



***DEBRE BERHAN UNIVERSITY***

***COLLEGE OF COMPUTING***

***DEPARTMENT OF SOFTWARE ENGINEERING***

***PREPARED BY : NATNAEL HABTE***

***ID : 0593/13***

***ASSIGNMENT TITLE : BUILDING AN END-TO-END DATA PIPELINE***

***SUBMITTED TO : DERBEW FELASMAN(MSc)***

***SUBMISSION DATE : FEBRUARY 13 /2025***

***DEBRE BERHAN ,ETHIOPIA***

# 1 Overview

This ETL pipeline takes raw e-commerce transaction data in a CSV file, cleans and transforms it, and loads it into a PostgreSQL database. The data is then used for analysis and visualization in Power BI.

- Data Source: kz.csv (E-commerce transactions)
- Processing: Python (pandas, sqlalchemy, psycopg2)
- Database: PostgreSQL
- Visualization: Power BI

## 2 Extract: Loading Raw Data

### 2.1 Source Data

The raw data comes from a CSV file (kz.csv) containing e-commerce transactions.

### 2.2 Code: Loading CSV File

```
1  import pandas as pd
2
3  # File paths
4  file_path = "kz.csv"
5  output_file = "transformed_kz.csv"
6
7  try:
8      # Load raw data
9      df = pd.read_csv(file_path, low_memory=False)
10     print(f"Successfully loaded {file_path}")
11 except Exception as e:
12     print(f"Error loading {file_path}: {e}")
13     exit()
14
```

Findings:

- The dataset contains duplicates and missing values.
- Some fields like price contain invalid values (non-numeric data).
- event\_time is not properly formatted.

### 3 Transform: Data Cleaning & Processing

#### 3.1 Design Choices & Cleaning Steps

Cleaning Step	Purpose
Remove Duplicates	Ensure unique orders and users
Handle Missing Values	Prevent data inconsistencies
Convert Price to Numeric	Standardize financial data
Standardize Text Columns	Ensure consistency in categorical data
Format DateTime	Ensure proper timestamp handling

## 3.2 Code: Cleaning and Transforming Data

```
1  # Remove duplicate orders
2  df.drop_duplicates(subset=["order_id"], keep="first", inplace=True)
3
4  # Ensure user_id is unique
5  df.drop_duplicates(subset=["user_id"], keep="first", inplace=True)
6
7  # Handle missing values
8  df["price"] = pd.to_numeric(df["price"], errors='coerce').fillna(0.0) # Replace invalid prices with 0.0
9  df["category_id"] = df["category_id"].fillna("unknown")
10 df["category_code"] = df["category_code"].fillna("unknown")
11 df["brand"] = df["brand"].fillna("unknown")
12 df["event_time"] = pd.to_datetime(df["event_time"], errors='coerce').fillna(pd.Timestamp("1970-01-01"))
13
14 # Standardize text columns
15 df["category_code"] = df["category_code"].astype(str).str.lower()
16 df["brand"] = df["brand"].astype(str).str.lower()
17
18 # Save the transformed data
19 df.to_csv(output_file, index=False)
20 print(f"\nTransformed data saved to {output_file}")
21
```

Findings:

- Missing values were handled appropriately.
- The dataset contained duplicate user\_id values, which were removed to maintain uniqueness.

## 4 Load: Storing Data in PostgreSQL

### 4.1 Database Schema

```
1  ✓ CREATE TABLE IF NOT EXISTS ecommerce_transactions (  
2      user_id VARCHAR(50) PRIMARY KEY,  
3      order_id VARCHAR(50),  
4      product_id VARCHAR(50),  
5      category_id TEXT,  
6      category_code TEXT,  
7      brand TEXT,  
8      price NUMERIC,  
9      event_time TIMESTAMP  
10 );  
11
```

Design Choice:

- order\_id is the primary key to ensure each user appears only once in the database.

### 4.2 Code: Loading Data into PostgreSQL

```
1  from sqlalchemy import create_engine  
2  import psycopg2  
3  
4  # Database connection details  
5  DB_USER = "postgres"  
6  DB_PASS = "nathab"  
7  DB_HOST = "localhost"  
8  DB_PORT = "3000"  
9  DB_NAME = "ecommerce_db"  
10  
11  ✓ try:  
12      # Connect to PostgreSQL  
13      conn = psycopg2.connect(  
14          dbname=DB_NAME, user=DB_USER, password=DB_PASS, host=DB_HOST, port=DB_PORT  
15      )  
16      cursor = conn.cursor()  
17      print("\nConnected to PostgreSQL successfully!")  
18
```

```

19     # Create table
20     create_table_query = """
21     CREATE TABLE IF NOT EXISTS ecommerce_transactions (
22         user_id VARCHAR(50) PRIMARY KEY,
23         order_id VARCHAR(50),
24         product_id VARCHAR(50),
25         category_id TEXT,
26         category_code TEXT,
27         brand TEXT,
28         price NUMERIC,
29         event_time TIMESTAMP
30     );
31     """
32     cursor.execute(create_table_query)
33     conn.commit()
34     print("Table checked/created successfully!")
35
36     # Load data
37     engine = create_engine(f"postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}")
38     df.to_sql("ecommerce_transactions", con=engine, if_exists="append", index=False)
39     print("Data successfully loaded into PostgreSQL")
40
41     except Exception as e:
42         print(f"Error loading data into PostgreSQL: {e}")
43
44     finally:
45         cursor.close()
46         conn.close()
47         print("Database connection closed.")

```

## Findings:

- Unique constraint errors were encountered due to duplicate user\_id values.
- These were resolved by ensuring user\_id is unique before insertion.

The screenshot shows a data tool interface with a sidebar on the left containing a tree view of database objects. The main area displays a table of data from a PostgreSQL database. The table has 7 columns: order\_id, product\_id, category\_id, category\_code, brand, and price. The data is filtered to show 17 rows out of 1000.

	order_id [PK] character varying (50)	product_id character varying (50)	category_id text	category_code text	brand text	price number
1	2294359932054536986	1515966223509089906	2.268105426648171e+18	electronics.tablet	samsung	1
2	2294444024058086220	2273948319057183658	2.2681054301629978e+18	electronics.audio.headphone	huawei	
3	2294584263154074236	2273948316817424439	2.26810547136784e+18	unknown	karcher	2
4	2295716521449619559	1515966223509261697	2.268105442636858e+18	furniture.kitchen.table	maestro	
5	2295740594749702229	1515966223509104892	2.268105428166509e+18	electronics.smartphone	apple	13
6	2295902490203259134	2273948311742316796	2.268105393848714e+18	appliances.kitchen.refrigerators	lg	4
7	2296164324487463110	1515966223509259473	2.2681054024470374e+18	appliances.personal.scales	polaris	
8	2296400480990920715	2273948308663698152	2.3744989140005924e+18	electronics.video.tv	samsung	4
9	2296628237930857206	1515966223509089660	2.2681054100219494e+18	computers.components.cpu	intel	
10	2296972701060825130	1515966223509104683	2.2681054027741932e+18	unknown	philips	
11	2297016008231092565	1515966223509089780	2.2681054072201554e+18	computers.notebook	asus	5
12	2297034737199350540	1515966223509719628	2.2681056355077325e+18	unknown	unknown	
13	2297174044555871159	2273948222957290212	2.2681054092250317e+18	computers.peripherals.monitor	samsung	2
14	2297252054407578606	2273948303177548033	2.268105407933187e+18	computers.peripherals.printer	epson	1
15	2297729407910937541	1515966223509105105	2.2681054275289748e+18	unknown	sbs	
16	2297770405059888020	1515966223509088578	2.268105428166509e+18	electronics.smartphone	samsung	
17	2297817716758675935	1515966223510177666	2.2681054422425935e+18	unknown	geyzer	

Total rows: 1000 of 1435266    Query complete 00:00:05.938    Ln 1, Col 1

## 5 Power BI Integration

### 5.1 Connecting PostgreSQL to Power BI

Steps:

1. Go to Home → Get Data → Database → PostgreSQL
2. Enter connection details:
  - Server: localhost
  - Database: ecommerce\_db
  - Username: postgres
  - Password: \*\*\*\*\*
3. Click Load or Transform Data if preprocessing is needed.

## **6 Data Visualization in Power BI**

### **6.1 Creating Scatter Plot (Similar to Python)**

To replicate the Python scatter plot in Power BI:

Steps:

1. Drag price to the X-axis
2. Drag qty\_ordered to the Y-axis
3. Drag category\_name\_1 to the legend (color differentiation)
4. Set Scatter Chart as the visualization type.

## **7 Summary & Key Learnings**

### **7.1 Project Achievements**

- Extracted raw data from CSV.
- Cleaned & transformed data (handled missing values, duplicates, standardization).
- Stored data in PostgreSQL, ensuring user\_id is unique.
- Connected PostgreSQL to Power BI for analysis.

### **7.2 Future Improvements**

- Optimize performance by indexing frequently queried columns.
- Enhance error handling for database operations.
- Implement automated ETL pipeline for continuous data updates.