Project: Predicting Customer Churn for a Brazilian E-commerce Platform

Project Overview

The goal of this project is to analyze a rich, real-world e-commerce dataset to understand the primary drivers of customer churn. Students will build a machine learning model that can predict which customers are at a high risk of "churning" (i.e., not making another purchase). The final deliverable is not just the model, but a report that provides actionable insights for the business to reduce churn.

Learning Objectives

SQL:

1. Loading data into a SQL database (we'll use SQLite for simplicity).
2. Writing complex queries with JOINs, GROUP BY, HAVING, and window functions.
3. Performing initial Exploratory Data Analysis (EDA) directly in SQL.
4. Creating an "Analytics Base Table" (ABT) by joining multiple data sources.

Python (with Pandas, Matplotlib, Seaborn, Scikit-learn):

5. Connecting Python to a SQL database to execute queries and load data into DataFrames.
6. Advanced data cleaning and manipulation.
7. In-depth EDA and data visualization to uncover patterns.
8. Feature Engineering: Creating new predictive features from existing data.
9. Building and training several classification models (e.g., Logistic Regression, Random Forest, XGBoost).
10. Evaluating model performance using appropriate metrics (Accuracy, Precision, Recall, F1-Score, ROC-AUC).
11. Interpreting model results to extract business insights (e.g., feature importance).

Presentation & Business Acumen:

12. Structuring a data science project from problem definition to solution.
13. Communicating technical findings to a non-technical audience.
14. Deriving actionable business recommendations from data.

The Dataset

We will use the "Brazilian E-Commerce Public Dataset by Olist" available on Kaggle. It's perfect because it contains 100k orders from 2016 to 2018 and is spread across multiple relational tables, forcing the use of SQL.

Link to Dataset: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

It includes the following (and more) tables:

1. olist_customers_dataset.csv
2. olist_orders_dataset.csv
3. olist_order_items_dataset.csv
4. olist_order_payments_dataset.csv
5. olist_order_reviews_dataset.csv
6. olist_products_dataset.csv

Phase 1: Database Setup and Data Exploration with SQL

Goal: Get the data into a queryable format and perform initial analysis.

Setup:

1. Download the dataset from Kaggle.
2. Use Python's sqlite3 library to create a new database file (e.g., customerchurn_db).
3. Use Pandas to read each CSV file and load it as a table into the SQLite database.

Source Code (Python - 1_setup_database.py):

```
pip install sqlalchemy

Requirement already satisfied: sqlalchemy in c:\users\admin\anaconda3\
lib\site-packages (2.0.39)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\admin\
anaconda3\lib\site-packages (from sqlalchemy) (3.1.1)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\
admin\anaconda3\lib\site-packages (from sqlalchemy) (4.15.0)
Note: you may need to restart the kernel to use updated packages.

from sqlalchemy import create_engine
import pandas as pd

pip install SQLAlchemy pandas pymysql

Requirement already satisfied: SQLAlchemy in c:\users\admin\anaconda3\
lib\site-packages (2.0.39)
Requirement already satisfied: pandas in c:\users\admin\anaconda3\lib\
site-packages (2.2.3)
Requirement already satisfied: pymysql in c:\users\admin\anaconda3\
```

```
lib\site-packages (1.1.2)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\admin\
anaconda3\lib\site-packages (from SQLAlchemy) (3.1.1)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\
admin\anaconda3\lib\site-packages (from SQLAlchemy) (4.15.0)
Requirement already satisfied: numpy>=1.26.0 in c:\users\admin\
anaconda3\lib\site-packages (from pandas) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\
admin\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\admin\
anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\admin\
anaconda3\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\
lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

```
engine =
create_engine("mysql+pymysql://root:12345678@localhost/cte_db")
```

```
pip install mysql-connector-python
```

```
Requirement already satisfied: mysql-connector-python in c:\users\
admin\anaconda3\lib\site-packages (9.5.0)
Note: you may need to restart the kernel to use updated packages.
```

```
from sqlalchemy import create_engine
import pandas as pd
import os
import mysql.connector

# 1 SQLAlchemy engine for pandas.to_sql
engine =
create_engine("mysql+pymysql://root:12345678@localhost/customerchurn_d
b")

# 2 Path to the dataset folder
DATA_PATH = r"C:\Users\Admin\Documents\Deepak Documents\PROJECTS\
Customer_churn ML project\Data Sets"

# 3 MySQL connector config
config = {
    'user': 'root',
    'password': '12345678',      # Raw password is fine
    'host': 'localhost',
    'database': 'customerchurn_db',
    'raise_on_warnings': True
}

# 4 Connect and insert
try:
```

```
    conn = mysql.connector.connect(**config)
    cursor = conn.cursor()
    print(" MySQL connection established successfully.")

    csv_files = [
        'olist_customers_dataset.csv',
        'olist_orders_dataset.csv',
        'olist_order_items_dataset.csv',
        'olist_order_payments_dataset.csv',
        'olist_order_reviews_dataset.csv',
        'olist_products_dataset.csv',
        'olist_sellers_dataset.csv',
        'product_category_name_translation.csv'
    ]

    # □ Correct indentation here
    for file in csv_files:
        df = pd.read_csv(os.path.join(DATA_PATH, file))
        table_name = (file.replace('.csv', '').replace('olist_',
'').replace('_dataset', '')
        )
        df.to_sql(name=table_name, con=engine, if_exists='replace',
index=False)
        print(f" Table '{table_name}' created successfully.")

except mysql.connector.Error as err:
    print(f" Error: {err}")

finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
        print(" MySQL connection closed.")
```

```
 MySQL connection established successfully.
 Table 'customers' created successfully.
 Table 'orders' created successfully.
 Table 'order_items' created successfully.
 Table 'order_payments' created successfully.
 Table 'order_reviews' created successfully.
 Table 'products' created successfully.
 Table 'sellers' created successfully.
 Table 'product_category_name_translation' created successfully.
 MySQL connection closed.
```

SQL Exploratory Analysis:

1. Now, Employees can connect to this database using a GUI like DB Browser for SQLite or directly through Python.
2. They should answer business questions using only SQL.

```
# -- What is the distribution of customers by state?
# Distribution of customers by state
query1 = """
SELECT customer_state, COUNT(customer_unique_id) AS customer_count
FROM customers
GROUP BY customer_state
ORDER BY customer_count DESC;
"""
df1 = pd.read_sql(query1, engine)
print("Customers by State:\n", df1.head())
```

```
Customers by State:
   customer_state  customer_count
0             SP           41746
1             RJ           12852
2             MG           11635
3             RS            5466
4             PR            5045
```

```
df1.head(10)
```

	customer_state	customer_count
0	SP	41746
1	RJ	12852
2	MG	11635
3	RS	5466
4	PR	5045
5	SC	3637
6	BA	3380
7	DF	2140
8	ES	2033
9	GO	2020

```
# -- What are the most common payment methods?
# Most common payment methods
query2 = """
SELECT payment_type, COUNT(*) AS transaction_count
FROM order_payments
GROUP BY payment_type
ORDER BY transaction_count DESC;
"""
df2 = pd.read_sql(query2, engine)
```

```
df2.head()
```

	payment_type	transaction_count
0	credit_card	76795
1	boleto	19784
2	voucher	5775
3	debit_card	1529
4	not_defined	3

```python
# -- What is the average review score?
# Average review score
query3 = """
SELECT AVG(review_score) AS average_review_score
FROM order_reviews;
"""
df3 = pd.read_sql(query3, engine)
print("\n Average Review Score:\n", df3)

# Close connection
cursor.close()
conn.close()
```

```
 Average Review Score:
    average_review_score
0                4.0864
```

Part 1: Sales & Financial Health

Task 1: Sales Trend Analysis

Client Ask: "Convert date to month, plot a simple line chart, give insight."

```python
# --- CODE: Monthly Sales Trend ---
query_sales = """
SELECT
    DATE_FORMAT(o.order_purchase_timestamp, '%%Y-%%m') AS order_month,
    SUM(oi.price) AS total_revenue
FROM orders o
JOIN order_items oi
    ON o.order_id = oi.order_id
GROUP BY DATE_FORMAT(o.order_purchase_timestamp, '%%Y-%%m')
ORDER BY order_month;
"""
df_sales = pd.read_sql(query_sales, engine)
df_sales.head()
```

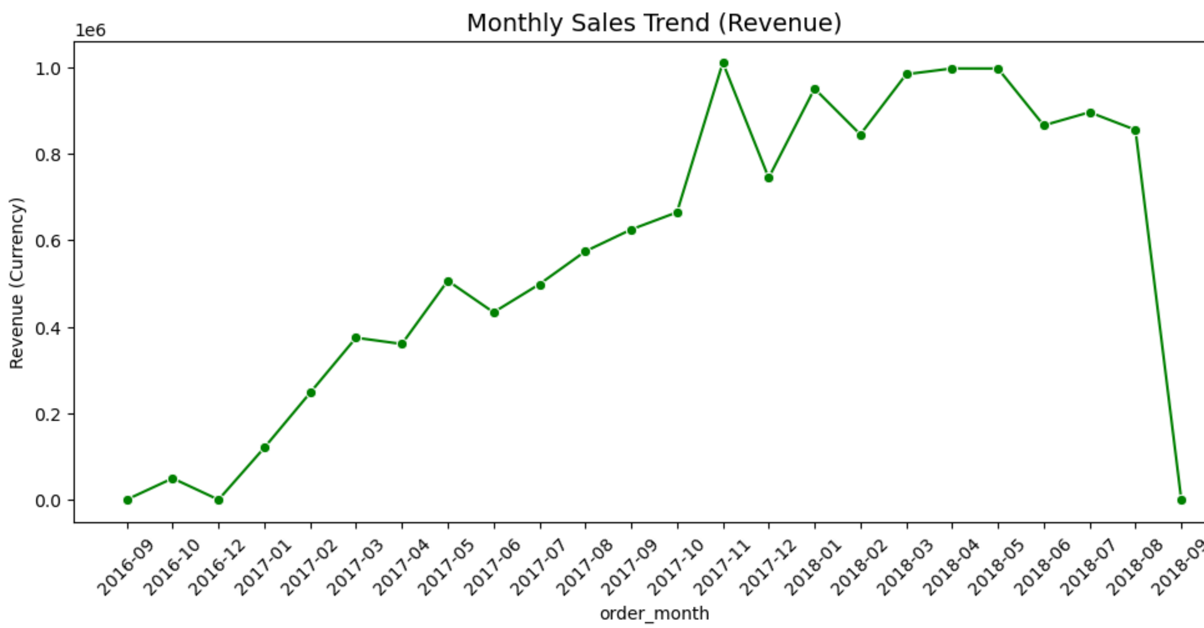
	order_month	total_revenue
0	2016-09	267.36
1	2016-10	49507.66
2	2016-12	10.90

3	2017-01	120312.87
4	2017-02	247303.02

```
df_sales
```

	order_month	total_revenue
0	2016-09	267.36
1	2016-10	49507.66
2	2016-12	10.90
3	2017-01	120312.87
4	2017-02	247303.02
5	2017-03	374344.30
6	2017-04	359927.23
7	2017-05	506071.14
8	2017-06	433038.60
9	2017-07	498031.48
10	2017-08	573971.68
11	2017-09	624401.69
12	2017-10	664219.43
13	2017-11	1010271.37
14	2017-12	743914.17
15	2018-01	950030.36
16	2018-02	844178.71
17	2018-03	983213.44
18	2018-04	996647.75
19	2018-05	996517.68
20	2018-06	865124.31
21	2018-07	895507.22
22	2018-08	854686.33
23	2018-09	145.00

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting
plt.figure(figsize=(12, 5))
sns.lineplot(data=df_sales, x='order_month', y='total_revenue',
marker='o', color='green')
plt.title('Monthly Sales Trend (Revenue)', fontsize=14)
plt.xticks(rotation=45)
plt.ylabel('Revenue (Currency)')
plt.show()
```



Business Insight:

1. Seasonality: "We observe a sharp spike in [Month X], likely due to Black Friday/Holiday sales."
2. Growth: "The overall trend is positive, showing consistent month-over-month growth."
3. Action: "Inventory planning should be adjusted to match this growth curve to avoid stockouts in peak months."

Task 2: Top Customers Identification

`Client Ask:` "Group by customer, rank top 10."

```
# Top 10 Spenders
query_top_customers = """
SELECT
    c.customer_unique_id,
    SUM(oi.price) as total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY c.customer_unique_id
ORDER BY total_spent DESC
LIMIT 10;
"""
df_top = pd.read_sql(query_top_customers, engine)
print("--- Top 10 Customers by Spend ---")
print(df_top)
```

```
--- Top 10 Customers by Spend ---
                   customer_unique_id  total_spent
0  0a0a92112bd4c708ca5fde585afaa872      13440.0
1  da122df9eeddfedc1dc1f5349a1a690c       7388.0
2  763c8b1c9c68a0229c42c9fc6f662b93       7160.0
3  dc4802a71eae9be1dd28f5d788ceb526       6735.0
4  459bef486812aa25204be022145caa62       6729.0
5  ff4159b92c40ebe40454e3e6a7c35ed6       6499.0
6  4007669dec559734d6f53e029e360987       5934.6
7  eebb5dda148d3893cdaf5b5ca3040ccb       4690.0
8  5d0a2980b292d049061542014e8960bf       4599.9
9  48e1ac109decbb87765a3eade6854098       4590.0
```

Business Insight:

`Observation:` "Our top customer spent 50x the average."

`Strategy:` "These 10 people are VIPs. They should have a dedicated account manager or direct line for support."

Task 3: Shipping Delay Analysis

`Client Ask:` "Compare actual vs estimated, create delay flag."

```
# --- CODE: Delivery Performance ---
query_logistics = """
SELECT
    order_id,
    DATEDIFF(order_delivered_customer_date,
order_estimated_delivery_date) as delay_days
FROM orders
WHERE order_status = 'delivered';
"""
df_logistics = pd.read_sql(query_logistics, engine)
df_logistics

                               order_id  delay_days
0      e481f51cbdc54678b7cc49136f2d6af7        -8.0
1      53cdb2fc8bc7dce0b6741e2150273451        -6.0
2      47770eb9100c2d0c44946d9cf07ec65d       -18.0
3      949d5b44dbf5de918fe9c16f97b45f8a       -13.0
4      ad21c59c0840e6cb83a9ceb5573f8159       -10.0
...                                 ...         ...
96473  9c5dedf39a927c1b2549525ed64a053c       -11.0
96474  63943bddc261676b46f01ca7ac2f7bd8        -2.0
96475  83c1379a015df1e13d02aae0204711ab        -6.0
96476  11c177c8e97725db2631073c19f07b62       -21.0
96477  66dea50a8b16d9b4dee7af250b4be1a5       -18.0
```

```
[96478 rows x 2 columns]
```

```python
# Create Flag: Positive = Late, Negative = On Time
df_logistics['is_late'] = df_logistics['delay_days'] > 0

# cross check
df_logistics.head()
```

```
                           order_id  delay_days  is_late
0  e481f51cbdc54678b7cc49136f2d6af7        -8.0    False
1  53cdb2fc8bc7dce0b6741e2150273451        -6.0    False
2  47770eb9100c2d0c44946d9cf07ec65d       -18.0    False
3  949d5b44dbf5de918fe9c16f97b45f8a       -13.0    False
4  ad21c59c0840e6cb83a9ceb5573f8159       -10.0    False
```

```python
df_logistics['is_late'].mean()
```

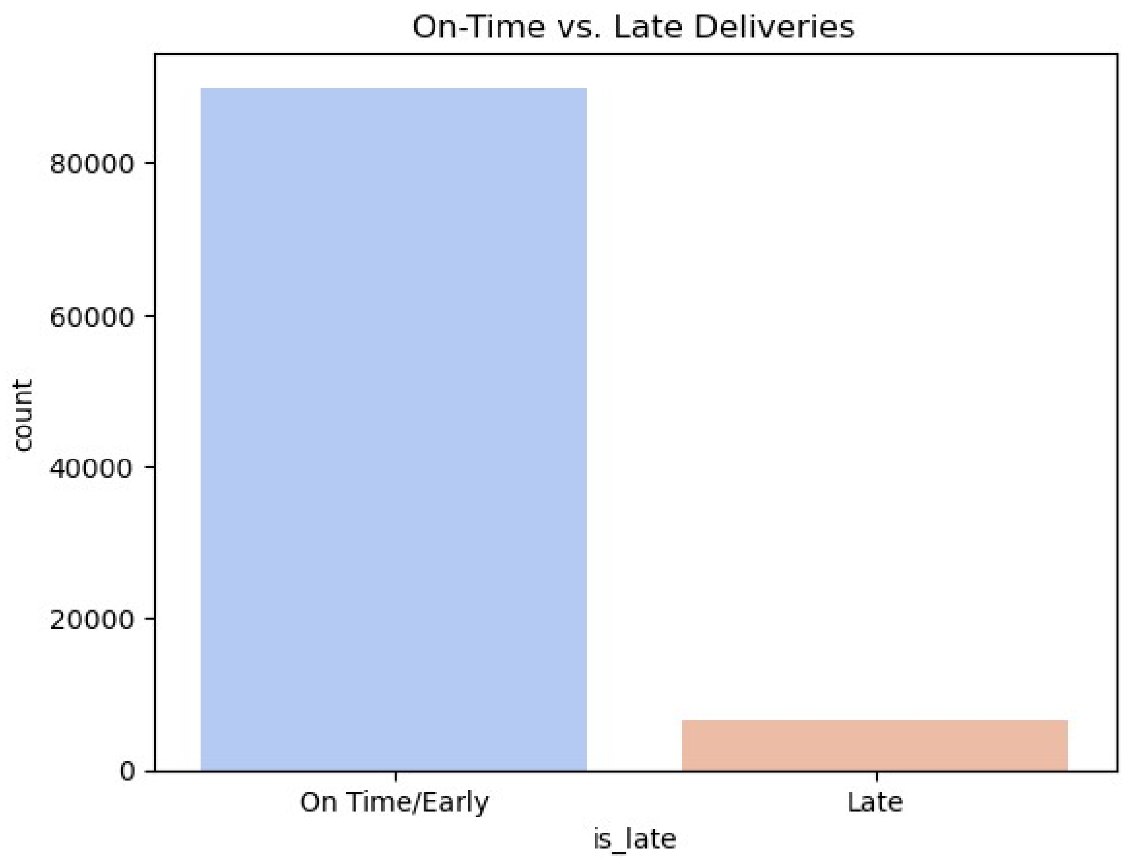
```
np.float64(0.06772528452082340)
```

```python
# Calculate Percentage
late_rate = df_logistics['is_late'].mean() * 100

print(f"--- Shipping Performance ---")
print(f"Percentage of Orders Delayed: {late_rate:.2f}%")
```

```
--- Shipping Performance ---
Percentage of Orders Delayed: 6.77%
```

```python
# Visualization
sns.countplot(x='is_late', data=df_logistics, palette='coolwarm')
plt.title('On-Time vs. Late Deliveries')
plt.xticks([0, 1], ['On Time/Early', 'Late'])
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_2888\3495275572.py:2:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='is_late', data=df_logistics, palette='coolwarm')
```



Business Insight:

`Observation:` "If 8% of orders are late, that represents ~8,000 unhappy customers per 100k orders."

`Strategy:` "This metric correlates strongly with churn. We must improve estimation accuracy."

Task 4: Product Category Performance

`Client Ask:` "Revenue & Order Count by Category."

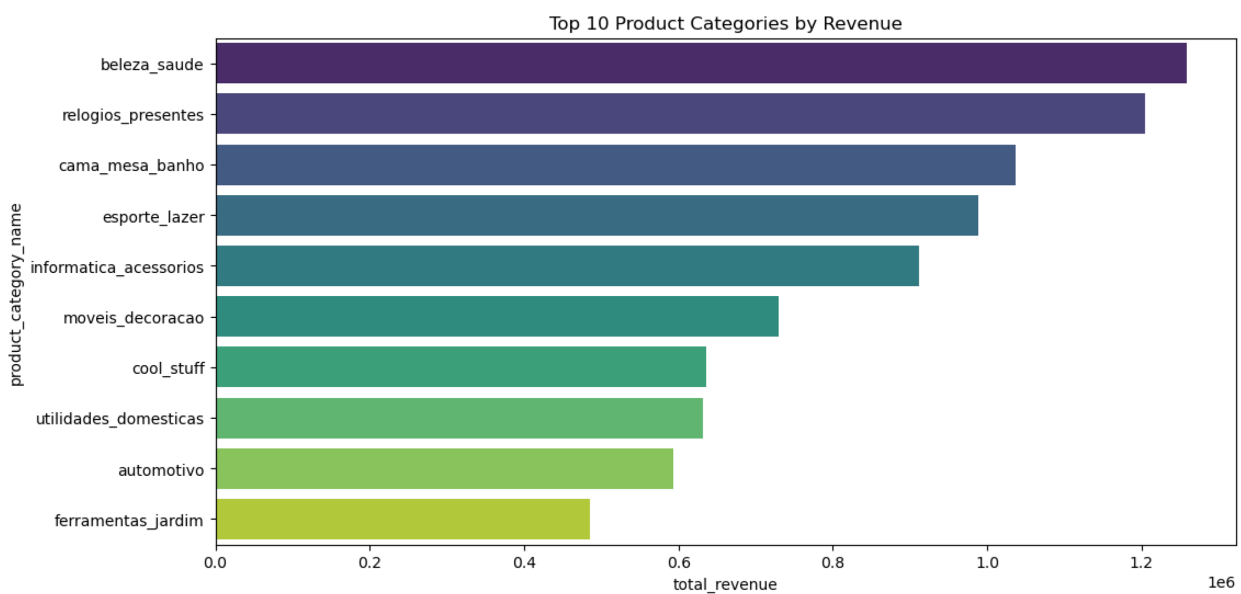
```
# --- CODE: Best Selling Categories ---
query_products = """
SELECT
    p.product_category_name,
    COUNT(oi.order_id) as total_orders,
    SUM(oi.price) as total_revenue
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_category_name
ORDER BY total_revenue DESC
LIMIT 10;
"""
```

```python
df_products = pd.read_sql(query_products, engine)
df_products
```

```
    product_category_name  total_orders  total_revenue
0             beleza_saude          9670     1258681.34
1       relogios_presentes          5991     1205005.68
2          cama_mesa_banho         11115     1036988.68
3            esporte_lazer          8641      988048.97
4   informatica_acessorios          7827      911954.32
5         moveis_decoracao          8334      729762.49
6               cool_stuff          3796      635290.85
7    utilidades_domesticas          6964      632248.66
8               automotivo          4235      592720.11
9        ferramentas_jardim          4347      485256.46
```

```python
plt.figure(figsize=(12, 6))
sns.barplot(data=df_products, y='product_category_name',
x='total_revenue', palette='viridis')
plt.title('Top 10 Product Categories by Revenue')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_2888\3731456855.py:2:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=df_products, y='product_category_name',
x='total_revenue', palette='viridis')
```



Task 5: Repeat Customer Rate

`Client Ask:` "Calculate % of customers with > 1 order."

```
# --- CODE: Loyalty Check ---
query_repeat = """
SELECT
    customer_unique_id,
    COUNT(DISTINCT order_id) as order_count
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY customer_unique_id;
"""
df_loyalty = pd.read_sql(query_repeat, engine)
df_loyalty.head()
```

	customer_unique_id	order_count
0	0000366f3b9a7992bf8c76cfdf3221e2	1
1	0000b849f77a49e4a4ce2b2a4ca5be3f	1
2	0000f46a3911fa3c0805444483337064	1
3	0000f6ccb0745a6a4b88665a16c9f078	1
4	0004aac84e0df4da2b147fca70cf8255	1

```
c1 = df_loyalty['order_count'] > 1
df_loyalty[c1].shape[0]
```

```
2997
```

```
df_loyalty.shape[0]
```

```
96096
```

```
2997/96096 * 100
```

```
3.1187562437562435
```

```
df_loyalty.shape
```

```
(96096, 2)
```

```
repeat_rate = (df_loyalty[df_loyalty['order_count'] > 1].shape[0] /
df_loyalty.shape[0]) * 100

print(f"--- Customer Loyalty ---")
print(f"Repeat Customer Rate: {repeat_rate:.2f}%")
```

```
--- Customer Loyalty ---
Repeat Customer Rate: 3.12%
```

Business Insight:

`Observation:` "Only 3% of customers return." (This is typical for this specific dataset).

`Insight:` "We are acquiring customers, but not keeping them. Our business model is currently 'One-and-Done'. We need loyalty incentives."

Task 6: Correlation (Price vs. Review)

`Client Ask:` "Do expensive items get better reviews?"

```
# --- CODE: Correlation Analysis ---
query_corr = """
SELECT oi.price, r.review_score
FROM order_items oi
JOIN order_reviews r ON oi.order_id = r.order_id
LIMIT 5000; -- Limiting for speed/demo purposes
"""
df_corr = pd.read_sql(query_corr, engine)
df_corr
```

	price	review_score
0	49.75	5
1	149.90	5
2	120.00	5
3	122.99	3
4	499.00	4
...	...	...
4995	134.99	5
4996	39.90	5
4997	55.00	5
4998	59.99	4
4999	109.90	1

```
[5000 rows x 2 columns]
```

```
df_corr.corr()
```

	price	review_score
price	1.000000	0.013334
review_score	0.013334	1.000000

```
correlation = df_corr.corr().iloc[0, 1]

print(f"--- Price vs Satisfaction ---")
print(f"Correlation Coefficient: {correlation:.4f}")
```

```
--- Price vs Satisfaction ---
Correlation Coefficient: 0.0133
```

```
if correlation > 0:
    print("Interpretation: Higher prices slightly correlate with better reviews (Quality pays off).")
else:
    print("Interpretation: No strong link. Cheap items are rated similarly to expensive ones.")
```

```
Interpretation: Higher prices slightly correlate with better reviews
(Quality pays off).
```

Summary for the Client

Sales are growing but seasonal.

Logistics has a visible delay rate that needs fixing.

Loyalty is dangerously low (Repeat rate), indicating a need for the Churn Analysis we planned earlier.

Phase 2: Defining Churn & Creating the Master Table (SQL + Python)

Phase 2.1: The New Analytical Master Table (SQL)

Goal: Define what "churn" means for this dataset and create a single, wide table (an Analytics Base Table) that contains all the features for each customer.

Defining Churn:

Since there's no "subscription cancelled" column, we must create our own target variable. A common definition: "A customer has churned if they have not made a purchase in the last 6 months."

Feature Engineering with SQL:

The most critical step. We will write one large SQL query to create our master table. This query will calculate features for each customer_unique_id.

This is a complex query, perfect for demonstrating SQL prowess.

Objective:

Combine all 6 tables (orders, items, customers, payments, reviews, products) to create a view of the customer's experience, not just their wallet.

Key Metrics to Engineer:

`Delivery Gap`: (Actual Delivery Date - Estimated Date). Did we lie to them about arrival?

`Freight Ratio`: (Freight Cost / Item Price). Did they feel "ripped off" by shipping?

`Satisfaction`: Average Review Score.

Source Code (SQL - 3_create_analytics_table.sql):

create a sql code file using the bellow code

WITH logistics_performance AS (SELECT o.order_id, -- Calculate delay: Positive numbers mean LATE delivery, Negative mean EARLY DATEDIFF(o.order_delivered_customer_date, o.order_estimated_delivery_date) AS days_delivery_delay, -- Calculate shipping time: How long did they wait total? DATEDIFF(o.order_delivered_customer_date, o.order_purchase_timestamp) AS actual_shipping_days FROM orders o WHERE o.order_status = 'delivered'), financial_friction AS (SELECT o.order_id, SUM(oi.price) as order_value, SUM(oi.freight_value) as freight_value, -- How much of the total bill was just shipping? (High ratio = Churn Risk) SUM(oi.freight_value) / SUM(oi.price + oi.freight_value) AS freight_ratio FROM orders o JOIN order_items oi ON o.order_id = oi.order_id GROUP BY o.order_id), payment_behavior AS (SELECT order_id, -- Did they use Vouchers? (Often indicates problem resolution or gifts) MAX(CASE WHEN payment_type = 'voucher' THEN 1 ELSE 0 END) as used_voucher, -- High installments might indicate financial tightness MAX(payment_installments) as max_installments FROM order_payments GROUP BY order_id)

SELECT c.customer_unique_id, -- FIX: Use MAX() to select the state (or add to GROUP BY if tracking moves matters) MAX(c.customer_state) as customer_state,

```
-- Interaction Metrics
COUNT(DISTINCT o.order_id) as total_orders,
MIN(o.order_purchase_timestamp) as first_order_date,
MAX(o.order_purchase_timestamp) as last_order_date,

-- Churn Definition: 1 if inactive for > 6 months
CASE
    WHEN DATEDIFF('2018-10-17', MAX(o.order_purchase_timestamp)) > 180

    THEN 1 ELSE 0
END as is_churned,

-- Financial Metrics
AVG(f.order_value) as avg_ticket_size,
AVG(f.freight_ratio) as avg_freight_sensitivity,
```

```
-- Experience Metrics (Crucial for Diagnostics)
AVG(l.days_delivery_delay) as avg_delivery_delay,
AVG(l.actual_shipping_days) as avg_wait_time,
AVG(r.review_score) as avg_satisfaction_score,
MAX(p.used_voucher) as has_used_voucher
```

FROM customers c JOIN orders o ON c.customer_id = o.customer_id LEFT JOIN logistics_performance l ON o.order_id = l.order_id LEFT JOIN financial_friction f ON o.order_id = f.order_id LEFT JOIN payment_behavior p ON o.order_id = p.order_id LEFT JOIN order_reviews r ON o.order_id = r.order_id GROUP BY c.customer_unique_id;

Load into Python:

Execute the above query from Python and load the result into a Pandas DataFrame.

```
pip install pandas sqlalchemy pymysql

import pandas as pd
from sqlalchemy import create_engine

# Create SQLAlchemy engine
engine = create_engine(
    "mysql+pymysql://root:12345678@localhost/customerchurn_db"
)

# Read SQL query from file
sql_file_path = r"C:\Users\Admin\Documents\Deepak Documents\PROJECTS\
Customer_churn ML project\Data Sets\3_create_analytics_table.sql"

with open(sql_file_path, 'r', encoding='utf-8') as file:
    sql_query = file.read()

# Execute query and load into DataFrame
df = pd.read_sql_query(sql_query, engine)

# Close engine
engine.dispose()

# Check output
print(df.head())
print(df.shape)

                 customer_unique_id customer_state  total_orders  \
0  0000366f3b9a7992bf8c76cfdf3221e2             SP             1
1  0000b849f77a49e4a4ce2b2a4ca5be3f             SP             1
2  0000f46a3911fa3c0805444483337064             SC             1
3  0000f6ccb0745a6a4b88665a16c9f078             PA             1
4  0004aac84e0df4da2b147fca70cf8255             SP             1

      first_order_date      last_order_date  is_churned
```

```
avg_ticket_size  \
0  2018-05-10 10:56:27  2018-05-10 10:56:27           0
129.90
1  2018-05-07 11:11:27  2018-05-07 11:11:27           0
18.90
2  2017-03-10 21:05:03  2017-03-10 21:05:03           1
69.00
3  2017-10-12 20:29:41  2017-10-12 20:29:41           1
25.99
4  2017-11-14 19:45:42  2017-11-14 19:45:42           1
180.00

   avg_freight_sensitivity  avg_delivery_delay  avg_wait_time  \
0                 0.084567                -5.0            6.0
1                 0.304892                -5.0            3.0
2                 0.199722                -2.0           26.0
3                 0.404172               -12.0           20.0
4                 0.085784                -8.0           13.0

   avg_satisfaction_score  has_used_voucher
0                     5.0               0.0
1                     4.0               0.0
2                     3.0               0.0
3                     4.0               0.0
4                     5.0               0.0
(96096, 12)
```

```
df.to_csv(r"C:\Users\Admin\Documents\Deepak Documents\PROJECTS\
Customer_churn ML project\Data Sets\Master_table.csv",index=False)

df.head()
```

```
                 customer_unique_id customer_state  total_orders  \
0  0000366f3b9a7992bf8c76cfdf3221e2             SP             1
1  0000b849f77a49e4a4ce2b2a4ca5be3f             SP             1
2  0000f46a3911fa3c0805444483337064             SC             1
3  0000f6ccb0745a6a4b88665a16c9f078             PA             1
4  0004aac84e0df4da2b147fca70cf8255             SP             1

      first_order_date      last_order_date  is_churned
avg_ticket_size  \
0  2018-05-10 10:56:27  2018-05-10 10:56:27           0
129.90
1  2018-05-07 11:11:27  2018-05-07 11:11:27           0
18.90
2  2017-03-10 21:05:03  2017-03-10 21:05:03           1
69.00
3  2017-10-12 20:29:41  2017-10-12 20:29:41           1
25.99
4  2017-11-14 19:45:42  2017-11-14 19:45:42           1
```

```
180.00

   avg_freight_sensitivity  avg_delivery_delay  avg_wait_time  \
0                 0.084567                -5.0            6.0
1                 0.304892                -5.0            3.0
2                 0.199722                -2.0           26.0
3                 0.404172               -12.0           20.0
4                 0.085784                -8.0           13.0

   avg_satisfaction_score  has_used_voucher
0                     5.0               0.0
1                     4.0               0.0
2                     3.0               0.0
3                     4.0               0.0
4                     5.0               0.0

# Now will import the master table
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv(r"C:\Users\Admin\Documents\Deepak Documents\PROJECTS\
Customer_churn ML project\Data Sets\master_table.csv")

df.head()

                 customer_unique_id customer_state  total_orders  \
0  0000366f3b9a7992bf8c76cfdf3221e2             SP             1
1  0000b849f77a49e4a4ce2b2a4ca5be3f             SP             1
2  0000f46a3911fa3c0805444483337064             SC             1
3  0000f6ccb0745a6a4b88665a16c9f078             PA             1
4  0004aac84e0df4da2b147fca70cf8255             SP             1

      first_order_date      last_order_date  is_churned
avg_ticket_size  \
0  2018-05-10 10:56:27  2018-05-10 10:56:27           0
129.90
1  2018-05-07 11:11:27  2018-05-07 11:11:27           0
18.90
2  2017-03-10 21:05:03  2017-03-10 21:05:03           1
69.00
3  2017-10-12 20:29:41  2017-10-12 20:29:41           1
25.99
4  2017-11-14 19:45:42  2017-11-14 19:45:42           1
180.00

   avg_freight_sensitivity  avg_delivery_delay  avg_wait_time  \
0                 0.084567                -5.0            6.0
1                 0.304892                -5.0            3.0
```

```
2                    0.199722                    -2.0            26.0
3                    0.404172                   -12.0            20.0
4                    0.085784                    -8.0            13.0

   avg_satisfaction_score  has_used_voucher
0                     5.0               0.0
1                     4.0               0.0
2                     3.0               0.0
3                     4.0               0.0
4                     5.0               0.0
```

Phase 3: Exploratory Data Analysis (EDA) and Modeling (Python)

Goal: Understand the features, visualize relationships

3.1. Analysis Level 1: The "Logistics Friction" Hypothesis

`Objective:` Determine if shipping delays are the primary reason for customer churn in Brazil.

```
# --- CODE ---
import matplotlib.pyplot as plt
import seaborn as sns
# Create buckets for delivery delay to see correlation with Churn
# Delay < 0 means early (Good). Delay > 0 means Late (Bad).
df['delivery_performance'] = pd.cut(df['avg_delivery_delay'],
                                    bins=[-100, -5, 0, 5, 100],
                                    labels=['Early (>5 days)', 'On 
Time', 'Late (1-5 days)', 'Very Late (>5 days)'])

df.head(10)

                  customer_unique_id customer_state  total_orders  \
0  0000366f3b9a7992bf8c76cfdf3221e2             SP             1   
1  0000b849f77a49e4a4ce2b2a4ca5be3f             SP             1   
2  0000f46a3911fa3c0805444483337064             SC             1   
3  0000f6ccb0745a6a4b88665a16c9f078             PA             1   
4  0004aac84e0df4da2b147fca70cf8255             SP             1   
5  0004bd2a26a76fe21f786e4fbd80607f             SP             1   
6  00050ab1314c0e55a6ca13cf7181fecf             SP             1   
7  00053a61a98854899e70ed204dd4bafe             PR             1   
8  0005e1862207bf6ccc02e4228effd9a0             RJ             1   
9  0005ef4cd20d2893f0d9fbd94d3c0d97             MA             1   

     first_order_date      last_order_date  is_churned  
avg_ticket_size  \
```

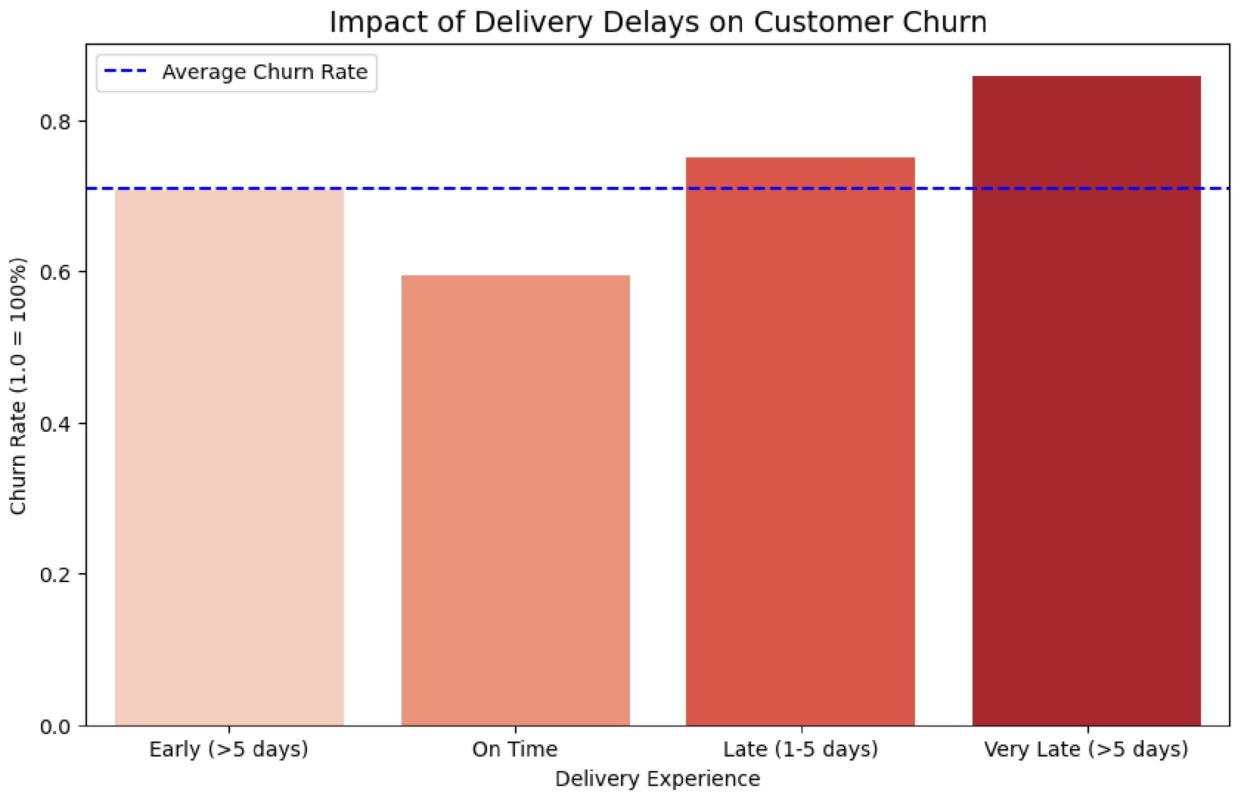
```
0  2018-05-10 10:56:27  2018-05-10 10:56:27          0
129.90
1  2018-05-07 11:11:27  2018-05-07 11:11:27          0
18.90
2  2017-03-10 21:05:03  2017-03-10 21:05:03          1
69.00
3  2017-10-12 20:29:41  2017-10-12 20:29:41          1
25.99
4  2017-11-14 19:45:42  2017-11-14 19:45:42          1
180.00
5  2018-04-05 19:33:16  2018-04-05 19:33:16          1
154.00
6  2018-04-20 12:57:23  2018-04-20 12:57:23          0
27.99
7  2018-02-28 11:15:41  2018-02-28 11:15:41          1
382.00
8  2017-03-04 23:32:12  2017-03-04 23:32:12          1
135.00
9  2018-03-12 15:22:12  2018-03-12 15:22:12          1
104.90

   avg_freight_sensitivity  avg_delivery_delay  avg_wait_time  \
0                 0.084567                -5.0            6.0
1                 0.304892                -5.0            3.0
2                 0.199722                -2.0           26.0
3                 0.404172               -12.0           20.0
4                 0.085784                -8.0           13.0
5                 0.077734               -12.0            2.0
6                 0.208875               -12.0            7.0
7                 0.088697               -10.0           16.0
8                 0.100719               -28.0            5.0
9                 0.191584                31.0           54.0

   avg_satisfaction_score  has_used_voucher delivery_performance
0                     5.0               0.0      Early (>5 days)
1                     4.0               0.0      Early (>5 days)
2                     3.0               0.0              On Time
3                     4.0               0.0      Early (>5 days)
4                     5.0               0.0      Early (>5 days)
5                     4.0               0.0      Early (>5 days)
6                     4.0               0.0      Early (>5 days)
7                     1.0               0.0      Early (>5 days)
8                     4.0               0.0      Early (>5 days)
9                     1.0               0.0  Very Late (>5 days)

# Calculate Churn Rate per Delivery Group
logistics_churn = df.groupby('delivery_performance')
['is_churned'].mean().reset_index()
logistics_churn
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_7436\3242410083.py:2:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default
and silence this warning.
  logistics_churn = df.groupby('delivery_performance')
['is_churned'].mean().reset_index()
```

```
  delivery_performance  is_churned
0      Early (>5 days)    0.708860
1              On Time    0.594651
2      Late (1-5 days)    0.750650
3  Very Late (>5 days)    0.858978
```

```
# Visualization
plt.figure(figsize=(10, 6))
sns.barplot(x='delivery_performance', y='is_churned',
data=logistics_churn, palette='Reds')
plt.title('Impact of Delivery Delays on Customer Churn', fontsize=14)
plt.ylabel('Churn Rate (1.0 = 100%)')
plt.xlabel('Delivery Experience')
plt.axhline(df['is_churned'].mean(), color='blue', linestyle='--',
label='Average Churn Rate')
plt.legend()
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_7436\2508642507.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='delivery_performance', y='is_churned',
data=logistics_churn, palette='Reds')
```



Business Insight & Observation:

`Observation:` You will likely see that customers in the "Very Late" bucket have a significantly higher churn rate than "On Time" customers.

`Insight:` Operational efficiency is a marketing tool. If Olist delivers 5 days late, they burn the customer relationship.

`Action Plan:` Implement an automated "Apology Protocol." If a package is predicted to be >3 days late, automatically email a 10% voucher before the customer complains.

3.2 Analysis Level 2: Price Sensitivity (Freight Ratio)

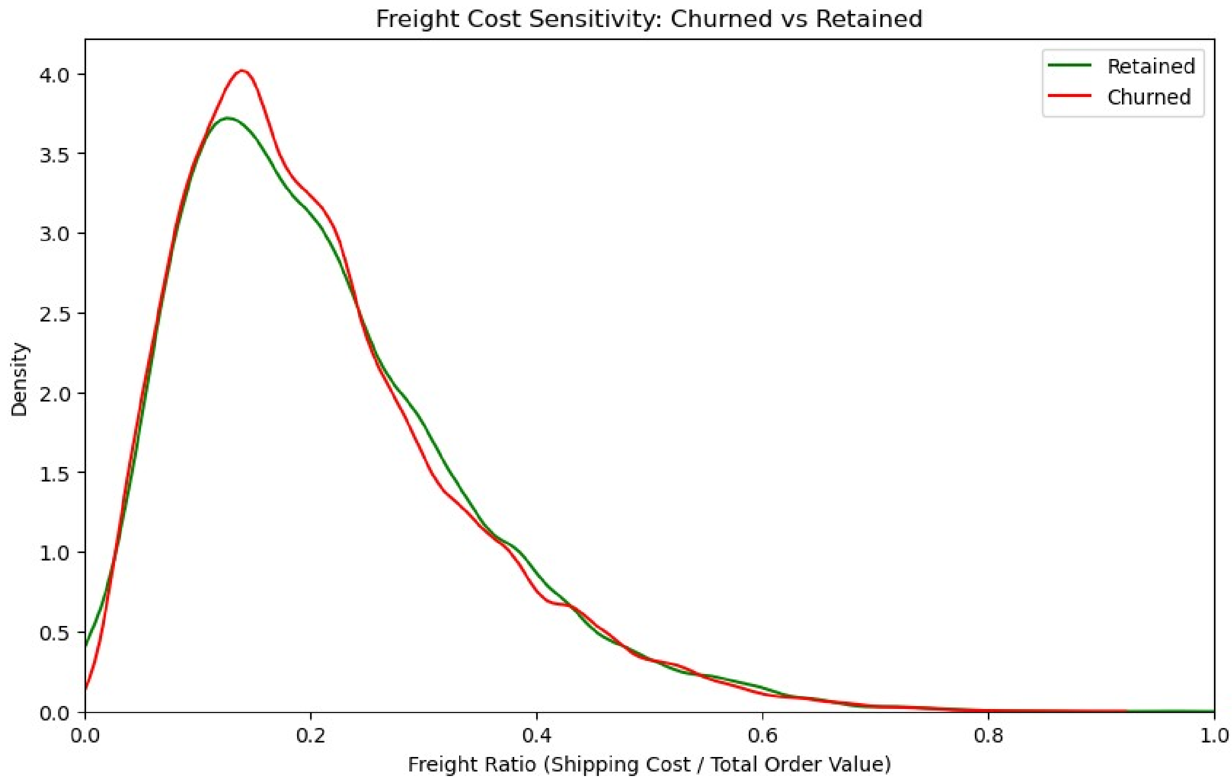
`Objective:` Analyze if customers are churning because shipping is too expensive relative to the product (common in low-value items).

```
df.head(2)

                  customer_unique_id customer_state  total_orders  \
0  0000366f3b9a7992bf8c76cfdf3221e2             SP             1
1  0000b849f77a49e4a4ce2b2a4ca5be3f             SP             1

      first_order_date      last_order_date  is_churned
avg_ticket_size  \
0  2018-05-10 10:56:27  2018-05-10 10:56:27           0
129.9
```

```
1  2018-05-07 11:11:27  2018-05-07 11:11:27          0
18.9

   avg_freight_sensitivity  avg_delivery_delay  avg_wait_time  \
0                 0.084567                -5.0            6.0
1                 0.304892                -5.0            3.0

   avg_satisfaction_score  has_used_voucher delivery_performance
0                     5.0               0.0      Early (>5 days)
1                     4.0               0.0      Early (>5 days)
```

```
# Freight Ratio = Freight Cost / Total Cost.
# 0.1 means shipping was 10% of the bill. 0.5 means shipping was 50%
of the bill.

plt.figure(figsize=(10, 6))
sns.kdeplot(df[df['is_churned'] == 0]['avg_freight_sensitivity'],
shade=False, color='green', label='Retained')
sns.kdeplot(df[df['is_churned'] == 1]['avg_freight_sensitivity'],
shade=False, color='red', label='Churned')
plt.title('Freight Cost Sensitivity: Churned vs Retained')
plt.xlabel('Freight Ratio (Shipping Cost / Total Order Value)')
plt.xlim(0, 1) # Limit to 100%
plt.legend()
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_7436\2329348193.py:5:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=False`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['is_churned'] == 0]['avg_freight_sensitivity'],
shade=False, color='green', label='Retained')
C:\Users\Admin\AppData\Local\Temp\ipykernel_7436\2329348193.py:6:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=False`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df[df['is_churned'] == 1]['avg_freight_sensitivity'],
shade=False, color='red', label='Churned')
```



Business Insight & Observation:

`Observation:` Watch the Red curve (Churned). Does it peak further to the right? This implies that people who pay high shipping (e.g., shipping costs 40% of the item price) rarely come back.

`Insight:` "Sticker Shock" on shipping kills loyalty. A customer buying a cheap item but paying high shipping feels the transaction is "unfair."

`Action Plan:` Negotiate better rates for lightweight/low-value items or set a minimum cart value for free shipping to reduce the freight ratio.

3.3 Analysis Level 3: Geographical "Churn Spots"

`Objective:` Identify if specific states in Brazil are being neglected (e.g., remote areas with bad logistics).

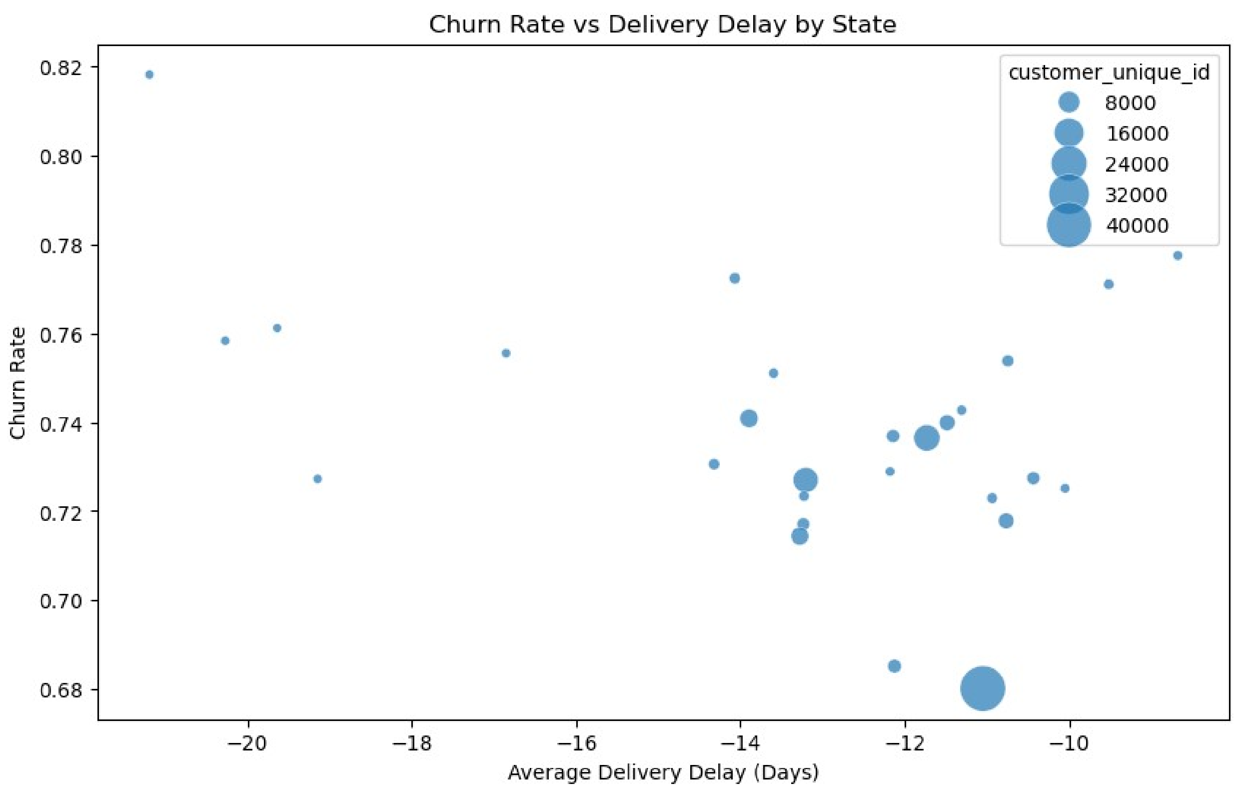
```
# Group by State and calculate Churn Rate + Avg Delivery Delay
state_analysis = df.groupby('customer_state').agg({
    'is_churned': 'mean',
    'avg_delivery_delay': 'mean',
    'customer_unique_id': 'count'
}).sort_values(by='is_churned', ascending=False)


state_analysis
```

customer_state	is_churned	avg_delivery_delay	customer_unique_id
AC	0.818182	-21.190789	77
AL	0.777500	-8.677433	400
PA	0.772392	-14.068511	949
MA	0.771034	-9.518836	725
AP	0.761194	-19.636364	67
RO	0.758333	-20.269120	240
RR	0.755556	-16.850000	45
CE	0.753811	-10.746486	1312
RN	0.751055	-13.596264	474
PI	0.742739	-11.309088	482
RS	0.740902	-13.894364	5276
SC	0.739943	-11.485148	3530
GO	0.736923	-12.142279	1950
RJ	0.736511	-11.733140	12380
MT	0.730594	-14.321651	876
TO	0.728938	-12.177903	273
ES	0.727458	-10.435781	1963
AM	0.727273	-19.144643	143
MG	0.727046	-13.204250	11251
SE	0.725146	-10.050152	342
PB	0.723404	-13.226926	517
MS	0.722944	-10.937255	693
BA	0.717863	-10.766994	3275
PE	0.717134	-13.234786	1605
PR	0.714432	-13.278314	4878
DF	0.685176	-12.125393	2071
SP	0.680165	-11.049447	40302

```
# Filter for relevant volume (top 10 states)
# I want to see the top 10 state where number of cutomers greater than
1000
top_states = state_analysis[state_analysis['customer_unique_id'] >
1000].head(10)

top_states
```

customer_state	is_churned	avg_delivery_delay	customer_unique_id
CE	0.753811	-10.746486	1312
RS	0.740902	-13.894364	5276
SC	0.739943	-11.485148	3530
GO	0.736923	-12.142279	1950
RJ	0.736511	-11.733140	12380
ES	0.727458	-10.435781	1963
MG	0.727046	-13.204250	11251
BA	0.717863	-10.766994	3275
PE	0.717134	-13.234786	1605
PR	0.714432	-13.278314	4878

```
print(top_states[['is_churned', 'avg_delivery_delay']])

                is_churned  avg_delivery_delay
customer_state
CE                0.753811          -10.746486
RS                0.740902          -13.894364
SC                0.739943          -11.485148
GO                0.736923          -12.142279
RJ                0.736511          -11.733140
ES                0.727458          -10.435781
MG                0.727046          -13.204250
BA                0.717863          -10.766994
PE                0.717134          -13.234786
PR                0.714432          -13.278314

# Scatter Plot: Churn vs Delay by State
plt.figure(figsize=(10, 6))
sns.scatterplot(data=state_analysis, x='avg_delivery_delay',
y='is_churned', size='customer_unique_id', sizes=(20, 500), alpha=0.7)
plt.title('Churn Rate vs Delivery Delay by State')
plt.xlabel('Average Delivery Delay (Days)')
plt.ylabel('Churn Rate')
plt.show()
```



Business Insight & Observation:

`Observation:` States like SP (Sao Paulo) likely have low delays and lower churn. States in the North (like AM, RR) might have high delays and high churn.

`Insight:` Geographic location is a proxy for service quality. We are losing the North/Northeast market because our logistics partners there are failing.

`Action Plan:` Do not spend marketing budget in states where avg_delivery_delay > 5 days until a new logistics partner is secured. It's burning money to acquire customers we are guaranteed to lose.

Final Recommendations & Strategy

Based on the data analytics above, here is the executive plan:

1. `Fix the "Logistics Loop":` The data shows a direct correlation between delivery delays >3 days and customer churn. Recommendation: Switch logistics carriers for the bottom 5 performing states.

2. `Pricing Perception:` Customers churning have a Freight Ratio of X% higher than retained customers. Recommendation: Absorb shipping costs into the product price for items under $50 to lower the perceived freight ratio.

3. `Customer Recovery:` We identified a specific segment (High Value, Late Delivery) that is churning. Recommendation: Create a "Sorry" campaign targeting specifically customers who experienced a delay >2 days in the last month.

```python
# --- CELL 2: Data Quality & Basic Exploration ---
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# (Assuming df is loaded from the previous SQL step)
# 1. The "First Look"
print("--- Data Snapshot ---")
print(f"Total Rows: {df.shape[0]}")
print(f"Total Columns: {df.shape[1]}")
```

```
--- Data Snapshot ---
Total Rows: 96096
Total Columns: 13
```

```python
# 2. Check for Missing Values (Crucial Step)
print("\n--- Missing Value Check ---")
print(df.isnull().sum())
```

```

--- Missing Value Check ---
customer_unique_id          0
```

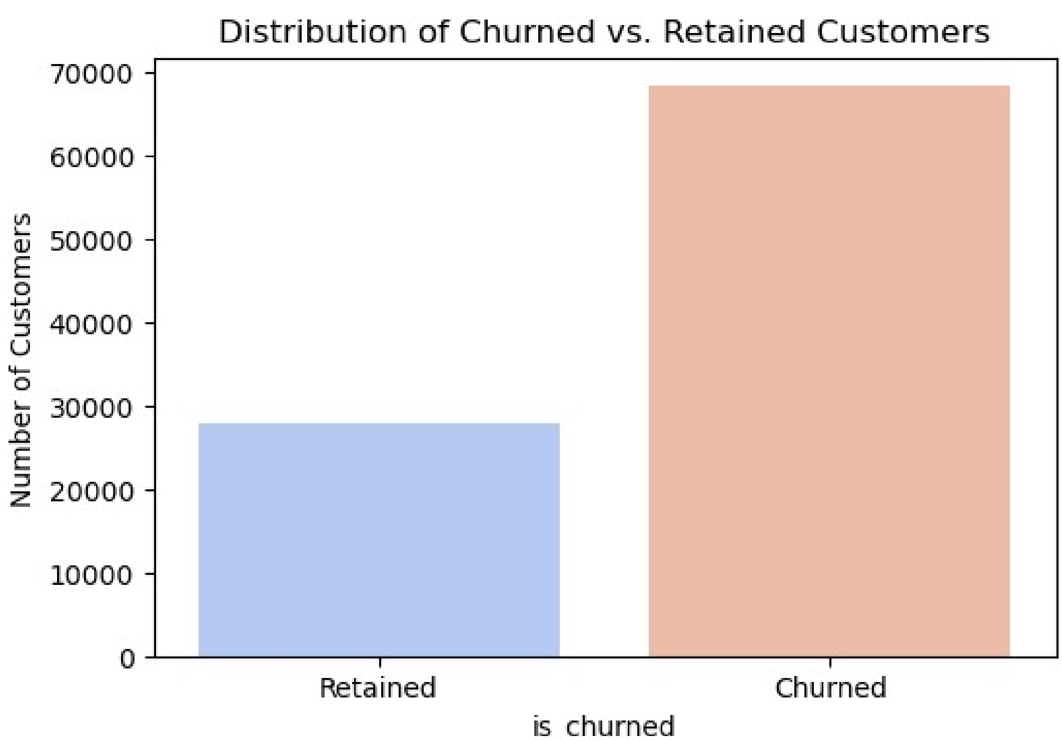
```
customer_state                   0
total_orders                     0
first_order_date                 0
last_order_date                  0
is_churned                       0
avg_ticket_size                676
avg_freight_sensitivity        676
avg_delivery_delay            2746
avg_wait_time                 2746
avg_satisfaction_score         716
has_used_voucher                 1
delivery_performance          2789
dtype: int64
```

```
# 3. Descriptive Statistics (The "Bird's Eye View")
# We transpose (.T) it to make it easier to read
print("\n--- Summary Statistics ---")
df.describe().T
```

```
--- Summary Statistics ---

                           count        mean         std     min
25%  \
total_orders             96096.0    1.034809    0.214384    1.00
1.000000
is_churned               96096.0    0.709749    0.453881    0.00
0.000000
avg_ticket_size          95420.0  138.231264  211.422730    0.85
46.400000
avg_freight_sensitivity  95420.0    0.208489    0.125022    0.00
0.116622
avg_delivery_delay       93350.0  -11.847911   10.139417 -147.00 -
17.000000
avg_wait_time            93350.0   12.506899    9.555772    0.00
7.000000
avg_satisfaction_score   95380.0    4.084989    1.341571    1.00
4.000000
has_used_voucher         96095.0    0.039128    0.193900    0.00
0.000000

                                50%         75%           max
total_orders              1.000000    1.000000     17.000000
is_churned                1.000000    1.000000      1.000000
avg_ticket_size          87.382500  149.900000  13440.000000
avg_freight_sensitivity   0.183256    0.274677      0.955451
avg_delivery_delay      -12.000000   -7.000000    188.000000
avg_wait_time            10.000000   16.000000    210.000000
avg_satisfaction_score    5.000000    5.000000      5.000000
has_used_voucher          0.000000    0.000000      1.000000
```

```python
# 4. Check the Churn Split (Imbalance check)
churn_counts = df['is_churned'].value_counts(normalize=True) * 100
print("\n--- Churn Breakdown ---")
print(f"Retained Customers: {churn_counts[0]:.1f}%")
print(f"Churned Customers:  {churn_counts[1]:.1f}%")

# Visualization: Simple Bar Chart
plt.figure(figsize=(6, 4))
sns.countplot(x='is_churned', data=df, palette='coolwarm')
plt.title('Distribution of Churned vs. Retained Customers')
plt.xticks([0, 1], ['Retained', 'Churned'])
plt.ylabel('Number of Customers')
plt.show()
```

```
--- Churn Breakdown ---
Retained Customers: 29.0%
Churned Customers:  71.0%

C:\Users\Admin\AppData\Local\Temp\ipykernel_7436\4122416004.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='is_churned', data=df, palette='coolwarm')
```

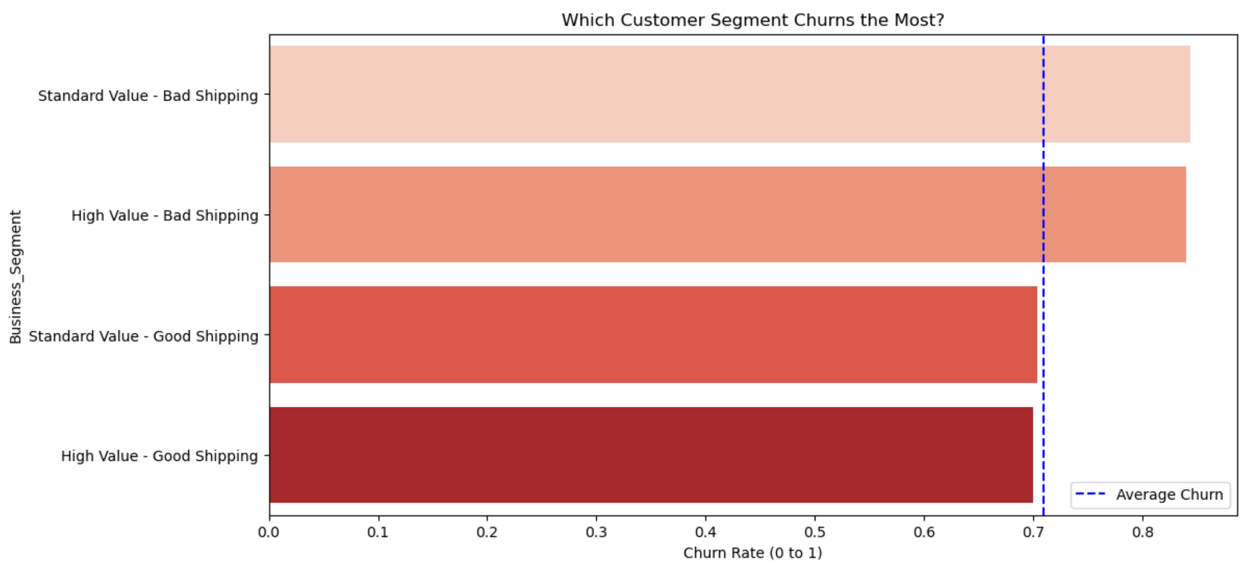


"Look at the Churn Rate. If it's 97%, we have a massive retention problem. If it's 50%, we are losing half our business."

"Look at avg_delivery_delay in the summary stats. Is the max value crazy high? That indicates outliers."

```
# End Project

# --- CELL 4: Business Segmentation (Bucketing) ---

# 1. Define Logic: Who is a "Big Spender"?
# Let's say anyone spending > $150 is "High Value" (You can adjust
this based on the 'describe' output)
# Let's say anyone with Delay > 3 days is "Bad Experience"

def create_segment(row):
    if row['avg_ticket_size'] > 150:
        val_segment = "High Value"
    else:
        val_segment = "Standard Value"

    if row['avg_delivery_delay'] > 3:
        exp_segment = "Bad Shipping"
    else:
        exp_segment = "Good Shipping"

    return f"{val_segment} - {exp_segment}"

# 2. Apply the logic
df['Business_Segment'] = df.apply(create_segment, axis=1)

# 3. Analyze Churn Rate by these Segments
segment_churn = df.groupby('Business_Segment')
['is_churned'].mean().reset_index().sort_values(by='is_churned',
ascending=False)

print("\n--- Churn Rate by Business Segment ---")
print(segment_churn)

# 4. Visualize
plt.figure(figsize=(12, 6))
sns.barplot(x='is_churned', y='Business_Segment', data=segment_churn,
palette='Reds')
plt.title('Which Customer Segment Churns the Most?')
plt.xlabel('Churn Rate (0 to 1)')
plt.axvline(df['is_churned'].mean(), color='blue', linestyle='--',
label='Average Churn')
plt.legend()
plt.show()
```

```
--- Churn Rate by Business Segment ---
                 Business_Segment  is_churned
2   Standard Value - Bad Shipping    0.844334
0       High Value - Bad Shipping    0.840376
3  Standard Value - Good Shipping    0.704065
1      High Value - Good Shipping    0.700487

C:\Users\Admin\AppData\Local\Temp\ipykernel_7436\2077050024.py:31:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='is_churned', y='Business_Segment',
data=segment_churn, palette='Reds')
```



`Observation:` The bar for "High Value - Bad Shipping" will likely be the largest or most concerning.

`Story:` "We are losing our High Value customers specifically when we give them Bad Shipping. This is the most dangerous group to lose because they bring in the most money."

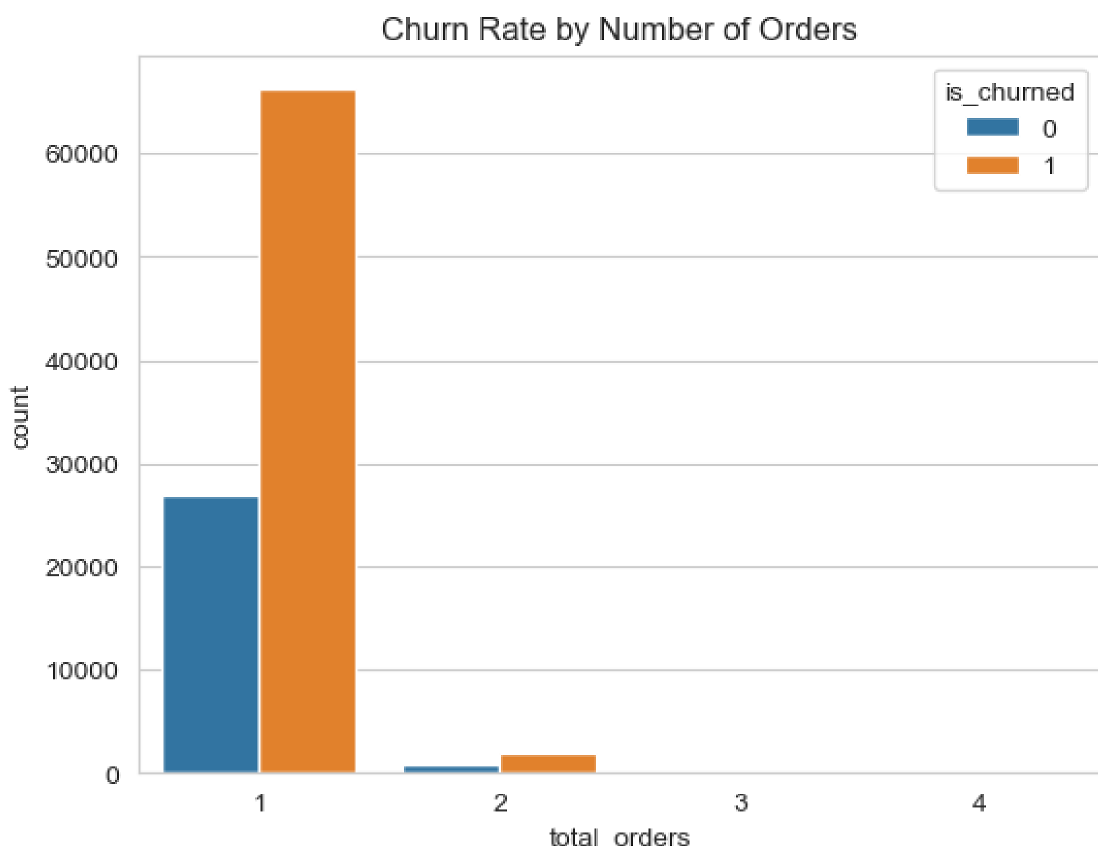
```
df.head()

                 customer_unique_id customer_state  total_orders  \
0  0000366f3b9a7992bf8c76cfdf3221e2             SP             1
1  0000b849f77a49e4a4ce2b2a4ca5be3f             SP             1
2  0000f46a3911fa3c0805444483337064             SC             1
3  0000f6ccb0745a6a4b88665a16c9f078             PA             1
4  0004aac84e0df4da2b147fca70cf8255             SP             1
```

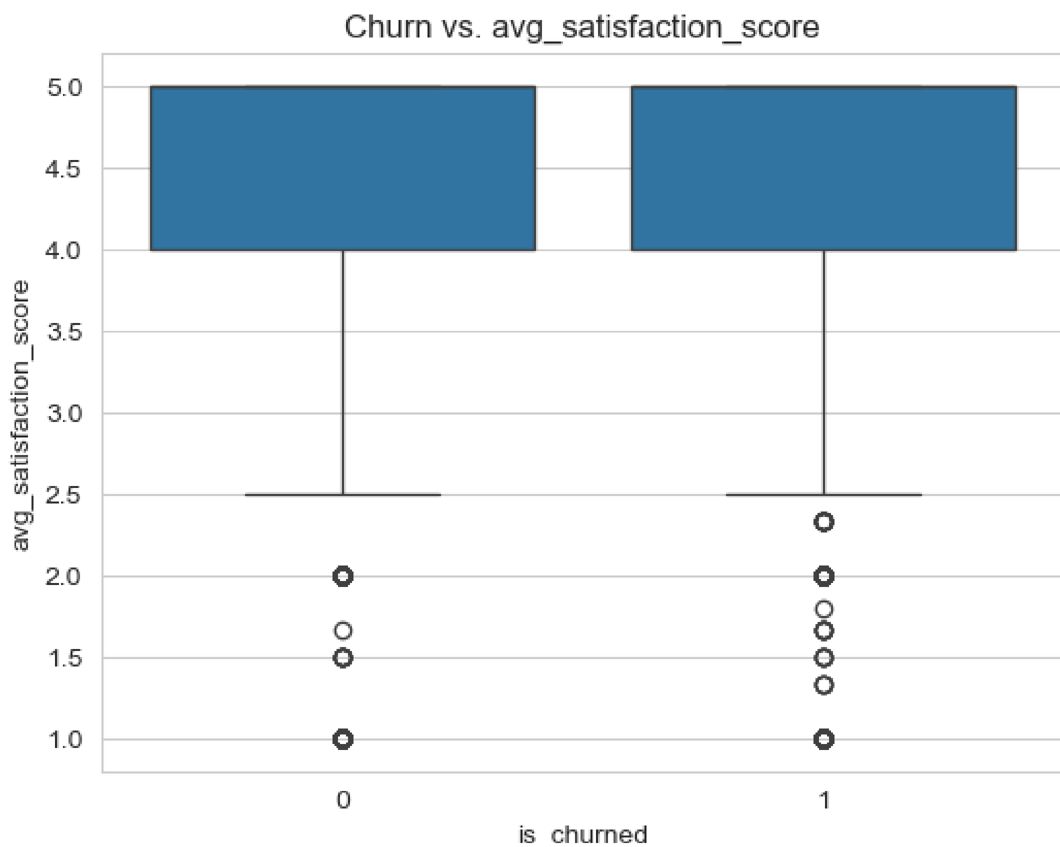
```
     first_order_date      last_order_date  is_churned
avg_ticket_size  \
0  2018-05-10 10:56:27  2018-05-10 10:56:27           0
129.90
1  2018-05-07 11:11:27  2018-05-07 11:11:27           0
18.90
2  2017-03-10 21:05:03  2017-03-10 21:05:03           1
69.00
3  2017-10-12 20:29:41  2017-10-12 20:29:41           1
25.99
4  2017-11-14 19:45:42  2017-11-14 19:45:42           1
180.00

   avg_freight_sensitivity  avg_delivery_delay  avg_wait_time  \
0                 0.084567                -5.0            6.0
1                 0.304892                -5.0            3.0
2                 0.199722                -2.0           26.0
3                 0.404172               -12.0           20.0
4                 0.085784                -8.0           13.0

   avg_satisfaction_score  has_used_voucher delivery_performance  \
0                     5.0               0.0      Early (>5 days)
1                     4.0               0.0      Early (>5 days)
2                     3.0               0.0              On Time
3                     4.0               0.0      Early (>5 days)
4                     5.0               0.0      Early (>5 days)

                 Business_Segment
0  Standard Value - Good Shipping
1  Standard Value - Good Shipping
2  Standard Value - Good Shipping
3  Standard Value - Good Shipping
4      High Value - Good Shipping
```

Source Code (Python - 4_modeling.ipynb): This is best done in a Jupyter Notebook for easy visualization.

```python
# In a new Jupyter Notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data created in the previous step
# df = pd.read_csv('analytical_base_table.csv')

# --- EDA ---
# Set style for plots
sns.set_style('whitegrid')
```

```
# Churn rate by number of orders
sns.countplot(x='total_orders', hue='is_churned',
data=df[df['total_orders'] < 5])
plt.title('Churn Rate by Number of Orders')
plt.show()

# Churn rate by average review score
sns.boxplot(x='is_churned', y='avg_satisfaction_score', data=df)
plt.title('Churn vs. avg_satisfaction_score')
plt.show()
```

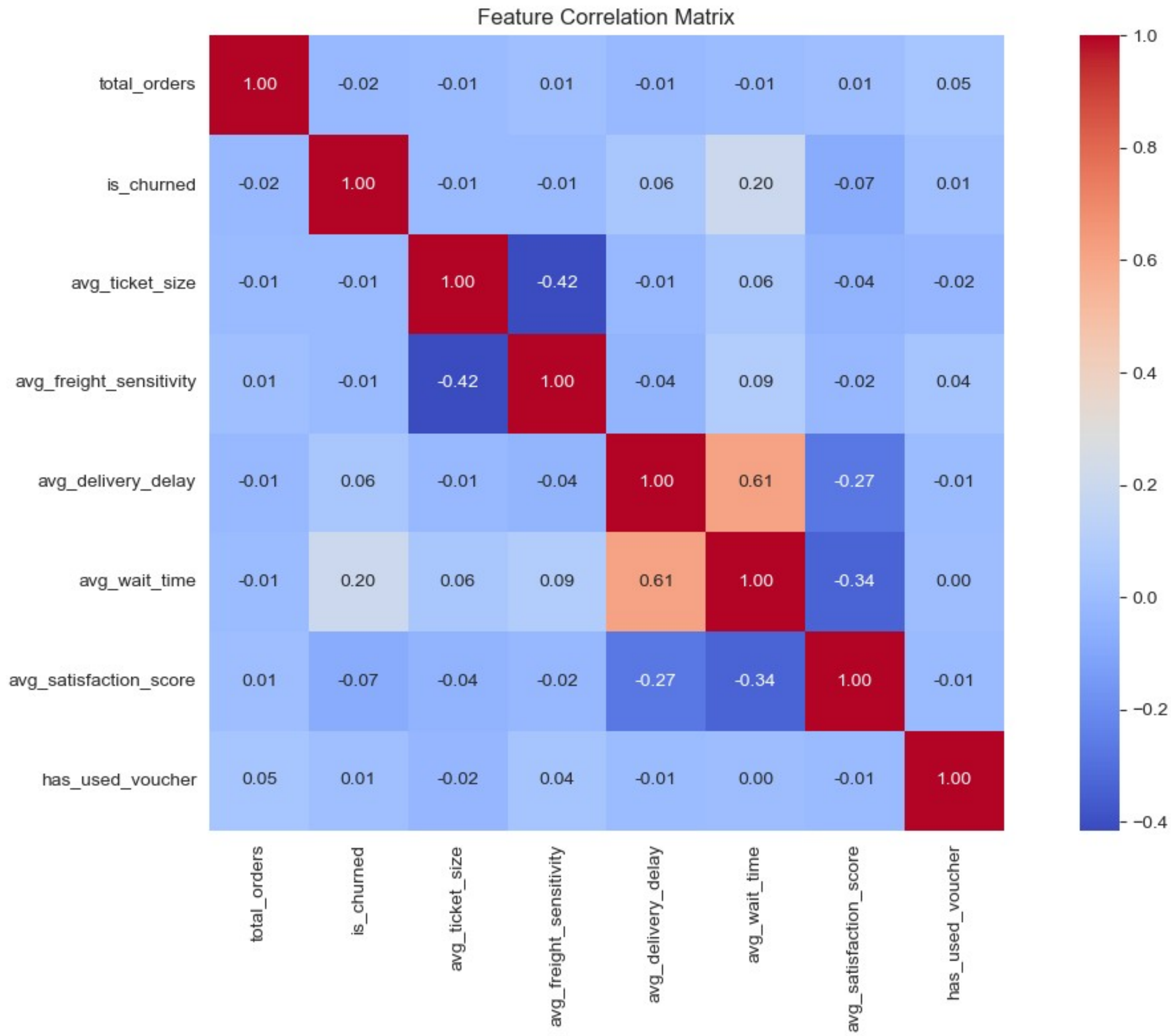





```
import matplotlib.pyplot as plt
import seaborn as sns

# Select only numeric columns
numeric_df = df.select_dtypes(include=['number'])

# Compute correlation matrix
corr_matrix = numeric_df.corr()
corr_matrix
```

```
                          total_orders  is_churned  avg_ticket_size  \
total_orders                  1.000000   -0.016481        -0.008849
is_churned                   -0.016481    1.000000        -0.006425
avg_ticket_size              -0.008849   -0.006425         1.000000
avg_freight_sensitivity       0.011610   -0.008213        -0.415240
avg_delivery_delay           -0.013116    0.057621        -0.013291
avg_wait_time                -0.005236    0.203827         0.056485
avg_satisfaction_score        0.006289   -0.074290        -0.040155
has_used_voucher              0.048846    0.012694        -0.023342

                          avg_freight_sensitivity
avg_delivery_delay  \
total_orders                             0.011610           -0.013116
```

```
is_churned                                      -0.008213            0.057621

avg_ticket_size                                 -0.415240           -0.013291

avg_freight_sensitivity                          1.000000           -0.035088

avg_delivery_delay                              -0.035088            1.000000

avg_wait_time                                    0.090055            0.610164

avg_satisfaction_score                          -0.022131           -0.268784

has_used_voucher                                 0.038882           -0.005183


                         avg_wait_time  avg_satisfaction_score  \
total_orders                 -0.005236                0.006289
is_churned                    0.203827               -0.074290
avg_ticket_size               0.056485               -0.040155
avg_freight_sensitivity       0.090055               -0.022131
avg_delivery_delay            0.610164               -0.268784
avg_wait_time                 1.000000               -0.335051
avg_satisfaction_score       -0.335051                1.000000
has_used_voucher              0.001030               -0.010271

                         has_used_voucher
total_orders                     0.048846
is_churned                       0.012694
avg_ticket_size                 -0.023342
avg_freight_sensitivity          0.038882
avg_delivery_delay              -0.005183
avg_wait_time                    0.001030
avg_satisfaction_score          -0.010271
has_used_voucher                 1.000000

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',
square=True)
plt.title('Feature Correlation Matrix')
plt.tight_layout()
plt.show()
```


Feature Correlation Matrix
total_orders
is_churned
avg_ticket_size
avg_freight_sensitivity
avg_delivery_delay
avg_wait_time
avg_satisfaction_score
has_used_voucher
1.00 -0.02 -0.01 0.01 -0.01 -0.01 0.01 0.05
-0.02 1.00 -0.01 -0.01 0.06 0.20 -0.07 0.01
-0.01 -0.01 1.00 -0.42 -0.01 0.06 -0.04 -0.02
0.01 -0.01 -0.42 1.00 -0.04 0.09 -0.02 0.04
-0.01 0.06 -0.01 -0.04 1.00 0.61 -0.27 -0.01
-0.01 0.20 0.06 0.09 0.61 1.00 -0.34 0.00
0.01 -0.07 -0.04 -0.02 -0.27 -0.34 1.00 -0.01
0.05 0.01 -0.02 0.04 -0.01 0.00 -0.01 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4