PROJECT REPORT

Design and Implementation of an End-to-End E-commerce ETL and Analytics Pipeline

Adhithyaraj P R

September 2025

Abstract

This project presents a complete end-to-end data engineering and analytical workflow developed using a real-world e-commerce dataset. The objective is to design a fully functional ETL pipeline that extracts raw transactional data, transforms it into a structured and reliable analytical repository, and produces insights through statistical analysis, visual exploration, and predictive modelling.

The workflow integrates multiple heterogeneous data tables, performs feature engineering for delivery performance, executes data quality checks, and generates analytical summaries. The project further incorporates a machine learning model to predict delivery delays and employs model explainability techniques to interpret the predictions. The final system includes an organised output directory structure and an SQLite-based analytical data store suitable for reporting, querying, and downstream machine-learning tasks. The pipeline demonstrates the practical application of data engineering principles in an e-commerce environment and is designed with clarity, reproducibility, and academic standards suitable for graduate programme evaluation.

1. CHAPTER 1: PROJECT DESCRIPTION AND OUTLINE

1.1. Background and Motivation

E‑commerce platforms produce large volumes of heterogeneous data representing customer behaviour, product demand, operational processes, payment patterns, and fulfilment efficiency. To extract value from such rich data sources, organisations rely on robust data engineering pipelines that integrate and transform raw information into actionable insights.

This project aims to replicate a real‑world scenario by designing and implementing a structured ETL and analytics workflow using the Olist dataset. The dataset is particularly suitable for academic demonstration because it includes detailed order transactions, product metadata, customer demographics, delivery timestamps, and review feedback.

1.2. Problem Statement

Although the dataset is comprehensive, the raw files are not ready for immediate analysis. The information is spread across multiple tables, contains missing values, inconsistent formats, and lacks engineered features required for meaningful interpretation. The project addresses these challenges by constructing a unified master table, performing data quality checks, deriving analytical metrics, and building predictive models.

1.3. Objectives of the Work

The project focuses on achieving the following:

- Develop a complete ETL pipeline from raw ingestion to processed output.
- Merge multiple relational datasets into a unified analytical master table.
- Conduct data quality assessments and statistical validation.
- Perform exploratory analysis supported by visualisations.

- Build a predictive model to identify orders at risk of delivery delays.
- Apply explainability techniques to interpret model behaviour.
- Load final analytical tables into an SQLite database for structured querying.

1.4. Scope of the Project

The scope covers data engineering, analytical processing, and predictive modelling. It does not include dashboard deployment or streaming pipelines. The final system reflects an offline analytical setup similar to industry batch-processing workflows.

1.5. Structure of the Report

The report is divided into five main chapters, beginning with an introduction followed by dataset details, methodological steps, analytical results, and concluding statements.

2. CHAPTER 2: REQUIREMENT ARTIFACTS

2.1. Overview of the Dataset

The Olist dataset is a multi-table relational dataset containing detailed information on e-commerce operations across Brazil. It includes customer records, order lifecycle timestamps, product categories, payments, reviews, and delivery details. Its breadth and depth make it ideal for building an end‑to‑end data engineering demonstration.

2.2. Description of Key Tables

Orders Table : Contains timestamps for purchase, approval, shipment, and delivery. Enables modelling of seasonality and fulfilment behaviour.

Order Items Table : Holds product-level financial components such as price and freight value. Forms the basis of revenue computation.

Customers Table : Provides unique identifiers and location attributes, enabling geographic analysis.

Payments Table : Includes payment methods and values. Aggregated to compute total financial inflow per order.

Products Table : Contains product category metadata, which is translated into English for easier interpretation.

Reviews Table : Documents customer satisfaction through numeric review scores and timestamps.

2.3. Data Challenges

The dataset includes incomplete timestamps, null delivery dates, inconsistent category naming, and variations in payment values. These challenges justify the need for a structured ETL pipeline.

3. CHAPTER 3: DESIGN AND METHODOLOGY

This project follows a structured data science workflow, as illustrated in the Employee Attrition Project.ipynb.

3.1. ETL Pipeline Overview

The pipeline follows a structured workflow beginning with raw data ingestion and progressing through cleaning, transformation, integration, quality checks, analytics, machine learning, and storage. A clear directory hierarchy is maintained to ensure reproducibility.

3.2. Extract Phase

Raw CSV files are uploaded into a defined raw directory. A custom loading function ensures controlled access and prevents accidental overwriting. No processing is done at this stage to preserve data integrity.

3.3. Transform Phase

3.3.1 Cleaning and Standardisation

All timestamps are converted to uniform datetime formats. Numeric fields such as price and freight are validated. Missing values are handled using appropriate imputation strategies.

3.3.2 Feature Engineering

Several new attributes are created, including:

- Total item amount per order line.
- Aggregated payment totals.
- Delivery delay measured in days.
- Total delivery time.
- English-translated category names.

3.3.3 Master Table Construction

Multiple tables are joined using relational keys to form a consolidated fact table. This unified dataset forms the basis for visual analytics and predictive modelling.

3.4. Data Quality Validation

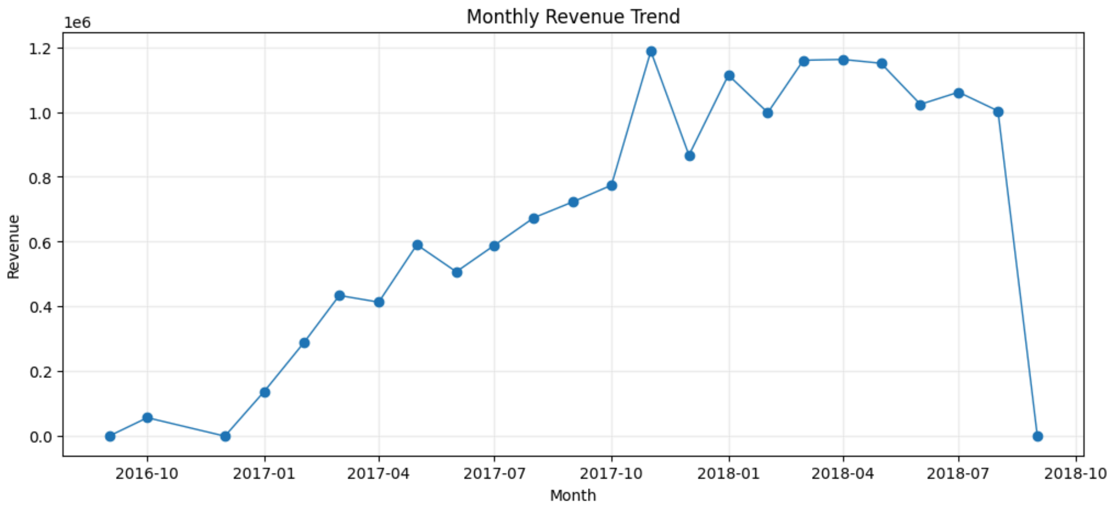
Null value reports, descriptive statistics, and consistency checks are generated and saved. These quality checks ensure reliability of downstream analytics.

3.5. Load Phase

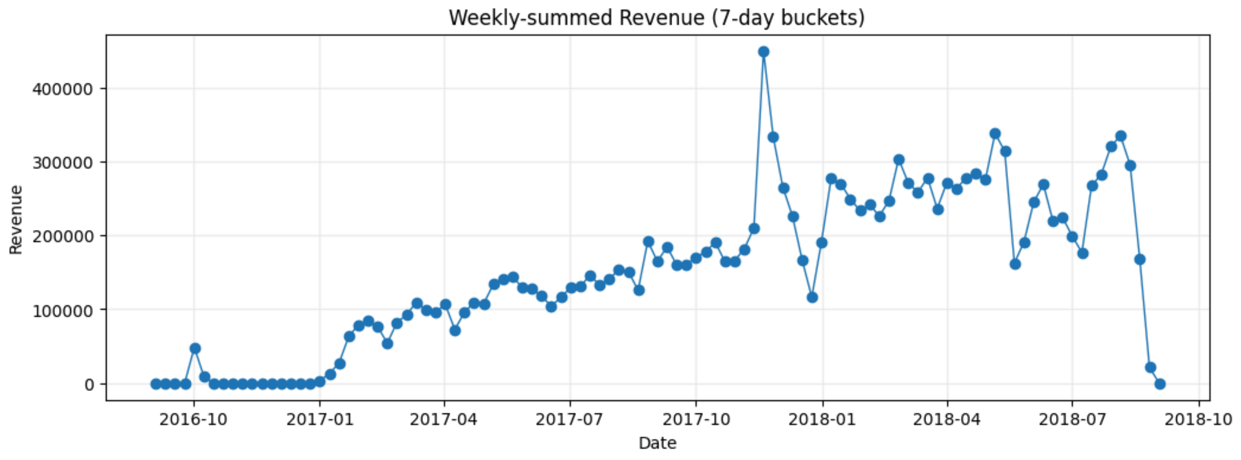
The cleaned analytical tables are written into an SQLite database. This database includes monthly revenue summaries, category‑wise and state‑wise revenue, weekly revenue, customer-level summaries, and the master fact table.

4. CHAPTER 4: RESULTS AND ANALYSIS

4.1. Revenue Trends

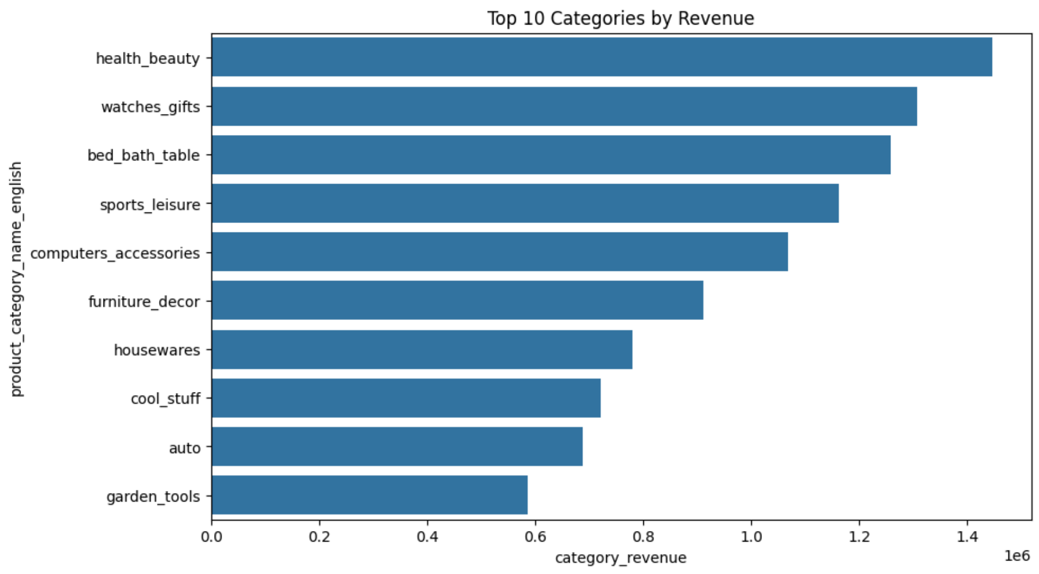


Monthly revenue exhibits seasonal patterns, with fluctuating demand across different periods. The trend reflects broader consumer behaviour and operational cycles.

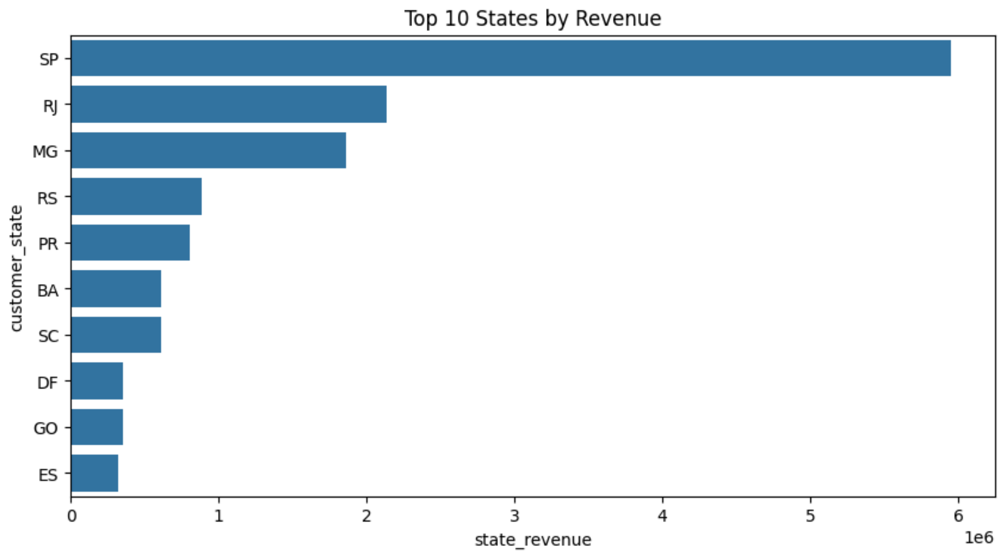


Weekly aggregated revenue captures shorter-term variation and highlights recurring peaks that align with typical order cycles.

4.2. Category and Regional Insights

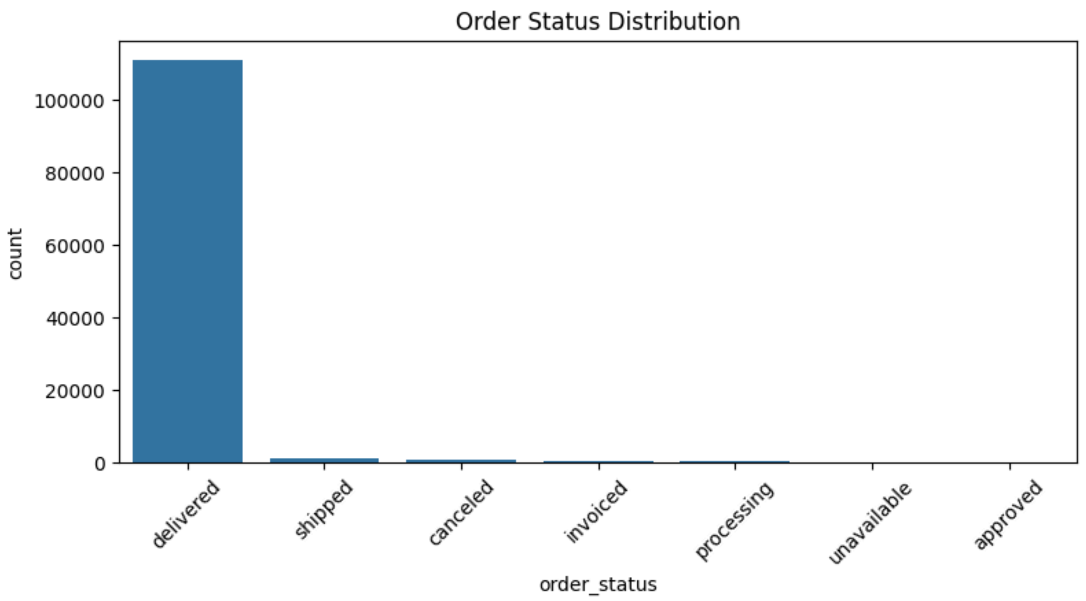


The analysis shows that a small number of product categories account for the highest revenue. This concentration provides valuable direction for inventory planning and promotional activities.



State-level analysis reveals strong demand clusters. These results support targeted logistics and regional strategies.

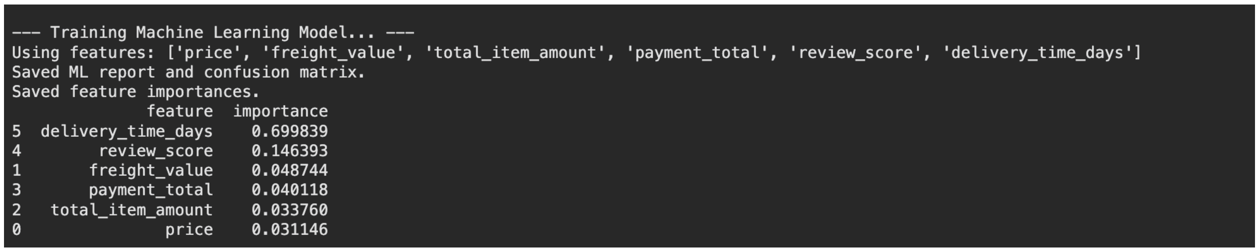
4.3. Operational Performance



The order status distribution highlights fulfilment behaviour, revealing proportions of completed, cancelled, and pending orders.

4.4. Predictive Modelling and Model Explainability

A Random Forest classifier is trained to predict delayed orders using price, freight value, review score, and delivery duration as features. The model offers a structured understanding of factors contributing to delays.



```
--- Training Machine Learning Model... ---
Using features: ['price', 'freight_value', 'total_item_amount', 'payment_total', 'review_score', 'delivery_time_days']
Saved ML report and confusion matrix.
Saved feature importances.
              feature  importance
5  delivery_time_days    0.699839
4        review_score    0.146393
1       freight_value    0.048744
3       payment_total    0.040118
2   total_item_amount    0.033760
0               price    0.031146
```

The confusion matrix indicates the model's accuracy in distinguishing between delayed and non-delayed orders. The SHAP summary visualisation highlights feature contributions, enabling a transparent understanding of model behaviour.

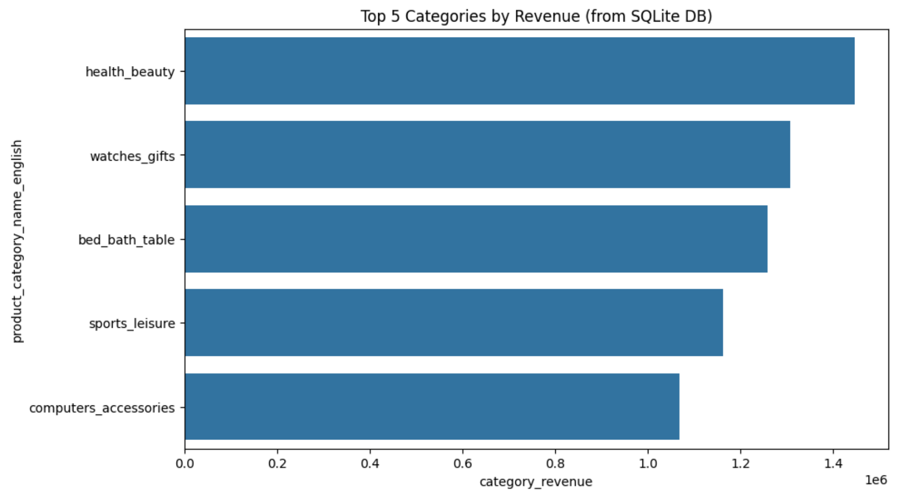
5. CHAPTER 5: CONCLUSIONS

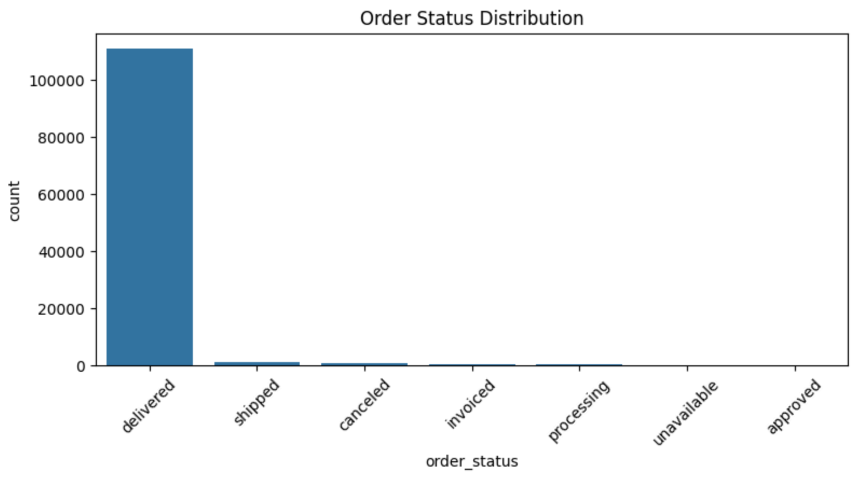
This project demonstrates a complete end-to-end e-commerce ETL and analytics pipeline suitable for academic evaluation and practical deployment. The pipeline successfully transforms raw multi-table input into a structured analytical environment, produces meaningful insights through visualisation, and implements a predictive model supported by explainable AI techniques.

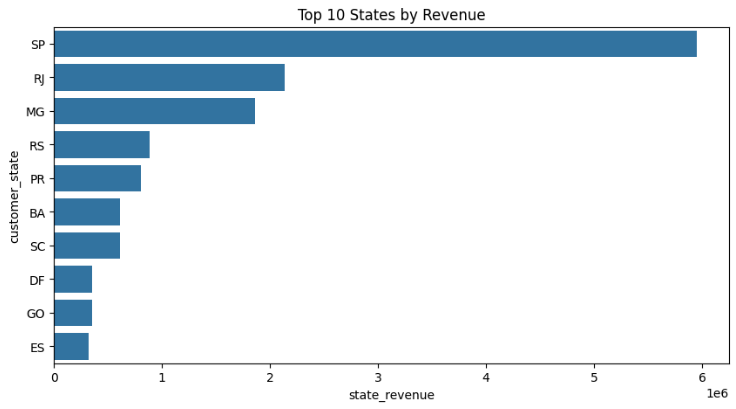
The work reflects competencies in data engineering, analytics, and machine learning, aligning well with the expectations of data-driven academic programmes at German universities.

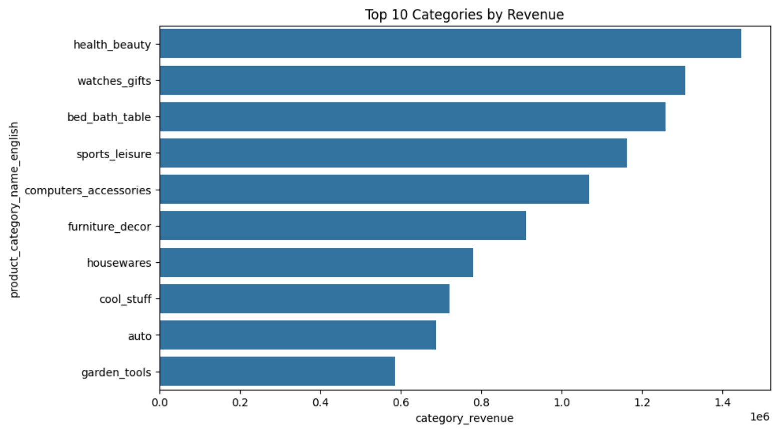
6. CHAPTER 6: APPENDIX

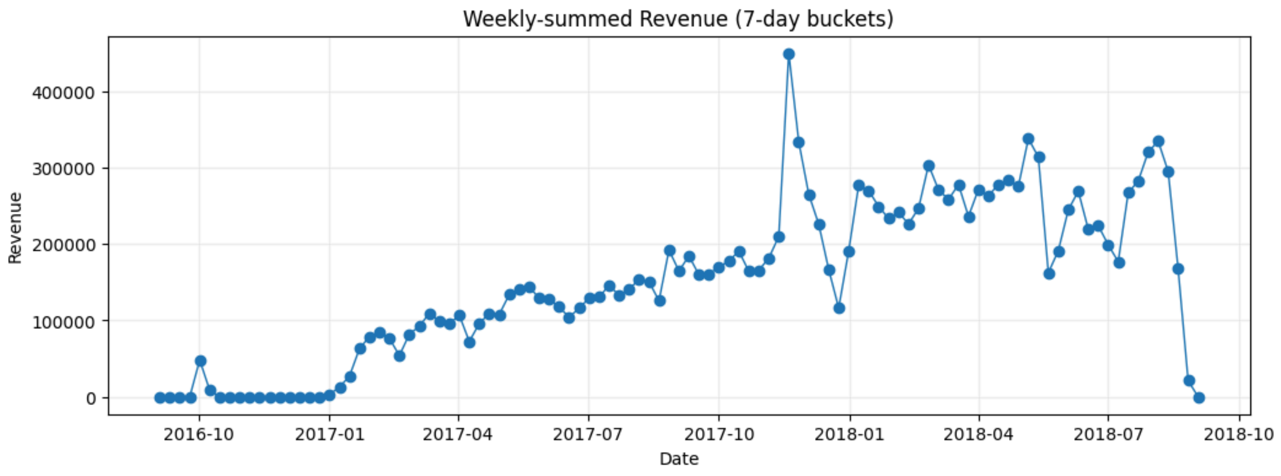
6.1. Visualizations

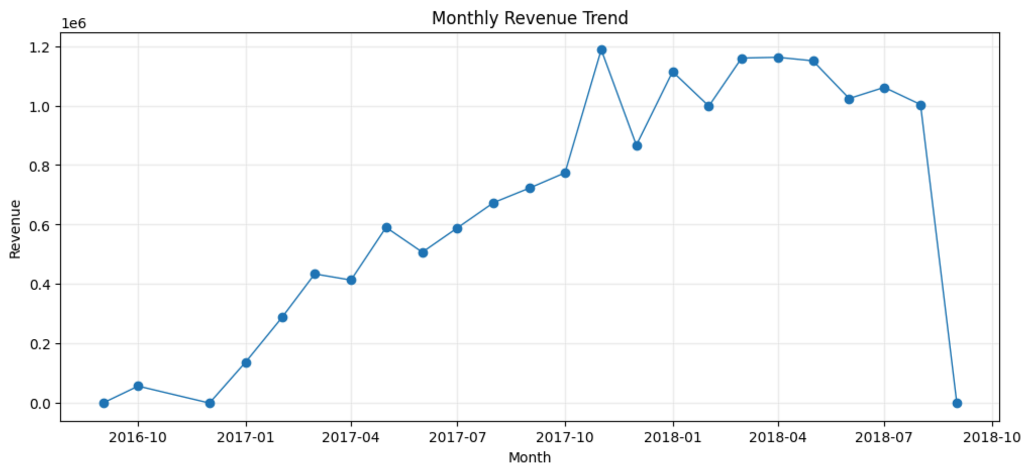







Top 10 Categories by Revenue
product_category_name_english
health_beauty
watches_gifts
bed_bath_table
sports_leisure
computers_accessories
furniture_decor
housewares
cool_stuff
auto
garden_tools
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
category_revenue


Weekly-summed Revenue (7-day buckets)
Revenue
400000
300000
200000
100000
0
2016-10
2017-01
2017-04
2017-07
2017-10
2018-01
2018-04
2018-07
2018-10
Date


Monthly Revenue Trend
1e6
Revenue
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2016-10
2017-01
2017-04
2017-07
2017-10
2018-01
2018-04
2018-07
2018-10
Month

6.2. Code Snippets

```
#
=============================================================================
==========

# IMPORT ALL LIBRARIES

#
=============================================================================
==========

import pandas as pd

import numpy as np

import os

import matplotlib.pyplot as plt

import seaborn as sns

from sqlalchemy import create_engine, text

import sqlite3

import glob

import shutil

import zipfile

from google.colab import files

from IPython.display import Image, display, HTML
```

```
# ML/SHAP Imports

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix

!pip install shap --quiet

import shap


print("--- All libraries imported successfully. ---")




#
===============================================================================
=========

# SETUP FOLDERS AND DEFINE FILE LOADING FUNCTION

#
===============================================================================
=========

# Create folder structure

BASE = "/content/project3"

RAW_DIR = os.path.join(BASE, "data", "raw")

STAGING_DIR = os.path.join(BASE, "data", "staging")
```

```
OUT_DIR = os.path.join(BASE, "outputs", "visuals")

os.makedirs(RAW_DIR, exist_ok=True)

os.makedirs(STAGING_DIR, exist_ok=True)

os.makedirs(OUT_DIR, exist_ok=True)

os.makedirs('outputs/qc', exist_ok=True)

os.makedirs('outputs/figures', exist_ok=True)

os.makedirs('outputs/tables', exist_ok=True)

print("Folder structure created.")



# Define the load_csv function

def load_csv(name):

   path = os.path.join(RAW_DIR, name)

   if not os.path.exists(path):

       raise FileNotFoundError(f"{name} not found in RAW_DIR")

   return pd.read_csv(path, low_memory=False)

print("load_csv function defined.")




#
==============================================================================
=========
# UPLOAD AND LOAD DATA (ETL - EXTRACT)

#
==============================================================================
=========

print("\n--- Please upload the 9 Olist CSV files... ---")

uploaded = files.upload()

for fn in uploaded.keys():

    src = "/content/" + fn

    dst = os.path.join(RAW_DIR, fn)

    shutil.move(src, dst)

print(f"Moved {len(uploaded.keys())} files to {RAW_DIR}.")



# Load CSVs into DataFrames

orders_df = load_csv("olist_orders_dataset.csv")

items_df = load_csv("olist_order_items_dataset.csv")

customers_df = load_csv("olist_customers_dataset.csv")

payments_df = load_csv("olist_order_payments_dataset.csv")

products_df = load_csv("olist_products_dataset.csv")

translation_df = load_csv("product_category_name_translation.csv")

reviews_df = load_csv("olist_order_reviews_dataset.csv")

# We don't need these two for the main analysis, but they are part of the set
# sellers_df = load_csv("olist_sellers_dataset.csv")

# geolocation_df = load_csv("olist_geolocation_dataset.csv")

print("--- All key files loaded successfully. ---")




#
=====================================================================
=========

# CLEANING & FEATURE ENGINEERING (ETL - TRANSFORM)

#
=====================================================================
=========

print("\n--- Cleaning data and engineering new features... ---")



# Basic cleaning and type conversion

orders_df['order_purchase_timestamp'] = pd.to_datetime(orders_df['order_purchase_timestamp'],
errors='coerce')

items_df['price'] = pd.to_numeric(items_df['price'], errors='coerce').fillna(0)

items_df['freight_value'] = pd.to_numeric(items_df['freight_value'], errors='coerce').fillna(0)

payments_df['payment_value'] = pd.to_numeric(payments_df['payment_value'],
errors='coerce').fillna(0)
```

```
# Create item total

items_df['total_item_amount'] = items_df['price'] + items_df['freight_value']



# Aggregate payments

payments_sum = payments_df.groupby('order_id', as_index=False)['payment_value'].sum()

payments_sum.rename(columns={'payment_value': 'payment_total'}, inplace=True)

print("Basic transforms done.")



# Feature Engineering: Delivery Delay

orders_subset = orders_df[['order_id', 'order_purchase_timestamp', 'order_delivered_customer_date', 'order_estimated_delivery_date']]

reviews_subset = reviews_df[['order_id', 'review_score']]

orders_subset['order_purchase_timestamp'] = pd.to_datetime(orders_subset['order_purchase_timestamp'], errors='coerce')

orders_subset['order_delivered_customer_date'] = pd.to_datetime(orders_subset['order_delivered_customer_date'], errors='coerce')

orders_subset['order_estimated_delivery_date'] = pd.to_datetime(orders_subset['order_estimated_delivery_date'], errors='coerce')



delivery_df = orders_subset.merge(reviews_subset, on='order_id', how='left')

delivery_df['delivery_delay_days'] = (delivery_df['order_delivered_customer_date'] - delivery_df['order_estimated_delivery_date']).dt.days
```

```
delivery_df['delivery_time_days'] = (delivery_df['order_delivered_customer_date'] -
delivery_df['order_purchase_timestamp']).dt.days

print("Delivery and review metrics calculated.")



#
=====================================================================
=========

# BUILD MASTER TABLE (ETL - TRANSFORM)

#
=====================================================================
=========

print("\n--- Building Master Table... ---")

master = items_df.merge(

    orders_df[['order_id','customer_id','order_status','order_purchase_timestamp']],

    on='order_id', how='left'

).merge(

    products_df[['product_id','product_category_name']],

    on='product_id', how='left'

).merge(

    customers_df[['customer_id','customer_unique_id','customer_city','customer_state']],

    on='customer_id', how='left'
).merge(

    payments_sum, on='order_id', how='left'

).merge(

    delivery_df[['order_id', 'review_score', 'delivery_delay_days', 'delivery_time_days']],

    on='order_id', how='left'

)



# Add English translation

master = master.merge(

    translation_df[['product_category_name','product_category_name_english']],

    on='product_category_name',

    how='left'

)



# Fill missing payment totals with the item total

master['payment_total'] = master['payment_total'].fillna(master['total_item_amount'])



print(f"MASTER TABLE created. Shape: {master.shape}")
```

```
#
===========================================================================
=========

# DATA QUALITY CHECKS

#
===========================================================================
=========

print("\n--- Running Data Quality Checks... ---")

# Nulls report

null_report = master.isnull().sum().reset_index()

null_report.columns = ['column', 'null_count']

null_report['null_pct'] = (null_report['null_count'] / len(master) * 100).round(3)

null_report.to_csv('outputs/qc/null_report.csv', index=False)

print("Saved: outputs/qc/null_report.csv")


# Numeric stats

numeric_stats = master.select_dtypes(include=[np.number]).describe().transpose()

numeric_stats.to_csv('outputs/qc/numeric_stats.csv')

print("Saved: outputs/qc/numeric_stats.csv")
```

```
#
=============================================================================
=========

# DIAGNOSTIC PLOTTING & ANALYSIS

#
=============================================================================
=========

print("\n--- Generating Diagnostic Plots... ---")


# Helper to show and save plot

def save_and_show(fig, fname):

    fig.savefig(fname, bbox_inches='tight')

    print("Saved:", fname)

    plt.show()


# Ensure datetimes and derived fields

if 'order_purchase_timestamp' in master.columns:

        master['order_purchase_timestamp'] = pd.to_datetime(master['order_purchase_timestamp'],
errors='coerce')

    master = master.dropna(subset=['order_purchase_timestamp']).copy()

    master['order_date'] = master['order_purchase_timestamp'].dt.floor('D')

    master['order_month'] = master['order_purchase_timestamp'].dt.to_period('M').astype(str)
```

```
else:
    print("No 'order_purchase_timestamp' column; time plots will be skipped.")


# --- Monthly revenue ---
if 'order_month' in master.columns:
    monthly = master.groupby('order_month', as_index=False)['total_item_amount'].sum().rename(columns={'total_item_amount':'monthly_revenue'})
    monthly['month_start'] = pd.to_datetime(monthly['order_month'] + '-01')
    fig, ax = plt.subplots(figsize=(12,5))
    ax.plot(monthly['month_start'], monthly['monthly_revenue'], marker='o', linewidth=1)
    ax.set_title('Monthly Revenue Trend')
    ax.set_xlabel('Month'); ax.set_ylabel('Revenue'); ax.grid(alpha=0.2)
    save_and_show(fig, 'outputs/figures/monthly_revenue_trend.png')


# --- Weekly-summed revenue ---
if 'order_date' in master.columns:
    daily = master.groupby('order_date', as_index=False)['total_item_amount'].sum().rename(columns={'total_item_amount':'daily_revenue'})
    daily['order_date'] = pd.to_datetime(daily['order_date'])
    daily = daily.set_index('order_date').sort_index()

    weekly = daily.resample('7D').sum().reset_index()

    fig, ax = plt.subplots(figsize=(12,4))

    ax.plot(weekly['order_date'], weekly['daily_revenue'], marker='o', linewidth=1)

    ax.set_title('Weekly-summed Revenue (7-day buckets)')

    ax.set_xlabel('Date'); ax.set_ylabel('Revenue'); ax.grid(alpha=0.2)

    save_and_show(fig, 'outputs/figures/weekly_revenue_trend.png')


# --- Top categories by revenue ---

if 'product_category_name_english' in master.columns:

    cat_rev = master.groupby('product_category_name_english', as_index=False)['total_item_amount'].sum().rename(columns={'total_item_amount':'category_revenue'}).sort_values('category_revenue', ascending=False)

    fig, ax = plt.subplots(figsize=(10,6))

    sns.barplot(data=cat_rev.head(10), x='category_revenue', y='product_category_name_english', ax=ax)

    ax.set_title('Top 10 Categories by Revenue')

    save_and_show(fig, 'outputs/figures/top10_categories_revenue.png')


# --- Top states ---

if 'customer_state' in master.columns:
    state_rev = master.groupby('customer_state', as_index=False)['total_item_amount'].sum().rename(columns={'total_item_amount':'state_revenue'}).sort_values('state_revenue', ascending=False)

    fig, ax = plt.subplots(figsize=(10,5))

    sns.barplot(data=state_rev.head(10), x='state_revenue', y='customer_state', ax=ax)

    ax.set_title('Top 10 States by Revenue')

    save_and_show(fig, 'outputs/figures/top_states_revenue.png')


# --- Order status distribution ---

if 'order_status' in master.columns:

    order_status_counts = master['order_status'].value_counts().rename_axis('order_status').reset_index(name='count')

    fig, ax = plt.subplots(figsize=(9,4))

    sns.barplot(data=order_status_counts, x='order_status', y='count', order=order_status_counts['order_status'], ax=ax)

    ax.set_title('Order Status Distribution')

    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

    save_and_show(fig, 'outputs/figures/order_status_distribution.png')
```

```
#
================================================================================
=========

# DISPLAY SAVED PLOTS

#
================================================================================
=========

print("\n--- Displaying All Saved Plots: ---")

imgs = sorted(glob.glob("outputs/figures/*.png"))

html = "<div style='display:flex; flex-wrap:wrap; justify-content: center;'>"

print(f"Displaying {len(imgs)} saved images:")

for p in imgs:

    html += f"""

    <div style='margin:10px; border: 1px solid #ccc; padding: 5px; text-align:center;'>

        <img src='{p}' width=400>

        <div style='font-size:12px;'>{os.path.basename(p)}</div>

    </div>

    """

html += "</div>"

display(HTML(html))
```

```
# 
=====================================================================
=========
# EXPORT SUMMARY CSVs
# 
=====================================================================
=========
print("\n--- Exporting Summary Tables to CSV... ---")
if 'monthly' in locals():
    monthly.to_csv('outputs/tables/monthly_revenue.csv', index=False)
if 'weekly' in locals():
    weekly.to_csv('outputs/tables/weekly_revenue.csv', index=False)
if 'cat_rev' in locals():
    cat_rev.to_csv('outputs/tables/category_revenue.csv', index=False)
if 'state_rev' in locals():
    state_rev.to_csv('outputs/tables/state_revenue.csv', index=False)
if 'top_customers' not in locals() and 'customer_unique_id' in master.columns:
    top_customers = master.groupby('customer_unique_id', 
as_index=False)['total_item_amount'].sum().rename(columns={'total_item_amount':'customer_re
venue'}).sort_values('customer_revenue', ascending=False).head(10)
if 'top_customers' in locals():
    top_customers.to_csv('outputs/tables/top_customers.csv', index=False)

print("Saved CSV summaries in outputs/tables.")




#
==============================================================================
=========

# SAVE TO SQLITE DATABASE (ETL - LOAD)

#
==============================================================================
=========

print("\n--- Saving Analytical Tables to SQLite Database... ---")

DB_PATH = '/content/project3/data/ecommerce_analytics.db'

engine = create_engine(f"sqlite:///{DB_PATH}")


to_save = {}

if 'monthly' in locals(): to_save['monthly_sales'] = monthly

if 'cat_rev' in locals(): to_save['sales_by_category'] = cat_rev

if 'state_rev' in locals(): to_save['state_summary'] = state_rev

if 'top_customers' in locals(): to_save['top_customers'] = top_customers

if 'weekly' in locals(): to_save['weekly_revenue'] = weekly
```

```
# Save the main master table as well
to_save['fact_orders'] = master


for name, df in to_save.items():
    try:
        df.to_sql(name, engine, if_exists='replace', index=False)
        print(f"Saved table {name} ({len(df)} rows)")
    except Exception as e:
        print(f"Could not save {name} -> {e}")


with engine.connect() as conn:
    res = conn.execute(text("SELECT name FROM sqlite_master WHERE type='table';"))
    tables = [r[0] for r in res.fetchall()]
print(f"Database created at: {DB_PATH}")
print("DB tables now:", tables)



#
==============================================================================
=========
# MACHINE LEARNING MODEL (PREDICT DELAY)

#
==============================================================================
=========

print("\n--- Training Machine Learning Model... ---")

clf = None # Initialize clf

X_test = None # Initialize X_test


if 'delivery_delay_days' not in master.columns:

    print("No 'delivery_delay_days' column - skipping ML.")

else:

    df_ml = master.dropna(subset=['delivery_delay_days', 'delivery_time_days',
'review_score']).copy()

    df_ml['delayed'] = (df_ml['delivery_delay_days'] > 3).astype(int) # Target: delayed by more
than 3 days


    # Define features and target

    features = [

        'price', 'freight_value', 'total_item_amount',

        'payment_total', 'review_score', 'delivery_time_days'

    ]
    target = 'delayed'

    # Ensure all feature columns exist and are numeric
    valid_features = [f for f in features if f in df_ml.columns and pd.api.types.is_numeric_dtype(df_ml[f])]

    if len(valid_features) < 3:
        print(f"Not enough numeric features for ML model. Found: {valid_features}")
    else:
        print(f"Using features: {valid_features}")
        X = df_ml[valid_features].fillna(0)
        y = df_ml[target]

        X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

        clf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1, max_depth=10) # Reduced estimators/depth for speed
        clf.fit(X_train, y_train)

        y_pred = clf.predict(X_test)
        # --- Save Reports ---

        report = classification_report(y_test, y_pred, output_dict=True)

        pd.DataFrame(report).transpose().to_csv('outputs/tables/delay_model_report.csv')



        cm = confusion_matrix(y_test, y_pred)

        plt.figure(figsize=(4,3));

         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Delayed', 'Delayed'],
    yticklabels=['Not Delayed', 'Delayed'])

        plt.title('Confusion Matrix - Delay Classifier')

        plt.xlabel('Predicted'); plt.ylabel('Actual')

        plt.tight_layout()

        plt.savefig('outputs/figures/delay_confusion_matrix.png')

        plt.close() # Close plot to prevent double-display



        print("Saved ML report and confusion matrix.")



        # --- Feature importances ---

                    fi = pd.DataFrame({'feature': X.columns, 'importance':
    clf.feature_importances_}).sort_values('importance', ascending=False).head(10)

        fi.to_csv('outputs/tables/delay_feature_importance.csv', index=False)
```

```
        print("Saved feature importances.")
        print(fi)



#
=====================================================================
=========
# MODEL EXPLAINABILITY (SHAP)
#
=====================================================================
=========
print("\n--- Starting Model Explainability (SHAP) ---")


if 'clf' in locals() and clf is not None and 'X_test' in locals() and X_test is not None:
    try:
        explainer = shap.TreeExplainer(clf)


        # Sample the test data if it's large
        sample_size = min(1000, len(X_test))
        X_test_sample = X_test.sample(sample_size, random_state=42)
        shap_values = explainer.shap_values(X_test_sample)

        print("SHAP values calculated.")

        # Create the summary plot (beeswarm)
        print("--- SHAP Summary Plot (Beeswarm) ---")
        shap.summary_plot(shap_values[1], X_test_sample, show=False)
        plt.title("Feature Impact on Predicting DELAY (Class 1)")
        plt.savefig('outputs/figures/shap_delay_beeswarm.png', bbox_inches='tight')
        plt.show()

    except Exception as e:
        print(f"SHAP analysis failed: {e}")
else:
    print("--- SKIPPING SHAP ---")
    print("The model 'clf' or data 'X_test' is not defined. Please ensure the Machine Learning cell ran successfully.")
```

```
#
=====================================================================
=========
# FINAL SQL QUERY & PLOT
#
=====================================================================
=========
print("\n--- Running Final SQL Query from Database... ---")
query = """
SELECT product_category_name_english, category_revenue
FROM sales_by_category
ORDER BY category_revenue DESC
LIMIT 5;
"""
try:
    top5 = pd.read_sql(query, engine)
    print("Top 5 Categories from SQL:")
    print(top5)

    plt.figure(figsize=(10,6))
    sns.barplot(data=top5, x='category_revenue', y='product_category_name_english')
    plt.title("Top 5 Categories by Revenue (from SQLite DB)")
    plt.savefig('outputs/figures/sql_top5_categories.png', bbox_inches='tight')
    plt.show()
except Exception as e:
    print(f"Could not query SQL database: {e}")



#
=============================================================================
==========
# CREATE AND DOWNLOAD ZIP FILE
#
=============================================================================
==========
print("\n--- Creating Project ZIP File... ---")
root = '/content/project3'
zip_path = '/content/project3_output.zip'


try:
    with zipfile.ZipFile(zip_path, 'w', zipfile.ZIP_DEFLATED) as zf:
        for foldername, subfolders, filenames in os.walk(root):
        for filename in filenames:

            filepath = os.path.join(foldername, filename)

            zf.write(filepath, os.path.relpath(filepath, '/content'))

    print(f"ZIP created at {zip_path}, size (MB): {os.path.getsize(zip_path)/1024/1024:.2f}")



    # Download in Colab

    files.download(zip_path)

except Exception as e:

    print(f"ZIP/Download failed: {e}")



print("\n--- PROJECT COMPLETE ---")
```