E-Commerce Sales Data – Technical Documentation Report

1. Introduction

This report provides a comprehensive overview of the ETL pipeline, data cleaning methodologies, design rationale, and analytical assumptions employed in the examination of e-commerce sales data. It includes well-documented code for each phase of the Extract, Transform, Load (ETL) process and presents key business insights derived from Power BI visualizations.

2. ETL Pipeline

2.1 Extract (Data Ingestion)

- Data Source:
  The dataset originates from a leading online retailer specializing in home appliances and electronics.
  The data spans transactions from April 2020 to November 2020.
- Technologies Utilized:
  - Power BI for data visualization
  - Pandas (Python) for data preprocessing and cleaning

Code Snippet: Data Extraction Using Pandas

import pandas as pd

```
import pandas as pd
import psycopg2

# Load dataset (en/sure correct file path)
df = pd.read_csv("kz.csv")

# Check dataset structure
print(f"Total Rows: {df.shape[0]}")
print(df.head())
print(df.info())
```

2.2 Transform (Data Cleaning & Processing)

- Data Cleaning Process:
  Handling Missing Values:

- `category_code`, `brand`, and `price` had missing values, which were either filled or removed based on significance.

- Datetime Conversion:
  - The `event_time` column was converted to a DateTime format to enable accurate time-series analysis.
- Duplicate Removal:
  - Eliminated redundant records to maintain data integrity.
- Column Standardization:
  - Column names were standardized for readability and consistency.

Code Snippet: Data Cleaning

```
# Check missing values
print("Missing Values Before Handling:\n", df.isnull().sum())
# Remove duplicate rows and reset index
print(f"Duplicate Rows Before: {df.duplicated().sum()}")
df = df.drop_duplicates().reset_index(drop=True)
print(f"Duplicate Rows After: {df.duplicated().sum()}")
# Convert event_time to datetime
df["event_time"] = pd.to_datetime(df["event_time"])
# Convert user_id safely to integer
df["user_id"] = pd.to_numeric(df["user_id"], errors="coerce").fillna(-1).astype("Int64")
```

```
# Handle missing values
df = df.assign(
    category_code=df["category_code"].fillna("Unknown"),
    brand=df["brand"].fillna("Unknown"),
    price=df["price"].fillna(df["price"].median()),
    category_id=df["category_id"].fillna(df["category_id"].mode().iloc[0] if not df["category_id"].mode().empty else -1),
    user_id=df["user_id"].fillna(-1),
)
```

```
# Verify missing values are handled
print("Missing Values After Handling:\n", df.isnull().sum())
# Save cleaned dataset to the specified folder
output_path = "cleaned_dataset.csv"
df.to_csv(output_path, index=False)
print(f"✅ Cleaned dataset saved successfully at {output_path}!")
# Save cleaned dataset
df.to_csv("cleaned_dataset.csv", index=False)
print("✅ Dataset cleaned and saved successfully!")
```

2.3 Load (Data Storage & Integration)

```
# Database connection parameters
DB_NAME = "ecommerce_db"
DB_USER = "postgres"  # Default username
DB_PASSWORD = "kal@12"  # Replace with your actual password
DB_HOST = "localhost"  # Use "127.0.0.1" if needed
DB_PORT = "5432"  # Default PostgreSQL port

try:
    conn = psycopg2.connect(
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASSWORD,
        host=DB_HOST,
        port=DB_PORT
    )
    cursor = conn.cursor()
    print("✅ Connected to PostgreSQL successfully!")

except Exception as e:
    print("❌ Connection failed:", e)
```

```
create_table_query = """
CREATE TABLE IF NOT EXISTS ecommerce_data (
    event_time TIMESTAMP,
    order_id BIGINT PRIMARY KEY,
    product_id BIGINT,
    category_id BIGINT,
    category_code TEXT,
    brand TEXT,
    price NUMERIC(10,2),
    user_id BIGINT
);
"""
# Execute create table query
cursor.execute(create_table_query)
conn.commit()
print("✅ Table 'ecommerce_data' created successfully!")
```

```
# ------------------------------ #
# ✅ Insert Data into PostgreSQL
# ------------------------------ #

# Convert data types to ensure compatibility
df["event_time"] = pd.to_datetime(df["event_time"])  # Convert to datetime
df["order_id"] = df["order_id"].astype("Int64")  # Convert to integer
df["product_id"] = df["product_id"].astype("Int64")
df["category_id"] = df["category_id"].fillna(-1).astype("Int64")  # Fill NaN with -1
df["category_code"] = df["category_code"].astype(str)  # Ensure it's a string
df["brand"] = df["brand"].astype(str)  # Ensure it's a string
df["price"] = df["price"].astype(float)  # Ensure numeric type
df["user_id"] = df["user_id"].fillna(-1).astype("Int64")  # Fill NaN with -1
```

```
# Insert data row by row (Handling NULL values)
for _, row in df.iterrows():
    try:
        cursor.execute("""
            INSERT INTO ecommerce_data (event_time, order_id, product_id, category_id, category_code, brand, price, user_id)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            ON CONFLICT (order_id) DO NOTHING;
        """, (
            row["event_time"].strftime('%Y-%m-%d %H:%M:%S') if pd.notna(row["event_time"]) else None,
            int(row["order_id"]) if pd.notna(row["order_id"]) else None,
            int(row["product_id"]) if pd.notna(row["product_id"]) else None,
            int(row["category_id"]) if pd.notna(row["category_id"]) else None,
            row["category_code"] if pd.notna(row["category_code"]) else "Unknown",
            row["brand"] if pd.notna(row["brand"]) else "Unknown",
            float(row["price"]) if pd.notna(row["price"]) else 0.0,
            int(row["user_id"]) if pd.notna(row["user_id"]) else None
        ))
    except Exception as e:
        print(f"❌ Error inserting row: {row}")
        print("Error:", e)

# Commit changes and close connection
conn.commit()
cursor.close()
conn.close()
print("✅ Data successfully loaded into PostgreSQL!")
```

Storage and Integration Strategy:

The cleaned dataset was imported into Power BI for visualization and analysis.

- Logical relationships were established between `order_id`, `user_id`, and `product_id` to facilitate structured queries.

- DAX (Data Analysis Expressions) measures were utilized to compute key business metrics.

Steps to Load Data into Power BI:
1. Open Power BI and navigate to 'Get Data'.
2. Select 'CSV' (or appropriate database connection) and import the cleaned dataset.
3. Validate data types for key attributes (`event_time` as Date/Time, `price` as Currency, etc.).

3. Data Schema & Relationships

Schema Rationale:

Column Name	Data Type	Description
event_time	Date/Time	Timestamp of the purchase event
order_id	integer	Unique ID for each order
product_id	Integer	Unique ID for each product purchased
category_id	Integer	Identifier for the product category
category_code	String	Name of the product category (e.g., electronics, apparel)
brand	String	Brand of the purchased product
price	Float	Price of the product
user_id	Integer	Unique identifier for the customer

Why a Single Table?

Since all purchase-related details (orders, products, customers, and brands) are stored in **one table**, there was **no need to create relationships** between multiple tables.

How This Affects Power BI?

- **Easier Filtering:** Since everything is in one table, slicers and filters can directly access all fields.
- **More Efficient Data Modeling:** We don't need to join tables, making queries and calculations faster.

◇ Power BI Implementation

- Even with a **single table**, we optimized our Power BI model by:
  Using **DAX measures** instead of calculated columns to improve performance
- Applying **sort and rank functions** to analyze top-performing brands and categories.

5. Key Findings & Insights

Sales Performance Analysis:

- Total Revenue: Over $22 million in recorded sales.
- Peak Sales Periods: November and December exhibited the highest transaction volumes, highlighting strong holiday demand.
- Recommendation: Intensify marketing campaigns and promotional discounts during Q4 to maximize sales.

6. Power BI Dashboard Features

Implemented Visuals:

- KPI Cards: Total Sales, Total Orders, Total Customers, AOV.
- Line Chart: Sales trends over time.
- Bar Chart: Top five brands ranked by revenue.
- Pie Chart: Sales distribution by product category.
- Data Table: Detailed brand revenue rankings.

7. Conclusion & Next Steps

Summary of Findings:

- Electronics dominate sales, emphasizing the importance of high-margin products.
- Samsung & Apple are key drivers of revenue, necessitating continued brand collaboration.
- The average transaction value of $307 suggests a demand for premium products.
- Sales exhibit strong seasonality, with Q4 showing significant peaks.