status, purchase timestamp, and estimated vs. actual delivery dates.
  + Acts as a central hub linking customers, sellers, payments, products, and reviews.

## order\_items.csv

* + Detailed product-level data per order.
  + Includes price, freight cost, product ID, seller ID, and shipping information.

## products.csv

* + Metadata about the products such as category, weight, dimensions, and product names.
  + Useful for analyzing product performance by category or size/weight attributes.

## customers.csv

* + Contains customer location and identification details.
  + Enables geographic and demographic analysis of the customer base.

## sellers.csv

* + Contains seller location information.
  + Useful for analyzing distribution of sellers and delivery distances.

## order\_payments.csv

* + Payment-related information including type (credit card, voucher, etc.), number of installments, and payment value.

## order\_reviews.csv

* + Captures customer satisfaction data through star ratings and review comments.
  + Valuable for sentiment analysis and correlation with delivery times or product quality.

## product\_category\_name\_translation.csv

* + Provides English translations for the Portuguese product category names, enhancing readability and analysis.

## geolocation.csv

* + Contains latitude and longitude coordinates of customer zip codes.
  + Enables map visualizations and spatial analytics of demand and delivery.

**TOOLS AND TECHOLOGIES USED**

To effectively analyze the Brazilian E-Commerce dataset from Olist and derive meaningful insights, a combination of modern programming tools, libraries, and platforms was used. These technologies supported all stages of the project—from data wrangling and exploratory analysis to dashboard development and version control.

## Programming Language: Python

* + Python was the primary language used throughout the project due to its simplicity, versatility, and strong ecosystem for data science.
  + It enabled seamless data processing, analysis, and visualization.
  + Python scripts and Jupyter notebooks formed the core of data exploration and interpretation.

## Libraries and Frameworks

The following Python libraries were used extensively in this project:

## pandas

* + Used for data loading, cleaning, transformation, and manipulation.
  + Enabled efficient handling of large tabular datasets and merging of multiple relational files.

## matplotlib

* + Served as the foundational plotting library for creating basic visualizations like line charts, bar graphs, and histograms.
  + Useful for visualizing trends over time and comparing categorical data.

## seaborn

* + Built on top of matplotlib, Seaborn was used for statistical data visualization.
  + Helped in plotting complex visualizations like heatmaps and boxplots with minimal code.

## plotly

* + Used for interactive and dynamic visualizations.
  + Enabled zoomable and hover-enabled charts that enhanced data storytelling.

## folium

* + A powerful mapping library used to create interactive geographic maps.
  + Enabled visualization of customer and seller locations across Brazil using geolocation data.

## Visualization Tools

The project relied heavily on data visualization to derive and present insights:

* + **matplotlib** and **seaborn** were used for static visualizations during EDA.
  + **plotly** was used for interactive charts that allowed for deeper exploration of data patterns.
  + **folium** was used for spatial analysis, such as plotting customer distribution across Brazilian states and cities.

## Dashboarding Tool: Streamlit

* + A custom dashboard was built using **Streamlit** (dashboard/dashboard.py) to make the analysis interactive and user- friendly.
  + The dashboard allows users to view:
    - Sales metrics
    - Top categories and sellers
    - Delivery time analysis
    - Geographic insights
  + Streamlit was chosen for its simplicity and ability to deploy dashboards quickly using Python code alone.

 **Visualizations**: [Plotly Express](https://plotly.com/python/plotly-express/) and [Matplotlib](https://matplotlib.org/) for creating dynamic charts.

 **Machine Learning**:

[XGBoost](https://xgboost.readthedocs.io/) for regression tasks.

[Scikit-learn](https://scikit-learn.org/) for model training and evaluation.

[TensorFlow](https://www.tensorflow.org/) for building the LSTM model.

## Jupyter Notebook

* + The main analysis and visualizations were developed in a Jupyter Notebook (notebook.ipynb).
  + The notebook format made it easy to document the analysis process, display visual outputs alongside code, and share insights in a readable format.

## Version Control: GitHub

* + GitHub was used for managing the project source code and maintaining version control.
  + Enabled collaboration, tracking of code changes, and sharing of the final project via the repository:

https://github.com/ChandanChoudhury7727/Data-Analytics-Brazilian-Ecommerce.git

Together, these tools formed a powerful and efficient data science stack that supported robust data analysis, insightful visualizations, and interactive reporting.

# DATA PREPROCESSING

Data preprocessing is a critical phase in any data science project. It ensures that the raw data is cleaned, structured, and transformed into a format suitable for analysis. In this project, data preprocessing involved a series of steps including dataset merging, handling of missing values, data type conversion, and feature engineering. These steps were essential to unlock the full analytical potential of the Olist dataset.

## Merging Datasets

The Olist dataset comprises multiple interrelated CSV files. To perform a holistic analysis, it was necessary to merge these datasets based on common keys:

* + orders.csv served as the central table and was joined with:
    - order\_items.csv using order\_id
    - order\_reviews.csv using order\_id
    - order\_payments.csv using order\_id
    - customers.csv using customer\_id
  + order\_items.csv was further joined with:
    - products.csv using product\_id
    - sellers.csv using seller\_id
  + geolocation.csv was used to map zip codes to geographic coordinates for spatial visualization.

This relational joining of tables allowed for the creation of a unified and enriched dataset where all aspects of the customer journey (purchase, payment, delivery, feedback) could be analyzed together.

## Handling Missing Values

Handling missing data was a crucial step to ensure the reliability of the results. To identify missing values across the datasets, a custom visualization script named **plotMissingValue.py** was used.

Key steps included:

* + **Visualization of null values**: Heatmaps and bar charts were generated to detect which columns and rows contained missing values.

## Strategy for handling:

* + - Columns with a small number of missing entries (e.g., delivery dates) were imputed or filtered depending on the context.
    - Columns with excessive missing values or irrelevant attributes were dropped.
    - In cases where missing data had a semantic meaning (e.g., orders not yet delivered), such rows were preserved for specific analyses.

This step ensured that the datasets were clean and complete enough to derive accurate and trustworthy insights.

## Data Type Conversion

Correct data types are essential for accurate computation and visualization. Key conversions included:

* + **Date columns** such as order\_purchase\_timestamp, order\_delivered\_customer\_date, and shipping\_limit\_date were converted from strings to datetime objects using pd.to\_datetime().
  + **Categorical fields** such as order\_status, payment\_type, and product\_category\_name were cast to category data types to improve performance and facilitate analysis.
  + **Numeric conversions** were applied to ensure all monetary and quantitative fields were ready for aggregation and statistical calculations.

## Feature Engineering

To enhance the dataset and extract deeper insights, new features were engineered, especially from timestamp and location data:

## Time-based Features

* + **Order Year, Month, Day**: Extracted from the order\_purchase\_timestamp to analyze seasonality and monthly trends.
  + **Delivery Time**: Calculated as the difference between order\_delivered\_customer\_date and order\_purchase\_timestamp.
  + **Estimated vs. Actual Delivery Delay**: Measured to assess the efficiency of delivery logistics.
  + **Weekday of Purchase**: To understand buying behavior based on days of the week.

## Location-based Features

* + **Customer State and City Grouping**: Used to map demand density across Brazil.
  + **Delivery Distance Estimation**: Approximated using geolocation coordinates to evaluate logistics impact.

This thorough data preprocessing stage laid the foundation for a smooth and insightful analytical process, enabling meaningful interpretations and data- driven business decisions.

# EXPLORATORY DATA ANALYSIS (EDA)

The Exploratory Data Analysis (EDA) phase was essential in uncovering patterns, relationships, and anomalies within the Olist Brazilian E-Commerce dataset. Using a combination of statistical summaries and visualizations, the analysis provided valuable business insights into sales performance, customer behavior, product demand, and logistics efficiency.

This section summarizes the key analyses and findings, supported by various chart types including line plots, bar charts, pie charts, histograms, and heatmaps. Geographic patterns were visualized using **Folium** maps.

## Sales Trend Over Time

* + **Objective**: Understand the overall growth and seasonality of online sales.
  + **Method**: Aggregated order purchase timestamps on a monthly basis and visualized using a **line chart**.

## Findings:

* + - A steady increase in the number of orders was observed from late 2016 to mid-2018.
    - Peak sales occurred during **November and December**, likely due to holiday shopping and Black Friday promotions.
    - Dips in sales during mid-year months suggest seasonal variation in demand.

## Top-Selling Product Categories

* + **Objective**: Identify the most popular product categories by sales volume.
  + **Method**: Counted total items sold per category and visualized with a

## horizontal bar chart.

* + **Findings**:
    - **"bed\_bath\_table"**, **"health\_beauty"**, and **"sports\_leisure"** were among the top-selling categories.
    - The long tail of the chart showed many niche categories with low sales volume, indicating a wide but uneven product offering.

## Payment Method Distribution

* + **Objective**: Analyze customer preferences for payment methods.
  + **Method**: Counted the number of transactions by payment type and represented with a **pie chart** and **bar plot**.

## Findings:

* + - **Credit card** payments dominated, accounting for over **70%** of all transactions.
    - Other payment types included **boleto (bank slip)**, **voucher**, and

**debit card**, each with smaller shares.

* + - Credit card transactions often involved installment payments, reflecting consumer purchasing behavior in Brazil.

## Customer Location Heatmaps (Folium)

* + **Objective**: Visualize the geographic distribution of customers.
  + **Method**: Used geolocation data to create a **Folium heatmap** based on customer latitude and longitude.

## Findings:

* + - High customer density was observed in major cities such as **São Paulo**, **Rio de Janeiro**, **Belo Horizonte**, and **Brasília**.
    - The Southeast region of Brazil showed the highest concentration of buyers, aligning with its economic dominance.

## Review Score Analysis

* + **Objective**: Evaluate customer satisfaction levels and identify factors influencing reviews.
  + **Method**: Analyzed the distribution of review scores using **histograms** and

## box plots.

* + **Findings**:
    - The majority of reviews were either **5 stars** (positive) or **1 star**

(negative), indicating polarized customer experiences.

* + - Orders delivered on time or earlier were more likely to receive higher review scores.
    - Longer delays correlated with lower scores, especially 1- and 2-star reviews, underscoring the importance of timely delivery.

## Delivery Time Performance

* + **Objective**: Assess the efficiency of order deliveries and compare estimated vs. actual delivery times.

## Method:

* + - Calculated delivery duration for each order.
    - Compared it to the estimated shipping limit.
    - Visualized the results using **histograms** and **violin plots**.

## Findings:

* + - The **average delivery time** was around **12 days**, with a wide variance depending on region and product.
    - A significant number of orders were delivered after the estimated date.
    - Delayed deliveries were a key contributor to customer dissatisfaction.

## Overall Insights from EDA

* + Seasonal peaks and regional trends offer opportunities for targeted marketing.
  + Fast-moving product categories can guide inventory and logistics planning.
  + Streamlining delivery operations may improve review scores and reduce churn.
  + Payment flexibility (e.g., installment options) is critical in customer acquisition.

# DASHBOARD

To enhance interactivity and make the analysis results accessible to business users and stakeholders, an interactive dashboard was developed using **Streamlit**, a modern open-source framework for building data apps in Python.

The dashboard is implemented in the file: dashboard/dashboard.py

It provides a visual summary of the key performance indicators (KPIs) and insights derived from the dataset, offering users a simple and intuitive way to explore the Brazilian e-commerce data without needing to write code.

## Technology: Streamlit

* + **Streamlit** was chosen for its lightweight and rapid development capabilities.
  + It allows Python scripts to be transformed into shareable web apps with minimal overhead.
  + The framework supports interactivity through widgets like dropdowns, sliders, and checkboxes, all of which are utilized in the dashboard.

## Key KPIs Visualized in the Dashboard

The dashboard presents a variety of metrics and charts categorized under different tabs or sections:

## Total Sales Metrics

* + **Visuals**: Large, bold numeric displays and bar charts.

## Metrics Included:

* + - Total number of orders
    - Total revenue (sum of payment value.
    - Total number of unique customers and sellers
    - Average delivery duration
  + These metrics provide a quick snapshot of the overall business performance.

## Top Sellers

* + **Visuals**: Ranked bar chart or horizontal leaderboard.

## Metrics Included:

* + - Top 10 sellers by total sales
    - Number of orders handled
    - Seller geographic location (optionally via a map)
  + Useful for identifying high-performing vendors and evaluating seller concentration.

## Category Performance

* + **Visuals**: Bar plots, pie charts, and time-series line graphs.

## Metrics Included:

* + - Most popular product categories by order count
    - Revenue contribution by category
    - Category-wise trends over time
  + Allows for category-level demand forecasting and inventory planning.

## Map-Based Insights

* + **Visuals**: Interactive maps built using **Folium**, embedded within the Streamlit app.

## Features:

* + - Heatmaps of customer density by location (city/state)
    - Seller distribution across Brazil
    - Delivery patterns and regional delivery delays
  + Enhances understanding of geographic market penetration and logistics performance.

**Machine Learning**:

[XGBoost](https://xgboost.readthedocs.io/) for regression tasks.

[Scikit-learn](https://scikit-learn.org/) for model training and evaluation.

[TensorFlow](https://www.tensorflow.org/) for building the LSTM model.

## Dashboard Layout and Usability

The dashboard is structured to ensure a user-friendly experience:

## Sidebar Filters:

* + - Date range selectors
    - Order status filters (e.g., delivered, shipped)
    - Category filters
    - Region/state dropdowns

## Responsive Layout:

* + - KPIs at the top for high-level overview
    - Scrollable sections with charts below
    - Tabs or expanders for deeper analysis areas (e.g., reviews, payments)

## Interactivity:

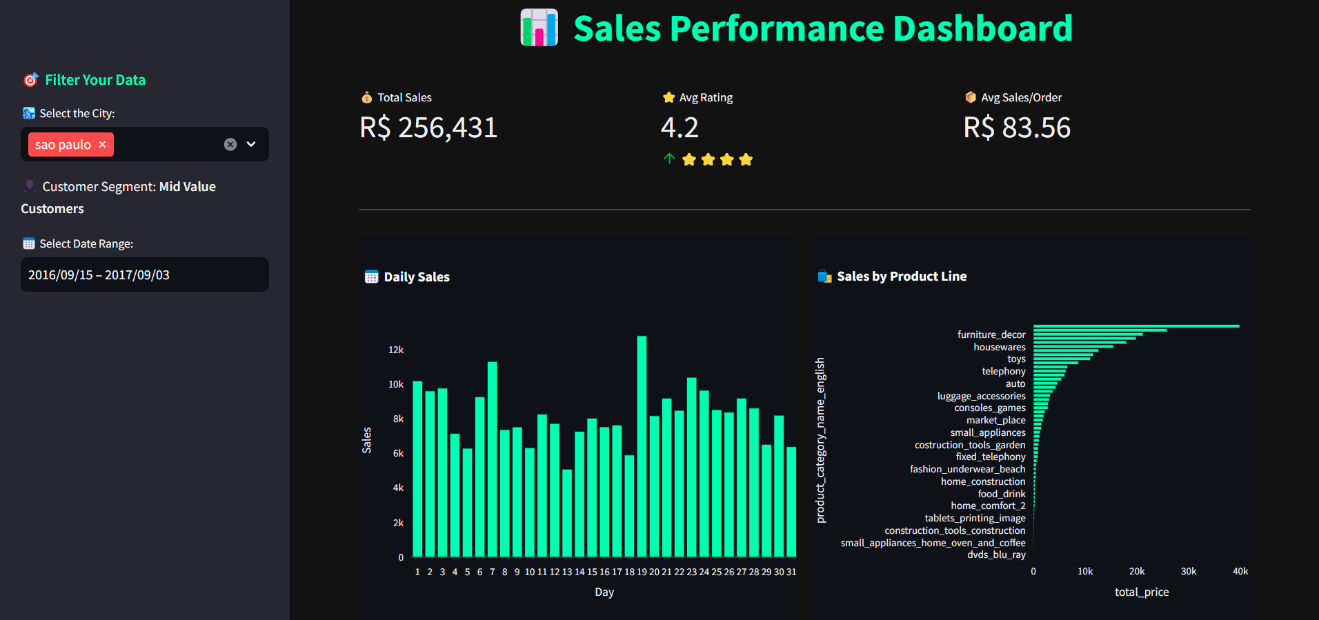
* + - Users can dynamically filter data and immediately see updated visualizations.
    - Hover tooltips on charts and maps provide more granular information.

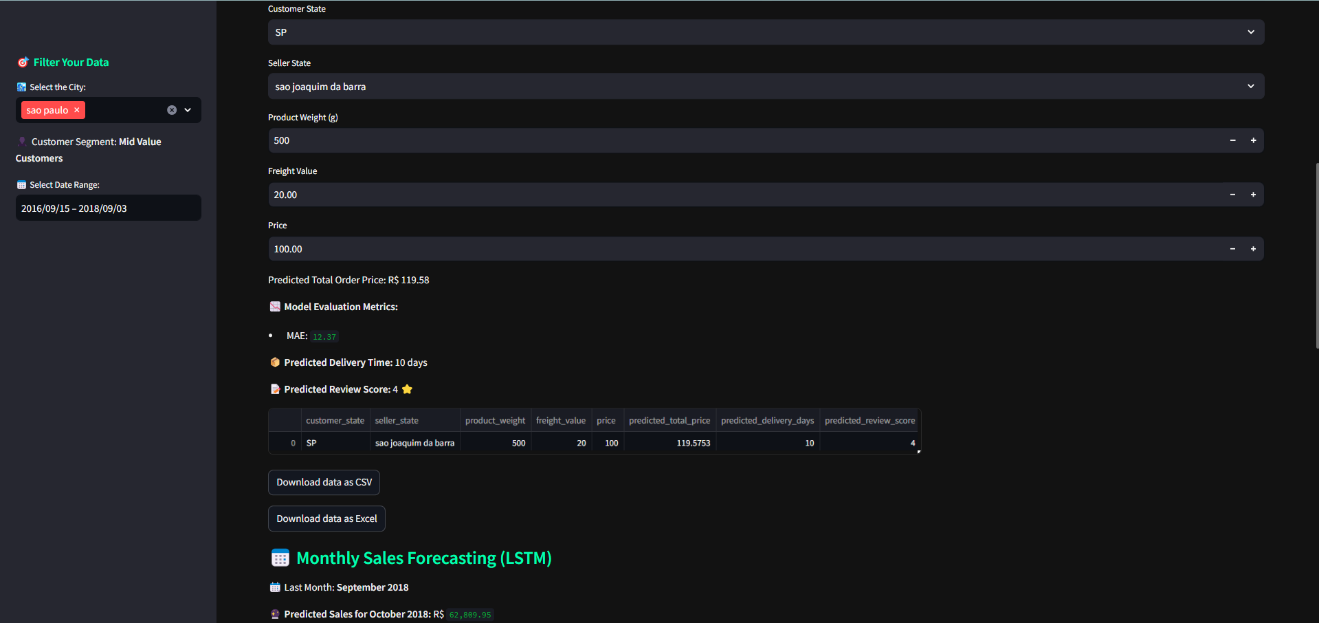
## Purpose and Benefits

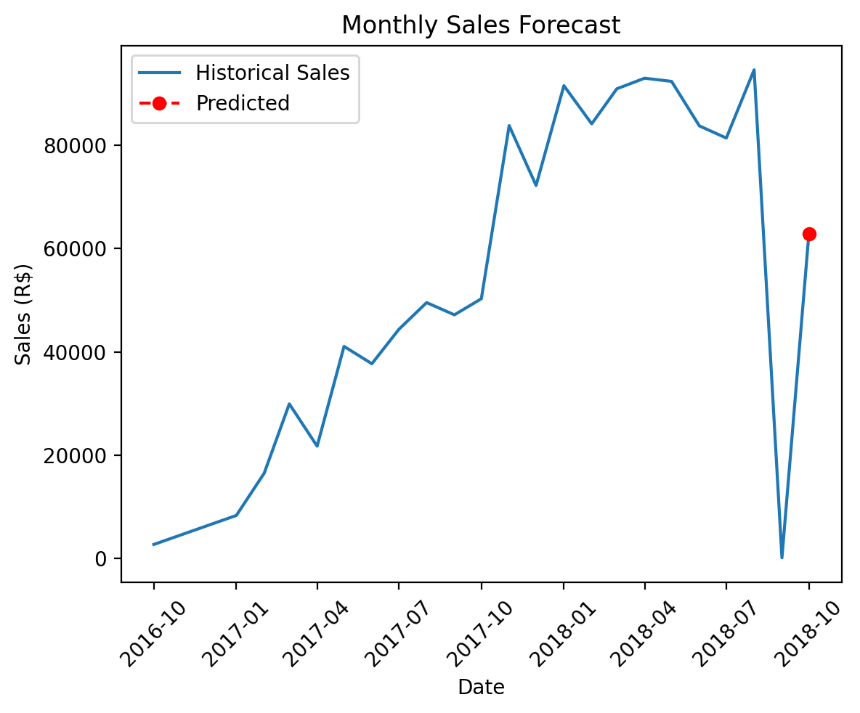
* + Designed for both **analysts** and **non-technical stakeholders**.
  + Enables **real-time exploration** of the dataset without needing to rerun scripts.
  + Helps in **monitoring sales**, **identifying performance bottlenecks**, and

**making informed decisions** based on data.

This Streamlit dashboard adds immense value to the project by transforming static analysis into an engaging, interactive platform for business intelligence.







# SOURCE CODE

### Prepare all the library needed

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from scipy import stats

from module.plotMissingValue import percentage as pmv

1. **Read the customer data and save it to variable `customers\_df`** customers\_df = pd.read\_csv("./data/customers\_dataset.csv") customers\_df.head()
2. **Read the geolocation data and save it to variable `geolocation\_df`** geolocation\_df = pd.read\_csv("./data/geolocation\_dataset.csv") geolocation\_df.head()
3. **Read the order items data and save it to variable`order\_items\_df`** order\_items\_df = pd.read\_csv("./data/order\_items\_dataset.csv") order\_items\_df.head()

### Read the order payments data and save it to variable `order\_payments\_df`

order\_payments\_df = pd.read\_csv("./data/order\_payments

\_dataset.csv") order\_payments\_df.head()

1. **Read the order reviews data and save it to variable `order\_reviews\_df`** order\_reviews\_df = pd.read\_csv("./data/order\_reviews\_dataset.csv") order\_reviews\_df.head()

### Read the orders data and save it to variable `orders\_df`

orders\_df = pd.read\_csv("./data/orders\_dataset.csv") orders\_df.head()

1. **Read the products data and save it to variable `products\_df`** products\_df = pd.read\_csv("./data/products\_dataset.csv") products\_df.head()

### Read the product category name data and save it to variable `product\_category\_name\_df`

product\_category\_name\_df = pd.read\_csv("./data/product

\_category\_name\_translation.csv") product\_category\_name\_df.head()

1. **Read the sellers data and save it to variable `sellers\_df`** sellers\_df = pd.read\_csv("./data/sellers\_dataset.csv") sellers\_df.head()

### Assess the customers table

# Check the summary of the dataset and its columns customers\_df.info()

# Check the missing value in the table customers\_df customers\_df.isna().sum()

# Check the duplicate value in the table customers\_df duplicate\_value = customers\_df.duplicated() print(f"Total duplicate value: {duplicate\_value.sum()}") # Check statistics of the table customers\_df customers\_df.describe()

### Assess the orders table

# Check the summary of the dataset and its columns orders\_df.info()

# Check the missing value in the table orders\_df orders\_df.isnull().sum()

# Check the duplicate value in the table orders\_df duplicate\_value = orders\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}")

### Assess the order\_items table

# Check the summary of the dataset and its columns order\_items\_df.info()

# Check the missing value in the table order\_items\_df order\_items\_df.isnull().sum()

# Check the duplicate value in the table order\_items\_df duplicate\_value = order\_items\_df.duplicated() print(f"Total duplicate value: {duplicate\_value.sum()}") # Check the duplicate value in order lines

order\_lines\_duplicate = order\_items\_df[["order\_id", "product\_id"]].duplicated().sum() print(f"Total duplicate value in order lines: {order\_lines\_duplicate}")

# Check the statistics of the table order\_items\_df order\_items\_df.describe() print(order\_items\_df.loc[order\_items\_df["freight\_value"].idxmax()]) # Plot the boxplot of the feature price sns.boxplot(x=order\_items\_df["price"])

# Plot the histogram of the attribute price plt.figure(figsize=(12, 6)) sns.histplot(data=order\_items\_df, x="price", bins=100) # Plot the boxplot of the feature freight\_value sns.boxplot(x=order\_items\_df["freight\_value"])

# Plot the histogram of the attribute freight\_value plt.figure(figsize=(12, 6))

sns.histplot(data=order\_items\_df, x="freight\_value", bins=100)

### Assess the order\_payments table

# Check the summary of the dataset and its columns order\_payments\_df.info()

# Check the missing value in the table order\_payments\_df order\_payments\_df.isnull().sum().sort\_values(ascending=False) # Check the duplicate value in the table order\_payments\_df duplicate\_value = order\_payments\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}") # Check the statistics

order\_payments\_df.describe()

# Plot the boxplot of the feature payment\_value sns.boxplot(x=order\_payments\_df["payment\_value"])

### Assess the order\_reviews table

# Check the summary of the dataset and its columns order\_reviews\_df.info()

#Check the missing value in the table order\_reviews\_df order\_reviews\_df.isnull().sum().sort\_values(ascending=False) # Check the duplicate value in the table order\_reviews\_df duplicate\_value = order\_reviews\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}") # Check the statistics

order\_reviews\_df.describe()

### Assess the products table

# Check the summary of the dataset and its columns products\_df.info()

# Check the decimal values in the attribute product\_name\_lenght products\_df["product\_name\_lenght"].apply(

lambda x: x != round(x) if pd.notnull(x) else False

).any()

# Check the decimal values in the attribute product\_description\_lenght products\_df["product\_description\_lenght"].apply(

lambda x: x != round(x) if pd.notnull(x) else False

).any()

# Check the decimal values in the attribute product\_photos\_qty products\_df["product\_photos\_qty"].apply(

lambda x: x != round(x) if pd.notnull(x) else False# Check the decimal values in the attribute product\_photos\_qty

products\_df["product\_photos\_qty"].apply(

lambda x: x != round(x) if pd.notnull(x) else False

).any()

# Check the decimal values in the attribute product\_length\_cm products\_df["product\_length\_cm"].apply(

lambda x: x != round(x) if pd.notnull(x) else False

).any()

# Check the decimal values in the attribute product\_length\_cm products\_df["product\_length\_cm"].apply(

lambda x: x != round(x) if pd.notnull(x) else False

).any()

# Check the decimal values in the attribute product\_width\_cm products\_df["product\_width\_cm"].apply(

lambda x: x != round(x) if pd.notnull(x) else False

).any()

# Check the missing value in the table products\_df products\_df.isnull().sum().sort\_values(ascending=False) # Check the duplicate value in the table products\_df duplicate\_value = products\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}") # Check the statistics

products\_df.describe()

### Assess the product\_category table

# Check the summary of the dataset and its columns product\_category\_name\_df.info()

# Check the missing value in the table product\_category\_name\_df product\_category\_name\_df.isnull().sum().sort\_values(ascending=False) # Check the missing value in the table product\_category\_name\_df product\_category\_name\_df.isnull().sum().sort\_values(ascending=False) # Check the duplicate value in the table product\_category\_name\_df duplicate\_value = product\_category\_name\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}") # Check the statistics

product\_category\_name\_df.describe()

### Assess the sellers table

# Check the summary of the dataset and its columns sellers\_df.info()

# Check the missing value in the table sellers\_df sellers\_df.isnull().sum().sort\_values(ascending=False) # Check the missing value in the table sellers\_df sellers\_df.isnull().sum().sort\_values(ascending=False) # Check the duplicate value in the table sellers\_df duplicate\_value = sellers\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}")

### Assess the geolocation table

# Check the summary of the dataset and its columns geolocation\_df.info()

# Check the missing value in the table geolocation\_df geolocation\_df.isnull().sum().sort\_values(ascending=False) # Check the duplicate value in the table geolocation\_df duplicate\_value = geolocation\_df.duplicated()

print(f"Total duplicate value: {duplicate\_value.sum()}")

### Cleaning the customers table

# Change the customer\_zip\_code\_prefix attribute to string customers\_df["customer\_zip\_code\_prefix"] = customers\_df[

"customer\_zip\_code\_prefix"

].astype(str)

# Check the data type of the customer\_zip\_code\_prefix attribute customers\_df.info()

# Change the order\_purchase\_timestamp attribute to datetime orders\_df["order\_purchase\_timestamp"] = pd.to\_datetime(

orders\_df["order\_purchase\_timestamp"]

)

# Change the order\_approved\_at attribute to datetime orders\_df["order\_approved\_at"] = pd.to\_datetime(orders\_df["order\_approved\_at"]) # Change the order\_delivered\_carrier\_date attribute to datetime orders\_df["order\_delivered\_carrier\_date"] = pd.to\_datetime(

orders\_df["order\_delivered\_carrier\_date"]

)

# Change the order\_delivered\_customer\_date attribute to datetime orders\_df["order\_delivered\_customer\_date"] = pd.to\_datetime(

orders\_df["order\_delivered\_customer\_date"]

)

# Change the order\_estimated\_delivery\_date attribute to datetime orders\_df["order\_estimated\_delivery\_date"] = pd.to\_datetime(

orders\_df["order\_estimated\_delivery\_date"]

)

# Check the data type of the attributes orders\_df.info()

# Define a function to convert the data type of a column to datetime def convert\_to\_datetime(dataFrame, columns):

for column in columns:

dataFrame[column] = pd.to\_datetime(dataFrame[column], format="%Y-%m-%d").dt.date # Check the data type of the attributes

orders\_df.head()

**Handling missing value** orders\_df.isnull().sum().sort\_values(ascending=False) # Plot the missing values

missing\_percentage = pmv(orders\_df) orders\_df.head()

# First three "order\_delivered\_customer\_date" values are empty. So, we will drop them before using fll() method

orders\_df.drop(orders\_df.index[:3], inplace=True) # Check the head of the dataset

orders\_df.head()

# Substitute the missing values with the previous values (forward fill): orders\_df["order\_delivered\_customer\_date"] = orders\_df[

"order\_delivered\_customer\_date"

].ffill()

# Check the missing values again

print("Missing values: ", orders\_df["order\_delivered\_customer\_date"].isnull().sum())

# A column with ordinal order dates will also be included. This numerical format will ease further tasks. # Adding a column with ordinal order dates

orders\_df["date\_ordinal"] = orders\_df["order\_purchase\_timestamp"].apply( lambda date: date.toordinal()

)

# Convert columns to date time orders\_df["order\_estimated\_delivery\_date"] = pd.to\_datetime(

orders\_df["order\_estimated\_delivery\_date"]

)

orders\_df["order\_delivered\_customer\_date"] = pd.to\_datetime( orders\_df["order\_delivered\_customer\_date"]

)

orders\_df["order\_purchase\_timestamp"] = pd.to\_datetime( orders\_df["order\_purchase\_timestamp"]

)

# Create shipping time column, which is the difference between the order estimated delivery date and the order delivered customer date

orders\_df["shipping\_time"] = ( orders\_df["order\_estimated\_delivery\_date"]

- orders\_df["order\_delivered\_customer\_date"]

)

# Create shipping duration column, which is the difference between the order delivered customer date and the order purchase timestamp

orders\_df["shipping\_duration"] = (

orders\_df["order\_delivered\_customer\_date"] - orders\_df["order\_purchase\_timestamp"]

)

# Create estimated duration column, which is the difference between the order estimated delivery date and the order purchase timestamp

orders\_df["estimated\_duration"] = (

orders\_df["order\_estimated\_delivery\_date"] - orders\_df["order\_purchase\_timestamp"]

)

orders\_df.head(10)

# Check if there are repeating orders

print(f"Number of unique orders: {orders\_df['order\_id'].nunique()}") print(f"Number of records: {orders\_df.shape[0]}")

### Cleaning the order items table

# Change the order\_item\_id attribute to string

order\_items\_df["order\_item\_id"] = order\_items\_df["order\_item\_id"].astype(str) # Check the data type of the order\_item\_id attribute

order\_items\_df.info()

# Check the statistics of the price attribute before removing outliers order\_items\_df["price"].describe()

# Plot a boxplot to visualize the outliers plt.figure(figsize=(12, 6)) sns.boxplot(x=order\_items\_df["price"]) plt.title("Boxplot of Price Outlier") plt.show()

# Handling outliers of the price attribute # Calculate Q1, Q3 and IQR

Q1 = order\_items\_df["price"].quantile(0.25) Q3 = order\_items\_df["price"].quantile(0.75)

IQR = Q3 - Q1

# Define bounds for the acceptable range lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

# Remove rows with 'price' outside the acceptable range order\_items\_df = order\_items\_df[

(order\_items\_df["price"] >= lower\_bound) & (order\_items\_df["price"] <= upper\_bound)

]

# Check the boxplot of the price attribute after removing outliers plt.figure(figsize=(12, 6))

sns.boxplot(x=order\_items\_df["price"]) plt.title("Boxplot of Price after Removing Outliers") plt.show()

# Check the statistics of the price attribute after removing outliers order\_items\_df["price"].describe()

# Check the statistics of the freight\_value attribute before removing outliers order\_items\_df["freight\_value"].describe()

# Plot a boxplot to visualize the outliers plt.figure(figsize=(12, 6)) sns.boxplot(x=order\_items\_df["freight\_value"]) plt.title("Boxplot of Freight Value's Outlier") plt.show()

# Handling outliers of the freight\_value attribute # Handling outliers of the freight\_value attribute # Calculate Q1, Q3 and IQR

Q1 = order\_items\_df["freight\_value"].quantile(0.25) Q3 = order\_items\_df["freight\_value"].quantile(0.75)

IQR = Q3 - Q1

# Define bounds for the acceptable range lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

# Remove rows with 'freight\_value' outside the acceptable range order\_items\_df = order\_items\_df[

(order\_items\_df["freight\_value"] >= lower\_bound)

& (order\_items\_df["freight\_value"] <= upper\_bound)

]

# Check the boxplot of the freight\_value attribute after removing outliers plt.figure(figsize=(12, 6))

sns.boxplot(x=order\_items\_df["freight\_value"]) plt.title("Boxplot of Freight Value after Removing Outliers") plt.show()

# Check the statistics of the freight\_value attribute after removing outliers order\_items\_df["freight\_value"].describe()

### Transform the order\_item\_id into qty to extract unit-per-order line profile

order\_items\_consolidated\_df = ( order\_items\_df.groupby(by=["product\_id", "order\_id"])

.agg(

{

"order\_item\_id": "count", "seller\_id": "first", "shipping\_limit\_date": "first", "price": "first", "freight\_value": "first",

}

)

.reset\_index()

)

order\_items\_consolidated\_df

# Checking is there any info was lost

print(f"Orders in new table: {order\_items\_consolidated\_df['order\_id'].nunique()}") print(f"Orders in old table: {order\_items\_df['order\_id'].nunique()}")

print(f"SKUs in new table: {order\_items\_consolidated\_df['product\_id'].nunique()}") print(f"SKUs in old table: {order\_items\_df['product\_id'].nunique()}")

print(

f"Total quantity in new table: {order\_items\_consolidated\_df['order\_item\_id'].sum()}"

)

print(f"Total quantity in old table: {len(order\_items\_df['order\_item\_id'])}") # Renaming the order\_item\_id column to quantity

order\_items\_consolidated\_df.rename(columns={"order\_item\_id": "qty"}, inplace=True) order\_items\_consolidated\_df

### Cleaning the order payment table

# Check the statistics of the payment\_value attribute before removing outliers order\_payments\_df["payment\_value"].describe()

# Plot a boxplot to visualize the outliers plt.figure(figsize=(12, 6)) sns.boxplot(x=order\_payments\_df["payment\_value"]) plt.title("Boxplot of Payment Value's Outlier") plt.show()

# Handling outliers of the payment\_value attribute # Handling outliers of the payment\_value attribute # Calculate Q1, Q3 and IQR

Q1 = order\_payments\_df["payment\_value"].quantile(0.25) Q3 = order\_payments\_df["payment\_value"].quantile(0.75)

IQR = Q3 - Q1

# Define bounds for the acceptable range lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

# Remove rows with 'payment\_value' outside the acceptable range order\_payments\_df = order\_payments\_df[

(order\_payments\_df["payment\_value"] >= lower\_bound)

& (order\_payments\_df["payment\_value"] <= upper\_bound)

]

# Plot a boxplot of the payment\_value attribute after removing outliers plt.figure(figsize=(12, 6)) sns.boxplot(x=order\_payments\_df["payment\_value"]) plt.title("Boxplot of Payment Value after Removing Outliers") plt.show()

# Check the statistics of the freight\_value attribute after removing outliers order\_payments\_df["payment\_value"].describe()

### Cleaning the order reviews table

order\_reviews\_df.info()

# Change the review\_creation\_date attribute to datetime order\_reviews\_df["review\_creation\_date"] = pd.to\_datetime(

order\_reviews\_df["review\_creation\_date"]

)

# Change the review\_answer\_timestamp attribute to datetime order\_reviews\_df["review\_answer\_timestamp"] = pd.to\_datetime(

order\_reviews\_df["review\_answer\_timestamp"]

)

# Check the data type of the attributes order\_reviews\_df.info() order\_reviews\_df.head()

### Handling missing value

# Backup the dataset order\_reviews\_df order\_reviews\_cleaned\_df = order\_reviews\_df.copy()

order\_reviews\_cleaned\_df.isnull().sum().sort\_values(ascending=False) # Fill the missing values in review\_comment\_title with "No comment"

order\_reviews\_cleaned\_df["review\_comment\_title"].fillna("No comment", inplace=True) # Fill the missing values in review\_comment\_message with "No comment"

order\_reviews\_cleaned\_df["review\_comment\_message"].fillna("No comment", inplace=True) # Check the missing values again

print(f"Missing values: {order\_reviews\_cleaned\_df.isnull().sum().sum()}")

# Merge the dataframes order\_reviews that filled the missing values with the missing values of the original dataframe

order\_reviews\_df = pd.merge(

order\_reviews\_cleaned\_df, order\_reviews\_df, how="left"

).copy() order\_reviews\_cleaned\_df.head()

### Cleaning the products table

# Change the product\_name\_lenght attribute to product\_name\_length products\_df.rename(columns={"product\_name\_lenght": "product\_name\_length"}, inplace=True) products\_df.info()

# Change the product\_name\_lenght attribute to product\_name\_length products\_df.rename(

columns={"product\_description\_lenght": "product\_description\_length"}, inplace=True

)

products\_df.info() products\_df.isnull().sum().sort\_values(ascending=False) # Plot the missing values for each column missing\_percentage = pmv(products\_df)

# Check the unique values of the product\_category\_name attribute

print(f"Number of unique values:\n {products\_df['product\_category\_name'].unique()}") print(

f"Frequency of each value in product\_category\_name attribute:\n

{products\_df['product\_category\_name'].value\_counts()}"

)

# Find the mode of the product\_category\_name attribute mode\_pcn = products\_df["product\_category\_name"].mode()[0]

# Fill the missing values in product\_category\_name with the mode products\_df["product\_category\_name"].fillna(mode\_pcn, inplace=True) # Plot the distribution values of the product\_name\_length attribute sns.histplot(data=products\_df, x="product\_name\_length", bins=20)

# Plot the distribution values of the product\_description\_length attribute sns.histplot(data=products\_df, x="product\_description\_length", bins=20) # Plot the distribution values of the product\_photos\_qty attribute sns.histplot(data=products\_df, x="product\_photos\_qty", bins=20)

# Plot the distribution values of the product\_weight\_g attribute sns.histplot(data=products\_df, x="product\_weight\_g", bins=20) # Plot the distribution values of the product\_length\_cm attribute sns.histplot(data=products\_df, x="product\_length\_cm", bins=20) # Plot the distribution values of the product\_height\_cm attribute

sns.histplot(data=products\_df, x="product\_height\_cm", bins=20) # Plot the distribution values of the product\_width\_cm attribute sns.histplot(data=products\_df, x="product\_width\_cm", bins=20) median\_pnl = products\_df["product\_name\_length"].median()

median\_pdl = products\_df["product\_description\_length"].median() median\_ppq = products\_df["product\_photos\_qty"].median() median\_pw = products\_df["product\_weight\_g"].median() median\_pl = products\_df["product\_length\_cm"].median() median\_ph = products\_df["product\_height\_cm"].median() median\_pw = products\_df["product\_width\_cm"].median()

# Fill the missing values with the median products\_df["product\_name\_length"].fillna(median\_pnl, inplace=True) products\_df["product\_description\_length"].fillna(median\_pdl, inplace=True) products\_df["product\_photos\_qty"].fillna(median\_ppq, inplace=True) products\_df["product\_weight\_g"].fillna(median\_pw, inplace=True) products\_df["product\_length\_cm"].fillna(median\_pl, inplace=True) products\_df["product\_height\_cm"].fillna(median\_ph, inplace=True) products\_df["product\_width\_cm"].fillna(median\_pw, inplace=True)

# Check the missing values again

print(f"Missing values: {products\_df.isnull().sum().sum()}")

### Handling format data type

# Change the datatype of the product\_name\_length attribute to int products\_df["product\_name\_length"] = products\_df["product\_name\_length"].astype(int) # Change the datatype of the product\_description\_length attribute to int products\_df["product\_description\_length"] = products\_df[

"product\_description\_length"

].astype(int)

# Change the datatype of the product\_photos\_qty attribute to int products\_df["product\_photos\_qty"] = products\_df["product\_photos\_qty"].astype(int) # Change the datatype of the product\_weight\_g attribute to int products\_df["product\_weight\_g"] = products\_df["product\_weight\_g"].astype(int)

# Change the datatype of the product\_length\_cm attribute to int

products\_df["product\_length\_cm"] = products\_df["product\_length\_cm"].astype(int) # Change the datatype of the product\_height\_cm attribute to int products\_df["product\_height\_cm"] = products\_df["product\_height\_cm"].astype(int) # Change the datatype of the product\_width\_cm attribute to int products\_df["product\_width\_cm"] = products\_df["product\_width\_cm"].astype(int) products\_df.head()

### Cleaning the sellers table

sellers\_df.info()

# Change the seller\_zip\_code\_prefix attribute to string sellers\_df["seller\_zip\_code\_prefix"] = sellers\_df["seller\_zip\_code\_prefix"].astype(str) # Check the data type of the seller\_zip\_code\_prefix attribute

sellers\_df.info()

### Cleaning the geolocation table

# Change the geolocation\_zip\_code\_prefix attribute to string geolocation\_df["geolocation\_zip\_code\_prefix"] = geolocation\_df[

"geolocation\_zip\_code\_prefix"

].astype(str)

# Check the data type of the geolocation\_zip\_code\_prefix attribute geolocation\_df.info()

### Handling the data duplication

# Drop the duplicate values in the geolocation\_df geolocation\_df.drop\_duplicates(

subset=["geolocation\_zip\_code\_prefix"], keep="first", inplace=True

)

# Check the duplicate values in the geolocation\_df duplicate\_value = geolocation\_df.duplicated() print(f"Total duplicate value: {duplicate\_value.sum()}")

Handling Inconsistent Values

# Drop the duplicate values in the geolocation\_df geolocation\_df.drop\_duplicates(

subset=["geolocation\_zip\_code\_prefix"], keep="first", inplace=True

)

# Check the duplicate values in the geolocation\_df duplicate\_value = geolocation\_df.duplicated() print(f"Total duplicate value: {duplicate\_value.sum()}")

# Check the frequency of each value in attribute geolocation\_city geolocation\_df.groupby(by=["geolocation\_city", "geolocation\_zip\_code\_prefix"]).agg(

{"geolocation\_city": "count"}

)

geolocation\_df.groupby(by=["geolocation\_city", "geolocation\_zip\_code\_prefix"]).agg(

{"geolocation\_city": "nunique"}

)

# Data homogenization

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "abadiânia" if x == "abadiania" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "abaeté" if x == "abaete" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "abaré" if x == "abare" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "abatiá" if x == "abatia" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "são paulo" if x == "sao paulo" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "cidade gaúcha" if x == "cidade" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "arraial do cabo" if x == "...arraial do cabo" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "quarto centenário" if x == "4o. centenario" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "quarto centenário" if x == "4º centenario" else x

)

geolocation\_df.geolocation\_city = geolocation\_df.geolocation\_city.apply( lambda x: "quarto centenário" if x == "quarto centenario" else x

)

geolocation\_df.geolocation\_city.unique()

### Merging datasets

# Combine the table orders and order items

sales\_df = orders\_df.merge(order\_items\_consolidated\_df, on="order\_id") # Check the summary of the dataset and its columns

sales\_df.info()

# Check the statistics of the dataset sales\_df.describe()

**Merging the products table with the order\_with\_items table** # Combine the table orders\_with\_items and order payments sales\_df = sales\_df.merge(products\_df, on="product\_id") sales\_df

# Check the summary of the dataset and its columns sales\_df.info()

# Check the statistics of the dataset sales\_df.describe().transpose()

### Merging the sales table with the product category name table

# Combine the table sales and product category name

sales\_df = sales\_df.merge(product\_category\_name\_df, on="product\_category\_name") sales\_df

# Check the summary of the dataset and its columns sales\_df.info()

### Exploratory Data Analysis (EDA)

sales\_df.info()

# Check the order delivery that is not delivered yet sales\_df[sales\_df["order\_delivered\_carrier\_date"].isnull()]

# Check the distribution of each value in the order\_status attribute sales\_df.groupby(by=("order\_status")).size().reset\_index(name="counts").sort\_values(

by="counts", ascending=False

)

# Visualize the distribution of the order\_status attribute plt.figure(figsize=(15, 6))

plt.title("Distribution of Order Status") plt.xlabel("Order Status") plt.ylabel("Count")

sns.countplot(x="order\_status", data=sales\_df)

# Check how many orders are delivered on time based on the estimated delivery date sales\_df["on\_time"] = np.where(

sales\_df["order\_delivered\_customer\_date"]

<= sales\_df["order\_estimated\_delivery\_date"]

)

sales\_df["on\_time"].value\_counts()

# Visualize the distribution of the on time attribute sns.countplot(x="on\_time", data=sales\_df) plt.title("Distribution of On Time")

plt.xlabel("On Time") plt.ylabel("Count") plt.show() sales\_df.info()

### Product Analysis

sales\_df.describe(include="all").transpose()

# Check the product category with the highest price sales\_df.groupby("product\_category\_name\_english").agg({"price": "sum"}).sort\_values(

by="price", ascending=False

)

# Check the product category with the lowest price sales\_df.groupby("product\_category\_name\_english").agg({"price": "sum"}).sort\_values(

by="price", ascending=True

)

# Prepare the data price\_vis = (

sales\_df.groupby("product\_category\_name\_english")

.agg({"price": "sum"})

.sort\_values(by="price", ascending=False)

)

# Visualize the distribution of the price on each product category plt.figure(figsize=(15, 6))

plt.title("Distribution of Price on Each Product Category") plt.xlabel("Product Category") plt.xticks(rotation="vertical")

plt.ylabel("Price") sns.barplot(

x="product\_category\_name\_english", y="price",

data=price\_vis, estimator=np.mean, ci=None,

)

# Check the product category with the highest freight value sales\_df.groupby("product\_category\_name\_english").agg(

{"freight\_value": "sum"}

).sort\_values(by="freight\_value", ascending=False)

# Check the product category with the lowest freight value sales\_df.groupby("product\_category\_name\_english").agg(

{"freight\_value": "sum"}

).sort\_values(by="freight\_value", ascending=True)

# Prepare the data for visualization freight\_value\_vis = (

sales\_df.groupby("product\_category\_name\_english")

.agg({"freight\_value": "sum"})

.sort\_values(by="freight\_value", ascending=False)

)

# Visualize the distribution of the freight value on each product category plt.figure(figsize=(15, 6))

plt.title("Distribution of Freight Value on Each Product Category") plt.xlabel("Product Category")

plt.xticks(rotation="vertical") plt.ylabel("Freight Value") sns.barplot(

x="product\_category\_name\_english", y="freight\_value", data=freight\_value\_vis, estimator=np.mean,

ci=None,

)

# Check the product category with the highest quantity sales sales\_df.groupby("product\_category\_name\_english").agg({"qty": "sum"}).sort\_values(

by="qty", ascending=False

)

# Check the product category with the lowest quantity sales sales\_df.groupby("product\_category\_name\_english").agg({"qty": "sum"}).sort\_values(

by="qty", ascending=True

)

# Prepare the data for visualization qty\_each\_category = (

sales\_df.groupby("product\_category\_name\_english")

.agg({"qty": "sum"})

.sort\_values(by="qty", ascending=False)

)

# Visualize the distribution of quantity sales on each product category plt.figure(figsize=(15, 6))

plt.title("Distribution of Quantity Sales on Each Product Category") plt.xlabel("Product Category")

plt.xticks(rotation="vertical") plt.ylabel("Quantity Sales") sns.barplot(

x="product\_category\_name\_english", y="qty",

data=qty\_each\_category, estimator=np.mean, ci=None,

)

# Check the product category with the fastest shipping duration sales\_df.groupby("product\_category\_name\_english").agg(

{"shipping\_duration": "sum"}

).sort\_values(by="shipping\_duration", ascending=True)

# Check the product category with the fastest shipping duration sales\_df.groupby("product\_category\_name\_english").agg(

{"shipping\_duration": "sum"}

).sort\_values(by="shipping\_duration", ascending=True)

# Check the product category with the longest shipping duration sales\_df.groupby("product\_category\_name\_english").agg(

{"shipping\_duration": "sum"}

).sort\_values(by="shipping\_duration", ascending=False)

### Payment Analysis

# Check the most frequent payment type order\_payments\_df.groupby(by=["payment\_type"]).size().reset\_index(

name="counts"

).sort\_values(ascending=False, by="counts")

# Visualize the distribution of payment type with pie chart plt.figure(figsize=(15, 6))

plt.title("Distribution of Payment Type") plt.pie(

order\_payments\_df.groupby(by=["payment\_type"])

.size()

.reset\_index(name="counts")["counts"], labels=order\_payments\_df.groupby(by=["payment\_type"])

.size()

.reset\_index(name="counts")["payment\_type"], autopct="%1.1f%%",

startangle=10,

)

plt.show() customers\_df.info()

# Check the distribution of the number of customers per city customers\_df.groupby("customer\_city").agg(

{"customer\_unique\_id": "nunique"}

).sort\_values(by="customer\_unique\_id", ascending=False)

# Prepare the data for visualization

customers\_per\_state = customers\_df["customer\_state"].value\_counts().reset\_index() customers\_per\_state.columns = ["customer\_state", "count"]

# Sort the data

customers\_per\_state = customers\_per\_state.sort\_values(by="count", ascending=False) # Visualize the count of the number of customers per state

plt.figure(figsize=(15, 6))

plt.title("Count of the Number of Customers per State") plt.xlabel("State")

plt.xticks(rotation="vertical") plt.ylabel("Number of Customers")

sns.barplot(x="customer\_state", y="count", data=customers\_per\_state.reset\_index())

### Reviews Analysis

# Merge the table sales and order\_reviews

sales\_df = sales\_df.merge(order\_reviews\_df, on="order\_id", how="left")

# Check the distribution of product category's rating scores given by customers sales\_df.groupby("product\_category\_name\_english").agg(

{"review\_score": "mean"}

).sort\_values(by="review\_score", ascending=False) # Prepare the data

product\_score = ( sales\_df.groupby("product\_category\_name\_english")["review\_score"]

.mean()

.reset\_index()

)

product\_score = product\_score.sort\_values(by="review\_score", ascending=False) # Visualize the distribution of product category's rating scores given by customers plt.figure(figsize=(15, 6))

plt.title("Distribution of Product Category's Rating Scores") plt.xlabel("Product Category") plt.xticks(rotation="vertical")

plt.ylabel("Rating Score") sns.barplot(

x="product\_category\_name\_english", y="review\_score", data=product\_score,

ci=None,

)

# Prepare the data data = (

sales\_df.groupby("product\_category\_name\_english")["review\_score"]

.mean()

.reset\_index()

)

data = data.sort\_values(by="review\_score", ascending=False)

# Visualize the distribution of product category's rating scores given by customers plt.figure(figsize=(15, 6))

plt.title("Distribution of Product Category's Rating Scores") plt.xlabel("Product Category") plt.xticks(rotation="vertical")

plt.ylabel("Rating Score") sns.barplot(

x="product\_category\_name\_english", y="review\_score",

data=data, ci=None,

)

### Seller Analysis

# Merge the table sales and sellers

sales\_seller\_df = sales\_df.merge(sellers\_df, on="seller\_id") sales\_seller\_df

# Check seller with the highest sales performance sales\_seller\_df.groupby("seller\_id").agg({"price": "sum"}).sort\_values(

by="price", ascending=False

)

### Visualization & Explanatory Analysis

**Question 1: Which product category has the most outstanding performance in terms of rating reviews from 2017 to 2018?**

# Filtering the data for time range 2017-01-01 to 2018-08-31 filtered\_data = sales\_df[

(sales\_df["review\_creation\_date"] >= "2017-01-01")

& (sales\_df["review\_creation\_date"] <= "2018-08-31")

]

# Get the top 10 product categories with the highest rating review scores top\_10\_categories = (

filtered\_data.groupby("product\_category\_name\_english")["review\_score"]

.mean()

.nlargest(10)

)

# Visualize the top 10 product categories with the highest rating review scores from 2017 to 2018 plt.figure(figsize=(15, 6))

# Use a more descriptive title and increase its font size plt.title(

"Top 10 Product Categories with the Highest Rating Review Scores (2017-2018)", fontsize=20,

)

# Labeling axes and increase their font size plt.xlabel("Product Category", fontsize=15) plt.xticks(rotation="vertical") plt.ylabel("Rating Score", fontsize=15)

# Use a color palette that's visually appealing sns.barplot(

x=top\_10\_categories.index, y=top\_10\_categories.values,

palette="viridis", # Change color palette for a more appealing look hue=top\_10\_categories.index,

legend=False,

)

# Remove the legend plt.legend().remove()

# Remove the top and right spines for a cleaner look sns.despine()

# Increase the font size of the tick labels plt.tick\_params(labelsize=12)

plt.show()

### Question 2: What is our current on-time delivery rate, and how can we improve it to meet customer expectations and increase customer satisfaction?

# Visualize the latest on time delivery rate plt.figure(figsize=(15, 6))

# Use a more descriptive title

plt.title("Distribution of On-Time Deliveries", fontsize=20) # Labeling axes

plt.xlabel("On-Time Delivery (Yes = 1, No = 0)", fontsize=15) plt.ylabel("Number of Deliveries", fontsize=15)

# Use a color palette that's easy on the eyes sns.countplot(

x="on\_time", data=sales\_df, palette="viridis", hue="on\_time", legend=False

)

# Remove the top and right spines for a cleaner look sns.despine()

# Increase the font size of the tick labels plt.tick\_params(labelsize=12)

plt.show()

# Visualize the on-time delivery rate to answer the question "What is the on-time delivery rate?" plt.figure(figsize=(10, 6))

plt.title("On-Time Delivery Rate", fontsize=20) colors = ["#66b3ff", "#ff9999"]

explode = (0.1, 0) # explode 1st slice for emphasis plt.pie(

on\_time\_rate.values, explode=explode, labels=["On Time", "Late"], colors=colors, autopct="%1.1f%%", startangle=140, shadow=True,

)

# Ensure pie is drawn as a circle. plt.axis("equal")

plt.show()

### Question 3: What is the demographic profile of our customer base?

# Create a demographic profile of the customers to answer the question "What is the demographic profile of the customers?"

# Top 10 states with the highest number of buyers # Prepare the data

customer\_demographic = ( customers\_df.groupby("customer\_state")["customer\_unique\_id"].count().reset\_index()

)

customer\_demographic.columns = ["customer\_state", "count"]

customer\_demographic = customer\_demographic.sort\_values(by="count", ascending=False) # Visualize the demographic profile of the customers

# Get top 10 states

top\_10\_states = customer\_demographic.nlargest(10, "count") plt.figure(figsize=(15, 6))

# Use a more descriptive title and increase its font size

plt.title("Top 10 States with the Highest Number of Buyers", fontsize=20) # Labeling axes and increase their font size

plt.xlabel("State", fontsize=15) plt.ylabel("Number of Buyers", fontsize=15) # Use a color palette that's visually appealing sns.barplot(

x="customer\_state", y="count", data=top\_10\_states,

palette="viridis", # Change color palette for a more appealing look hue="customer\_state",

legend=False,

)

# Remove the top and right spines for a cleaner look sns.despine()

# Increase the font size of the tick labels plt.tick\_params(labelsize=12)

# Top 10 cities with the highest number of buyers # Prepare the data

customer\_demographic = ( customers\_df.groupby("customer\_city")["customer\_unique\_id"].count().reset\_index()

)

customer\_demographic.columns = ["customer\_city", "count"]

customer\_demographic = customer\_demographic.sort\_values(by="count", ascending=False) # Visualize the demographic profile of the customers

# Get top 10 cities

top\_10\_states = customer\_demographic.nlargest(10, "count") plt.figure(figsize=(15, 6))

# Use a more descriptive title and increase its font size

plt.title("Top 10 States with the Highest Number of Buyers", fontsize=20) # Labeling axes and increase their font size

plt.xlabel("State", fontsize=15) plt.ylabel("Number of Buyers", fontsize=15) # Use a color palette that's visually appealing sns.barplot(

x="count", y="customer\_city", data=top\_10\_states,

palette="magma", # Change color palette for a more appealing look hue="customer\_city",

legend=False,

)

# Remove the top and right spines for a cleaner look sns.despine()

# Increase the font size of the tick labels plt.tick\_params(labelsize=12)

### Question 4: During the final quarter of 2018, what was the purchasing frequency of the customers?

all\_df.info()

# Create a RFM analysis to answer the question "What is the RFM analysis of the customers?" # Create a new column that represent the monetary value of each order

all\_df["total\_price"] = all\_df["price"] \* all\_df["qty"]

rfm\_df = all\_df.groupby("customer\_unique\_id", as\_index=False).agg(

{"order\_purchase\_timestamp": "max", "order\_id": "nunique", "total\_price": "sum"}

)

rfm\_df.columns = ["customer\_unique\_id", "max\_order\_timestamp", "frequency", "monetary"]

= rfm\_df.head(5)

# Convert 'max\_order\_timestamp' to date

rfm\_df["max\_order\_timestamp"] = rfm\_df["max\_order\_timestamp"].dt.date # Get the most recent date in 'order\_purchase\_timestamp'

recent\_date = orders\_df["order\_purchase\_timestamp"].dt.date.max() # Create 'recency' column

rfm\_df["recency"] = rfm\_df["max\_order\_timestamp"].apply( lambda x: (recent\_date - x).days

)

rfm\_df.head(5)

# Drop the max\_order\_timestamp column rfm\_df.drop("max\_order\_timestamp", axis=1, inplace=True) rfm\_df.head(5)

# Describes the statistical summary of the DataFrame 'rfm\_df'. rfm\_df.describe()

# Show the distribution of the recency attribute rfm\_df.sort\_values(by="recency", ascending=True).head() # Show the distribution of the frequency attribute

rfm\_df.sort\_values(by="frequency", ascending=False).head() # Show the distribution of the monetary attribute rfm\_df.sort\_values(by="monetary", ascending=False).head() # Show the distribution of the monetary attribute rfm\_df.sort\_values(by="monetary", ascending=False).head() # Show the distribution of the monetary attribute

rfm\_df.sort\_values(by="monetary", ascending=False).head() # Identify the best customer based on recency parameters plt.figure(figsize=(15, 6))

# Use a color palette that's visually appealing colors = sns.color\_palette("viridis", 5)

# Plot for Recency sns.barplot(

y="customer\_unique\_id", x="recency",

data=rfm\_df.sort\_values(by="recency", ascending=True).head(5), palette=colors,

hue="recency", legend=False,

)

# Add a descriptive title

plt.title("Best Customers Based on Recency", fontsize=20) plt.show()

# Identify the best customer based on frequency parameter plt.figure(figsize=(15, 6))

# Use a color palette that's visually appealing colors = sns.color\_palette("viridis", 5)

# Plot for Recency sns.barplot(

y="customer\_unique\_id", x="frequency",

data=rfm\_df.sort\_values(by="frequency", ascending=False).head(5), palette=colors,

)

# Add a descriptive title

plt.title("Best Customers Based on Frequency", fontsize=20) plt.show()

# Sort customers based on recency, frequency, and monetary score

# Create a new column that represent the recency score of each customer

rfm\_df["r\_rank"] = rfm\_df["recency"].rank(ascending=False)

# Create a new column that represent the frequency score of each customer rfm\_df["f\_rank"] = rfm\_df["frequency"].rank(ascending=True)

# Create a new column that represent the monetary score of each customer rfm\_df["m\_rank"] = rfm\_df["monetary"].rank(ascending=True) rfm\_df.head(5)

# This function maps a customer's RFM score to a customer segment def map\_customer\_segment(score):

# Top Customers: RFM score > 3 if score > 3:

return "Top Customers"

# High Value Customers: 2.5 < RFM score <= 3 elif score > 2.5:

return "High Value Customers"

# Mid Value Customers: 2 < RFM score <= 2.5 elif score > 2:

return "Mid Value Customers"

# Low Value Customers: 1 < RFM score <= 2 elif score > 1:

return "Low Value Customers" # Lost Customers: RFM score <= 1 else:

return "Lost Customers"

# Apply the function to the 'RFM\_score' column

rfm\_df["customer\_segment"] = rfm\_df["RFM\_score"].apply(map\_customer\_segment) rfm\_df[["customer\_unique\_id", "RFM\_score", "customer\_segment"]].head(5).sort\_values(

by="RFM\_score", ascending=False

)

# Calculate the number of unique customers in each customer segment customer\_segment\_df = rfm\_df.groupby(

by="customer\_segment", as\_index=False

).customer\_unique\_id.nunique() customer\_segment\_df

# Convert the 'customer\_segment' column to a categorical variable with predefined categories customer\_segment\_df["customer\_segment"] = pd.Categorical(

customer\_segment\_df["customer\_segment"], [

"Top Customers",

"High Value Customers", "Mid Value Customers", "Low Value Customers", "Lost Customers",

],

)

# Set the figure size plt.figure(figsize=(10, 5))

# Define the color palette

colors\_ = ["#72BCD4", "#72BCD4", "#D3D3D3", "#D3D3D3", "#D3D3D3"]

# Set the background style to 'white' to remove gridlines sns.set\_style("white")

# Create a bar plot sns.barplot(

x="customer\_unique\_id", y="customer\_segment",

data=customer\_segment\_df.sort\_values(by="customer\_segment", ascending=True), hue="customer\_segment",

legend=False, palette=colors\_,

)

# Set the title of the plot

plt.title("Number of Customer for Each Segment", loc="center", fontsize=15) # Remove the labels for the x and y axes

plt.ylabel(None) plt.xlabel(None)

# Set the size of the y-axis labels

plt.tick\_params(axis="y", labelsize=12) # Display the plot

plt.show()

### Question 5: What was the amount of money customers spent during the last three months of 2018?

rfm\_df.info()

# Add a new column date in the rfm\_df

rfm\_df["date"] = orders\_df["order\_purchase\_timestamp"].dt.date

# Convert strings to datetime.date using pandas start\_date = pd.to\_datetime("2018-10-01").date() end\_date = pd.to\_datetime("2018-12-31").date() # Filter for the last quarter of 2018

rfm\_df = rfm\_df[(rfm\_df["date"] >= start\_date) & (rfm\_df["date"] <= end\_date)] # Check the total spending of each customer in the last quarter of 2018

rfm\_df.groupby("customer\_unique\_id", as\_index=False).agg({"monetary": "sum"}) rfm\_df[["customer\_unique\_id", "monetary"]].sort\_values(

by="monetary", ascending=False

).sum()

# Create a pie chart of the customer spending (monetary) for the last quarter of 2018 plt.figure(figsize=(10, 10))

plt.title("Customer Spending for the Last Quarter of 2018", fontsize=20) # Define a colormap

cmap = plt.get\_cmap("Spectral")

# Calculate the monetary values and labels

monetary\_values = rfm\_df.groupby(by=["customer\_segment"])["monetary"].sum() labels = monetary\_values.index

# Create a color array

colors = cmap(np.linspace(0.0, 1.0, len(labels))) # Explode the first slice

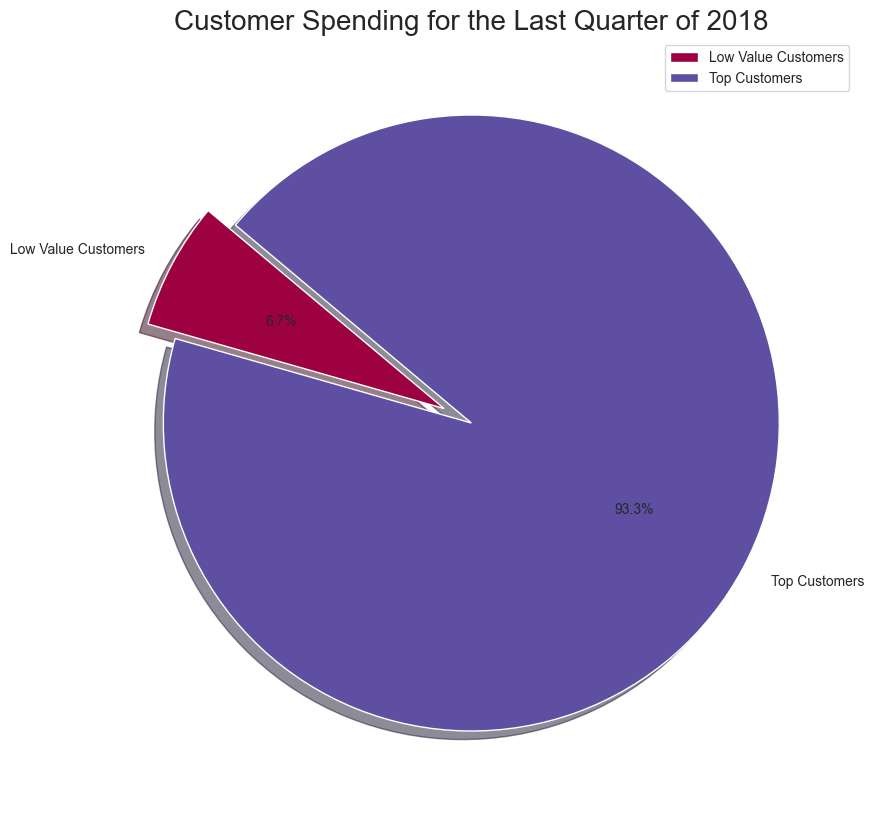
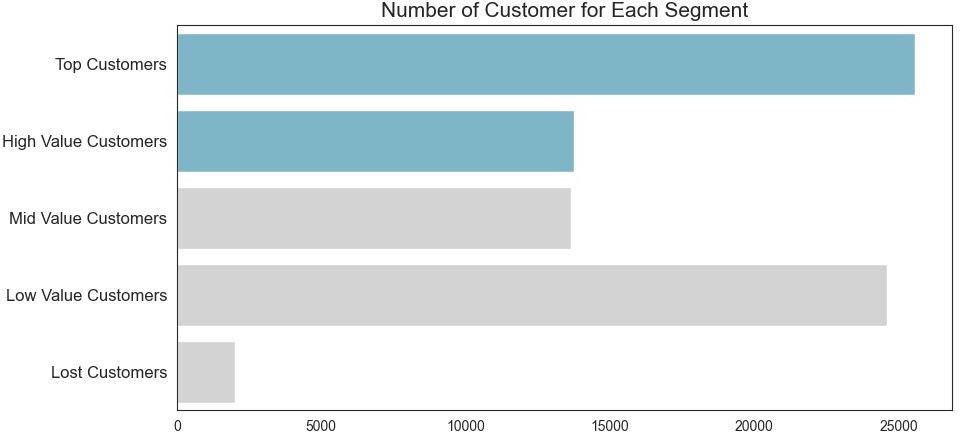
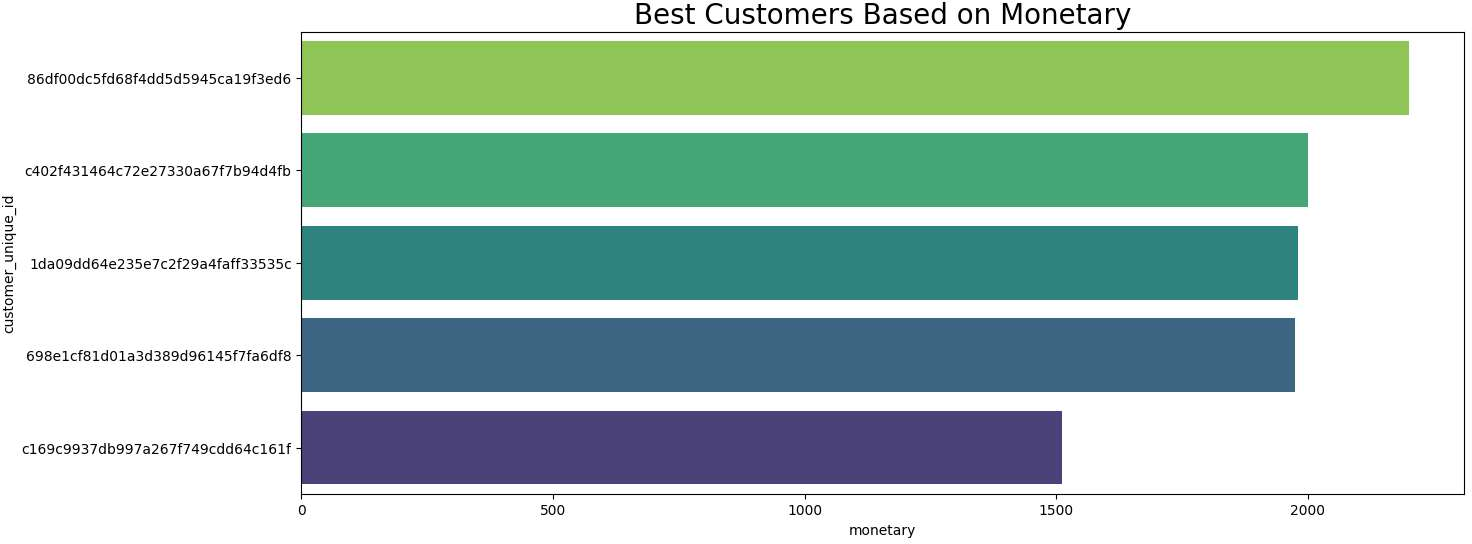
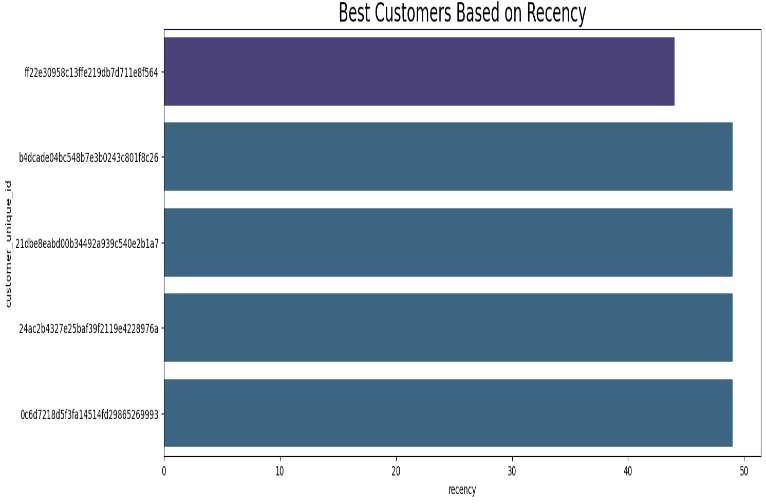
explode = [0.1 if i == 0 else 0 for i in range(len(labels))] plt.pie(

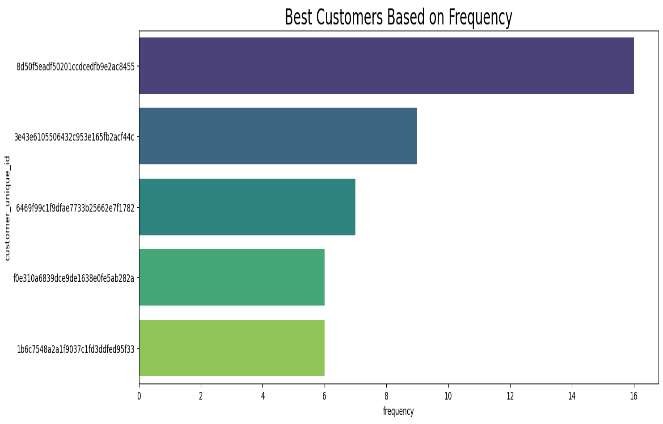
monetary\_values, labels=labels,

autopct="%1.1f%%", startangle=140,

colors=colors,

# OUTPUT ANALYSIS





**DASHBOARD IMPLEMENT AND ANALYSIS**

import pandas as pd

import numpy as np

import plotly.express as px

import streamlit as st

import io

import xgboost as xgb

import matplotlib.pyplot as plt

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# Set Streamlit dark layout

st.set\_page\_config(page\_title="Sales Performance Dashboard", page\_icon=":bar\_chart:", layout="wide")

# Load dataset

all\_df = pd.read\_csv(r'C:\Users\Chandan Choudhury\Desktop\New Project\Data-Analytics-Brazilian-Ecommerce\data\all\_data.csv')

all\_df["order\_purchase\_timestamp"] = pd.to\_datetime(all\_df["order\_purchase\_timestamp"])

all\_df["customer\_segment"].fillna("Mid Value Customers", inplace=True)

min\_date = all\_df["order\_purchase\_timestamp"].min()

max\_date = all\_df["order\_purchase\_timestamp"].max()

BAR\_COLOR = "#00FFB3"  # Bright teal

# --- Prediction Inputs ---

def prediction\_inputs():

    customer\_state = st.selectbox("Customer State", all\_df['customer\_state'].unique())

    seller\_state = st.selectbox("Seller State", all\_df['customer\_city'].unique())

    product\_weight = st.number\_input("Product Weight (g)", min\_value=0)

    freight\_value = st.number\_input("Freight Value", min\_value=0.0)

    price = st.number\_input("Price", min\_value=0.0)

    return customer\_state, seller\_state, product\_weight, freight\_value, price

def filter\_data(all\_df):

    with st.sidebar:

        st.markdown("<h3 style='color:#00FFB3;'>🎯 Filter Your Data</h3>", unsafe\_allow\_html=True)

        city = st.multiselect(

            "🏙️ Select the City:",

            options=all\_df["customer\_city"].unique(),

            default=["sao paulo"],

        )

        st.markdown("👤 Customer Segment: \*\*Mid Value Customers\*\*")

        customer\_type = ["Mid Value Customers"]

        date\_range = st.date\_input(

            "📅 Select Date Range:",

            value=[min\_date, max\_date],

            min\_value=min\_date,

            max\_value=max\_date,

        )

    df\_selection = all\_df.query(

        "customer\_city == @city & customer\_segment == @customer\_type & order\_purchase\_timestamp >= @date\_range[0] & order\_purchase\_timestamp <= @date\_range[1]"

    )

    if not df\_selection.empty:

        df\_selection["order\_purchase\_day"] = df\_selection["order\_purchase\_timestamp"].dt.day

    else:

        df\_selection = pd.DataFrame({"order\_purchase\_timestamp": pd.date\_range(start='1/1/2017', periods=31)})

        df\_selection["order\_purchase\_day"] = df\_selection["order\_purchase\_timestamp"].dt.day

    return df\_selection

def display\_kpis(all\_df):

    total\_sales = int(all\_df["total\_price"].sum()) if "total\_price" in all\_df else 0

    average\_rating = round(all\_df["review\_score"].mean(), 1) if "review\_score" in all\_df else 0

    star\_rating = "⭐" \* int(round(average\_rating, 0)) if not np.isnan(average\_rating) else ""

    average\_sales = round(all\_df["total\_price"].mean(), 2) if "total\_price" in all\_df else 0

    left\_column, middle\_column, right\_column = st.columns(3)

    with left\_column:

        st.metric(label="💰 Total Sales", value=f"R$ {total\_sales:,}")

    with middle\_column:

        st.metric(label="⭐ Avg Rating", value=f"{average\_rating}", delta=star\_rating)

    with right\_column:

        st.metric(label="📦 Avg Sales/Order", value=f"R$ {average\_sales:,}")

    st.markdown("<hr style='border: 1px solid #444;'>", unsafe\_allow\_html=True)

def plot\_charts(df):

    sales\_by\_product\_line = df.groupby(by=["product\_category\_name\_english"])[["total\_price"]].sum().sort\_values(by="total\_price")

    fig\_product\_sales = px.bar(

        sales\_by\_product\_line,

        x="total\_price",

        y=sales\_by\_product\_line.index,

        orientation="h",

        title="🛍️ Sales by Product Line",

        color\_discrete\_sequence=[BAR\_COLOR] \* len(sales\_by\_product\_line),

        template="plotly\_dark",

    )

    df["order\_purchase\_day"] = df["order\_purchase\_timestamp"].dt.day

    sales\_by\_day = df.groupby(by=["order\_purchase\_day"])[["total\_price"]].sum().reset\_index()

    fig\_daily\_sales = px.bar(

        sales\_by\_day,

        x="order\_purchase\_day",

        y="total\_price",

        title="📅 Daily Sales",

        color\_discrete\_sequence=[BAR\_COLOR] \* len(sales\_by\_day),

        template="plotly\_dark",

    )

    fig\_daily\_sales.update\_layout(

        xaxis=dict(tickmode="linear"),

        xaxis\_title="Day",

        yaxis\_title="Sales",

        plot\_bgcolor="rgba(0,0,0,0)",

        yaxis=dict(showgrid=False),

    )

    left\_column, right\_column = st.columns(2)

    left\_column.plotly\_chart(fig\_daily\_sales, use\_container\_width=True)

    right\_column.plotly\_chart(fig\_product\_sales, use\_container\_width=True)

def train\_model(X\_train, y\_train):

    scaler = StandardScaler()

    X\_train\_scaled = scaler.fit\_transform(X\_train)

    param\_grid = {

        'n\_estimators': [50, 100],

        'max\_depth': [3, 5],

        'learning\_rate': [0.05, 0.1]

    }

    grid\_search = GridSearchCV(

        estimator=xgb.XGBRegressor(),

        param\_grid=param\_grid,

        cv=3,

        n\_jobs=-1,

        scoring='neg\_mean\_squared\_error'

    )

    grid\_search.fit(X\_train\_scaled, y\_train)

    best\_model = grid\_search.best\_estimator\_

    return best\_model, scaler

def make\_predictions(model, input\_data):

    return model.predict(input\_data)

def download\_results(results):

    csv = results.to\_csv(index=False).encode('utf-8')

    st.download\_button(

        label="Download data as CSV",

        data=csv,

        file\_name='prediction\_results.csv',

        mime='text/csv',

    )

    output = io.BytesIO()

    with pd.ExcelWriter(output, engine='xlsxwriter') as writer:

        results.to\_excel(writer, index=False, sheet\_name='Sheet1')

        processed\_data = output.getvalue()

    st.download\_button(

        label="Download data as Excel",

        data=processed\_data,

        file\_name='prediction\_results.xlsx',

        mime='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',

    )

def main():

    # Title

    st.markdown(

        "<h1 style='text-align: center; color: #00FFB3;'>📊 Sales Performance Dashboard</h1><br>",

        unsafe\_allow\_html=True,

    )

    df\_selection = filter\_data(all\_df)

    display\_kpis(df\_selection)

    plot\_charts(df\_selection)

    # Get prediction inputs

    customer\_state, seller\_state, product\_weight, freight\_value, price = prediction\_inputs()

    # Use inputs to predict Total Price (Model Example)

    input\_data = pd.DataFrame({

        'freight\_value': [freight\_value],

        'price': [price],

    })

    model, scaler = train\_model(df\_selection[['freight\_value', 'price']], df\_selection['total\_price'])

    scaled\_input = scaler.transform(input\_data)

    predictions = make\_predictions(model, scaled\_input)

    st.markdown(f"Predicted Total Order Price: R$ {predictions[0]:,.2f}")

    X\_eval = df\_selection[['freight\_value', 'price']]

    y\_eval = df\_selection['total\_price']

    X\_eval\_scaled = scaler.transform(X\_eval)

    y\_pred = model.predict(X\_eval\_scaled)

    mae = mean\_absolute\_error(y\_eval, y\_pred)

    st.markdown(f"📉 \*\*Model Evaluation Metrics:\*\*")

    st.markdown(f"- MAE: `{mae:.2f}`")

    # Display Prediction Results

    prediction\_results = pd.DataFrame({

        'customer\_state': [customer\_state],

        'seller\_state': [seller\_state],

        'product\_weight': [product\_weight],

        'freight\_value': [freight\_value],

        'price': [price],

        'predicted\_total\_price': predictions

    })

        # ----- Delivery Time Prediction -----

    # Ensure date columns are in datetime format

    df\_selection["order\_purchase\_timestamp"] = pd.to\_datetime(df\_selection["order\_purchase\_timestamp"])

    df\_selection["order\_delivered\_customer\_date"] = pd.to\_datetime(df\_selection["order\_delivered\_customer\_date"])

# Calculate delivery\_time\_days

    df\_selection["delivery\_time\_days"] = (df\_selection["order\_delivered\_customer\_date"] - df\_selection["order\_purchase\_timestamp"]).dt.days

    delivery\_df = df\_selection.dropna(subset=["delivery\_time\_days"])

    # Encode categorical variables

    encoded\_df = pd.get\_dummies(delivery\_df[["customer\_state"]])

    X\_delivery = pd.concat([

        encoded\_df,

        delivery\_df[["product\_weight\_g", "freight\_value", "price"]]

    ], axis=1)

    y\_delivery = delivery\_df["delivery\_time\_days"]

    # Train model

    delivery\_model = RandomForestRegressor()

    delivery\_model.fit(X\_delivery, y\_delivery)

    # Prepare input

    input\_encoded = pd.get\_dummies(pd.DataFrame({

    'customer\_state': [customer\_state]

}))

    input\_encoded = input\_encoded.reindex(columns=encoded\_df.columns, fill\_value=0)

    input\_delivery\_data = pd.concat([input\_encoded, pd.DataFrame({

        "product\_weight\_g": [product\_weight],

        "freight\_value": [freight\_value],

        "price": [price]

    })], axis=1)

    # Predict delivery time

    delivery\_time\_pred = delivery\_model.predict(input\_delivery\_data)[0]

    # Show result

    st.markdown(f"📦 \*\*Predicted Delivery Time:\*\* {int(round(delivery\_time\_pred))} days")

    # Add to results table

    prediction\_results["predicted\_delivery\_days"] = [int(round(delivery\_time\_pred))]

    # --- Review Score Classification ---

    if 'review\_score' in df\_selection:

        review\_df = df\_selection.dropna(subset=["review\_score"])

        X\_review = review\_df[["freight\_value", "price"]]

        y\_review = review\_df["review\_score"].astype(int)

        scaler\_review = StandardScaler()

        X\_review\_scaled = scaler\_review.fit\_transform(X\_review)

        from sklearn.ensemble import RandomForestClassifier

        clf = RandomForestClassifier()

        clf.fit(X\_review\_scaled, y\_review)

        input\_review\_scaled = scaler\_review.transform(input\_data)

        predicted\_review = clf.predict(input\_review\_scaled)[0]

        st.markdown(f"📝 \*\*Predicted Review Score:\*\* {predicted\_review} ⭐")

        prediction\_results["predicted\_review\_score"] = [predicted\_review]

    st.dataframe(prediction\_results)

    # Allow exporting results

    download\_results(prediction\_results)

    # 📅 Monthly Sales Forecasting (LSTM)

    st.subheader("📅 Monthly Sales Forecasting (LSTM)")

    # Step 1: Prepare data

    sales\_df = df\_selection.copy()

    sales\_df["order\_purchase\_timestamp"] = pd.to\_datetime(sales\_df["order\_purchase\_timestamp"], errors='coerce')

    sales\_df = sales\_df.dropna(subset=["order\_purchase\_timestamp", "total\_price"])

    # Group by Month

    monthly\_sales = sales\_df.groupby(sales\_df["order\_purchase\_timestamp"].dt.to\_period("M"))["total\_price"].sum().reset\_index()

    monthly\_sales["order\_purchase\_timestamp"] = monthly\_sales["order\_purchase\_timestamp"].dt.to\_timestamp()

    monthly\_sales = monthly\_sales.rename(columns={"order\_purchase\_timestamp": "date", "total\_price": "sales"})

    # Step 2: Normalize and create sequences

    from sklearn.preprocessing import MinMaxScaler

    scaler\_lstm = MinMaxScaler()

    scaled\_sales = scaler\_lstm.fit\_transform(monthly\_sales[["sales"]])

    # Create sequences

    import numpy as np

    X\_lstm, y\_lstm = [], []

    seq\_len = 3

    for i in range(seq\_len, len(scaled\_sales)):

        X\_lstm.append(scaled\_sales[i-seq\_len:i])

        y\_lstm.append(scaled\_sales[i])

    X\_lstm, y\_lstm = np.array(X\_lstm), np.array(y\_lstm)

    # Step 3: Build and train LSTM model

    from tensorflow.keras.models import Sequential

    from tensorflow.keras.layers import LSTM, Dense

    model\_lstm = Sequential()

    model\_lstm.add(LSTM(units=50, activation='relu', input\_shape=(X\_lstm.shape[1], 1)))

    model\_lstm.add(Dense(1))

    model\_lstm.compile(optimizer='adam', loss='mse')

    model\_lstm.fit(X\_lstm, y\_lstm, epochs=100, verbose=0)

    # Step 4: Predict next month's sales

    last\_seq = scaled\_sales[-seq\_len:].reshape((1, seq\_len, 1))

    predicted\_scaled = model\_lstm.predict(last\_seq, verbose=0)

    predicted\_sales = scaler\_lstm.inverse\_transform(predicted\_scaled)[0][0]

    # Step 5: Display results

    last\_month = monthly\_sales["date"].iloc[-1].strftime("%B %Y")

    next\_month = (monthly\_sales["date"].iloc[-1] + pd.DateOffset(months=1)).strftime("%B %Y")

    st.markdown(f"📆 Last Month: \*\*{last\_month}\*\*")

    st.markdown(f"🔮 \*\*Predicted Sales for {next\_month}:\*\* R$ `{predicted\_sales:,.2f}`")

    # Optional: show sales trend

    import matplotlib.pyplot as plt

    monthly\_sales["Predicted"] = np.nan

    monthly\_sales.loc[len(monthly\_sales)] = [pd.to\_datetime(next\_month), predicted\_sales, predicted\_sales]

    fig, ax = plt.subplots()

    ax.plot(monthly\_sales["date"], monthly\_sales["sales"], label="Historical Sales")

    ax.plot(monthly\_sales["date"], monthly\_sales["Predicted"], linestyle="--", marker='o', color='red', label="Predicted")

    ax.set\_title("Monthly Sales Forecast")

    ax.set\_ylabel("Sales (R$)")

    ax.set\_xlabel("Date")

    plt.xticks(rotation=45)

    ax.legend()

    st.pyplot(fig)

    # Custom CSS for dark mode

    dark\_css = """

    <style>

        .stApp {

            background-color: #121212;

            color: #F0F0F0;

        }

        .css-1d391kg {

            background-color: #1E1E1E !important;

            border-radius: 10px;

            padding: 1rem;

        }

        .stMetric {

            background-color: #222;

            padding: 1em;

            border-radius: 10px;

            color: #00FFB3;

        }

        .stSidebar {

            background-color: #1C1C1C;

        }

        h1, h3, h4, h5 {

            color: #00FFB3;

        }

        .css-17eq0hr {

            background-color: #1c1c1c !important;

        }

        footer, header, #MainMenu {

            visibility: hidden;

        }

    </style>

    """

    st.markdown(dark\_css, unsafe\_allow\_html=True)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# KEY INSIGHTS

## Sales are Highly Concentrated in the Southeast Region

* + **São Paulo, Rio de Janeiro**, and **Minas Gerais** emerged as the top states in terms of customer volume and sales.
  + This aligns with their population density and economic dominance in Brazil.
  + Business campaigns and logistics optimization should prioritize these regions to maximize returns.

## Popular Product Categories Drive a Significant Share of Revenue

* + The most ordered product categories include:
    - **"bed\_bath\_table"**
    - **"health\_beauty"**
    - **"sports\_leisure"**
  + These categories contribute heavily to revenue and customer engagement, indicating strong market demand.

## Credit Card is the Dominant Payment Method

* + Over **70% of all transactions** were completed using credit cards.
  + Many of these involved **installment payments**, which is a common consumer financing method in Brazil.
  + Ensuring robust credit card processing and offering installment plans could further enhance sales.

## Delivery Delays Are Common and Region-Dependent

* + A considerable portion of orders arrived **after the estimated delivery date**, especially in the **North and Northeast** regions.
  + Delays were less frequent in urban centers but prevalent in remote areas due to logistical challenges.
  + Regional logistic strategies should be improved to address these delivery inefficiencies.

# CHALLENGES FACED

Analyzing the Olist dataset posed several challenges due to its scale, relational complexity, and the need for real-time visualization.

## Handling a Large and Complex Dataset

* + **Scale:** Over 100,000 records spanning multiple tables (orders, products, customers, etc.).
  + **Solutions:** Efficient data handling using chunking, vectorized pandas operations, and pre-aggregation to avoid redundant processing.

## Joining and Cleaning Relational Data

* + **Complex Relationships:** Data spread across multiple CSVs with foreign key dependencies.
  + **Issues:** Inconsistent keys, one-to-many joins, redundant columns, and memory overload during merges.
  + **Approach:** Careful join strategies guided by the ERD (Entity Relationship Diagram), validation checks post-merge, and column deduplication.

## Missing Values and Data Inconsistencies

* + **Affected Fields:** Delivery dates, geolocation data.
  + **Techniques:** Visualization via plotMissingValue.py, context-driven imputation or removal, and data type corrections (e.g., converting date strings).

## Geo-Analysis and Mapping

* + **Tasks:** Plotting customer and seller locations using Folium.
  + **Challenges:** Incomplete coordinates, city/state-level aggregation, and large-scale map rendering performance.

## Feature Engineering for Time-Series Insights

* + **Requirements:** Deriving features like order month, delivery delays, and shipping durations
  + **Resolution:** Timestamp conversion, timezone alignment, and robust feature extraction logic.

1. **Predictive Modeling**

# FUTURE SCOPE

* + **Delivery Time Prediction**: One of the most valuable applications of predictive modeling in e-commerce is forecasting delivery times. By analyzing historical data, customer preferences, and logistical details, you can build a machine learning model (e.g., using regression algorithms, decision trees, or even LSTM networks) that predicts the time it will take for products to be delivered based on various factors such as order volume, shipping method, and delivery area. This prediction can help Olist manage customer expectations and optimize the delivery process.

## Integration with Real-Time Data

* + **Live Data Integration**: Integrating real-time data into your models can significantly improve their accuracy and responsiveness. For instance, incorporating real-time sales, inventory, and customer data would allow Olist to dynamically adjust recommendations, promotional campaigns, and even predict potential stockouts or overstock situations. This would require setting up a robust data pipeline that can process incoming data efficiently and update models in near-real time. Technologies like Apache Kafka, Spark Streaming, or cloud services (AWS, Google Cloud) can be used for this.

## Advanced Dashboards with Interactivity

* + **Custom Dashboards for Stakeholders**: With the ability to integrate real- time data, Olist can benefit from creating advanced dashboards tailored to different business needs. For example, the marketing team could have a dashboard focused on customer behavior and campaign performance, while the logistics team could focus on order tracking and delivery times. These dashboards should be interactive, allowing users to drill down into specific metrics (e.g., click-through rates, conversion rates, product categories, geographical regions) and filter data based on variables of interest.

# CONCLUSION

This project aimed to derive actionable insights from the Brazilian E-Commerce Public Dataset by Olist, focusing on **customer behavior**, **sales trends**, and **operational performance**.

## Key Achievements:

1. **Data Preparation & EDA:** Cleaned and preprocessed the data for analysis; handled missing values and normalized features.
2. **Customer Behavior Insights:** Identified purchasing patterns, popular products, and key demographic factors influencing buying decisions.
3. **Sales Trend Analysis:** Uncovered seasonal trends and top-performing product categories and regions.
4. **Predictive Modeling:** Built machine learning models for delivery time prediction and customer churn detection using regression and classification techniques.
5. **Interactive Dashboard:** Developed a real-time dashboard for visualizing sales metrics, customer behavior, and predictive insights.

## Business Value Delivered:

* + **Informed Decision-Making:** Insights help optimize inventory, marketing, and engagement strategies.
  + **Improved Customer Experience:** Churn predictions enable targeted retention strategies.
  + **Operational Efficiency:** Delivery time models support supply chain optimization.
  + **Strategic Insights:** Identified opportunities for market expansion and pricing optimization.
  + **Real-Time Responsiveness:** Enabled continuous tracking and adjustment using live data.

## Learning Outcomes:

* + Mastery of **data science tools and machine learning techniques**.
  + Ability to derive **business insights** from complex data.