Debre Birhan University

Collage Of Computing

Department OF Software engineering

Fundamental OF Big Data Analytics And BI

End To End Pipeline

Prepared by - Surafel G/selassie
Id- 0430/13

Submited To - Derbew Felasman (MSC)
Submition Date - Feb - 14 - 2025 GC.

Github Link -
https://github.com/0001sura/bigdata.git

Table of content

ETL Pipeline Documentation

Introduction

An ETL (Extract, Transform, Load) pipeline is essential for processing and managing large datasets efficiently. This documentation provides an in-depth guide to an ETL pipeline designed to process an e-commerce dataset. The pipeline automates data extraction, transformation, and loading into a PostgreSQL database.

Objective

The primary objective of this ETL pipeline is to:

1. Extract raw data from CSV files.
2. Clean and transform data to ensure consistency and accuracy.
3. Load processed data into a PostgreSQL database for storage and further analysis.
4. Automate the ETL process for efficiency and scalability.

Prerequisites

Before running the ETL pipeline, ensure that you have the following dependencies installed:

```
pip install pandas sqlalchemy psycopg2
```

System Requirements

- **Operating System**: Windows, macOS, or Linux
- **Python Version**: 3.7 or higher
- **Database**: PostgreSQL 12 or higher
- **Memory Requirement**: At least 4GB RAM for large datasets

ETL Process

1. Data Extraction and Transformatio(clean_and_transform())

a) Load Raw Data

The raw dataset is read from:

data/raw/Pakistan Largest Ecommerce Dataset.csv

The dataset contains date columns which are parsed during loading.

b) Handling Missing Values

- **Categorical Columns:**
  - sales_commission_code: Missing values are replaced with 'Unknown'.
  - BI Status: '#REF!' values are replaced with 'Unknown', and missing values are also set to 'Unknown'.
- **Numerical Columns:**
  - Columns: price, grand_total, discount_amount, qty_ordered
  - Converted to numeric types; invalid values are coerced to NaN.
  - Missing values are replaced with the median for robustness.

c) Data Cleaning

- **Duplicate Removal:** All duplicate rows are dropped.
- **Date Columns Handling:** created_at, Working Date, and M-Y are converted to datetime format.

d) Dropping Unnecessary Columns

The following columns are removed:

MV, Year, Month, Customer Since, Unnamed: 19-25

2. Load Data to PostgreSQL (load_to_postgres(cleaned_df))

a) Database Connection

The function connects to a PostgreSQL database using:

```
postgresql://postgres:1221@localhost:5432/bigdata
```

b) Table Creation

- A table named ecommerce_data is created.
- Only the schema (column names and types) is written initially.

c) Data Insertion

- Data is inserted in chunks of 10,000 rows for efficiency.
- Existing data is preserved, and new data is appended.

3. Execution

The script is executed by running:

```
if __name__ == "__main__":
    df = clean_and_transform()
    load_to_postgres(df)
```

Error Handling

- If the CSV file is missing or corrupted, an appropriate error message is displayed.
- If database credentials are incorrect, the script terminates with a connection error message.
- If column names in the dataset change, the script will log an error and halt execution.

Implementatoin (Code)

1. Import the necessary files

```
import pandas as pd
from sqlalchemy import create_engine
```

2. Load the data

```
def clean_and_transform():
    # Load raw data
    raw_df = pd.read_csv(
        'data/raw/Pakistan Largest Ecommerce Dataset.csv',
        low_memory=False,
        parse_dates=['created_at', 'Working Date', 'M-Y'],
        infer_datetime_format=True
    )
```

3. Data Cleaning And Transformation

```
    # --- Null Handling ---
    # Categorical columns
    raw_df['sales_commission_code'] =
raw_df['sales_commission_code'].fillna('Unknown')
    raw_df['BI Status'] = raw_df['BI Status'].replace('#REF!',
'Unknown').fillna('Unknown')

    # Numerical columns
    num_cols = ['price', 'grand_total', 'discount_amount', 'qty_ordered']
    for col in num_cols:
        raw_df[col] = pd.to_numeric(raw_df[col], errors='coerce')
        raw_df[col] = raw_df[col].fillna(raw_df[col].median())

    # Remove duplicates
    raw_df = raw_df.drop_duplicates()

       date_cols = ['created_at', 'Working Date', 'M-Y']
    for col in date_cols:
        raw_df[col] = pd.to_datetime(raw_df[col], errors='coerce')

    # --- Column Removal ---
    cols_to_drop = [
        'MV', 'Year', 'Month', 'Customer Since',
        'Unnamed: 19', 'Unnamed: 20', 'Unnamed: 21',
        'Unnamed: 22', 'Unnamed: 23', 'Unnamed: 24', 'Unnamed: 25'
    ]
    return raw_df.drop(columns=[c for c in cols_to_drop if c in raw_df.columns])
```

4. Load To POStgreSQL

```
def load_to_postgres(cleaned_df):
    # Database connection
    engine = create_engine('postgresql://postgres:1221@localhost:5432/bigdata')

      cleaned_df.head(0).to_sql('ecommerce_data', engine, if_exists='replace',
index=False)

    # Load in chunks (for large datasets)
    cleaned_df.to_sql('ecommerce_data', engine, if_exists='append', index=False,
chunksize=10000)
    print("Data successfully loaded to PostgreSQL!")
    print(cleaned_df.shape)


if name == "main":
    df = clean_and_transform()
   load_to_postgres(df)
```

Creating Table

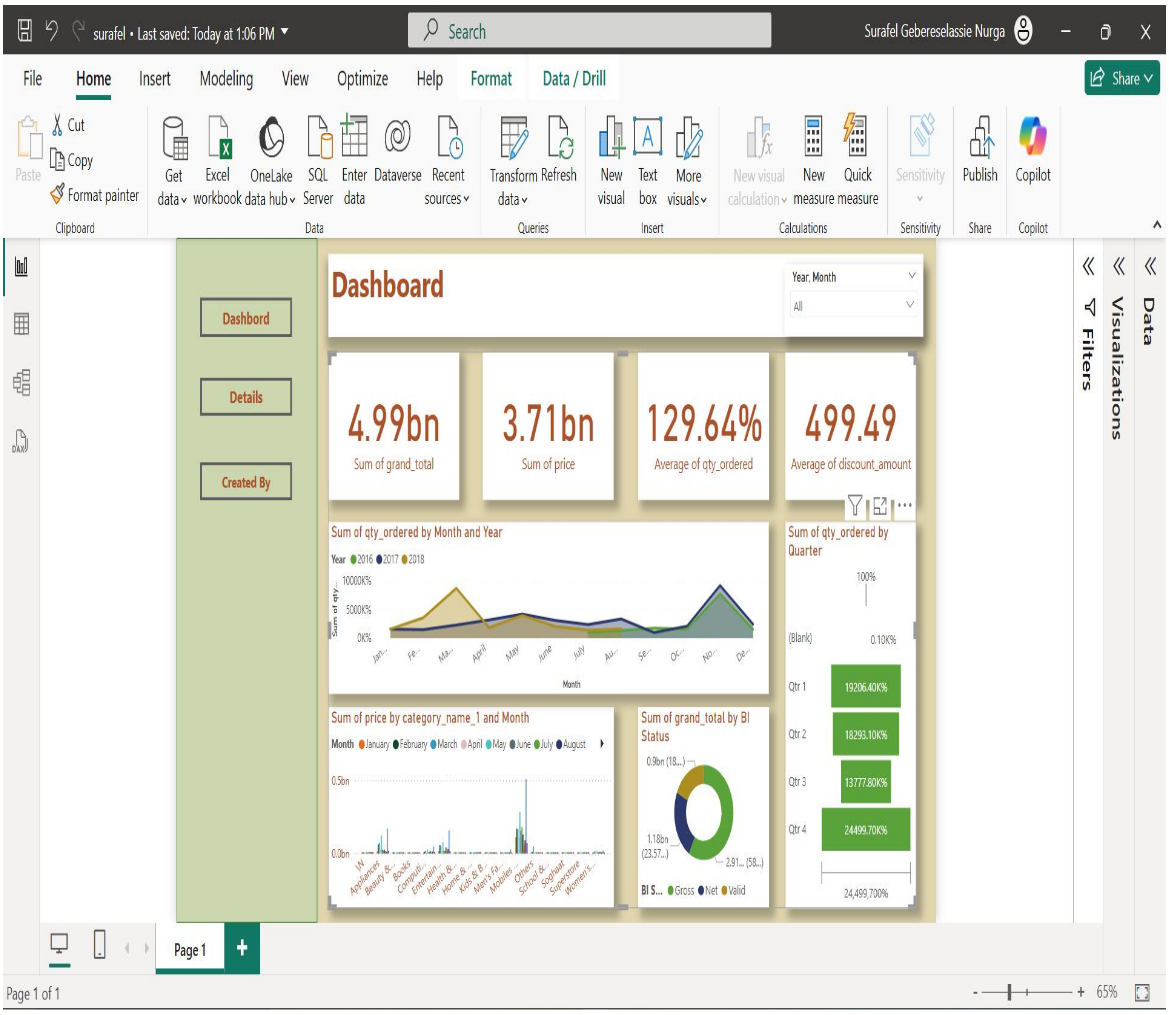
```
CREATE TABLE ecommerce_data (
    item_id BIGINT,
    status VARCHAR(50),
    created_at TIMESTAMP,
    sku VARCHAR(100),
    price FLOAT,
    qty_ordered INT,
    grand_total FLOAT,
    increment_id VARCHAR(50),
    category_name_1 VARCHAR(100),
    sales_commission_code VARCHAR(100),
    discount_amount FLOAT,
    payment_method VARCHAR(50),
    working_date TIMESTAMP,
    bi_status VARCHAR(50),
    m_y TIMESTAMP
);
```

Visualization (Dashboard)

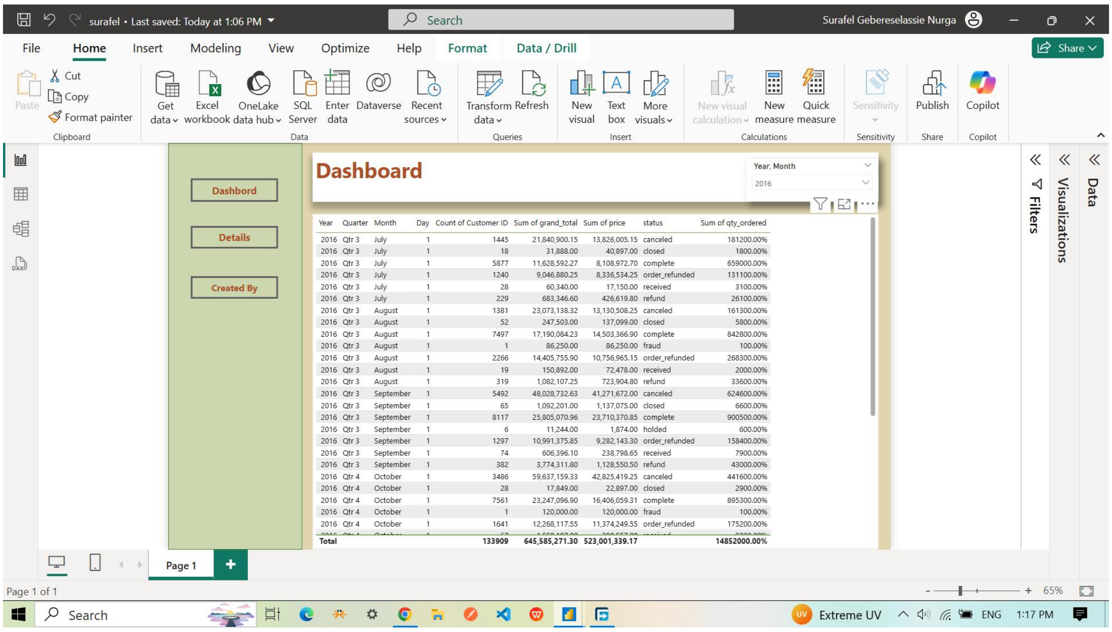
Link to the visualization(powerbi)

https://app.powerbi.com/links/6W9FCQRh4j?ctid=1695066a-e388-40d1-8ed5-5d0b28ba9f80&pbi_source=linkShare

1. Dashboard page



2. Details Page



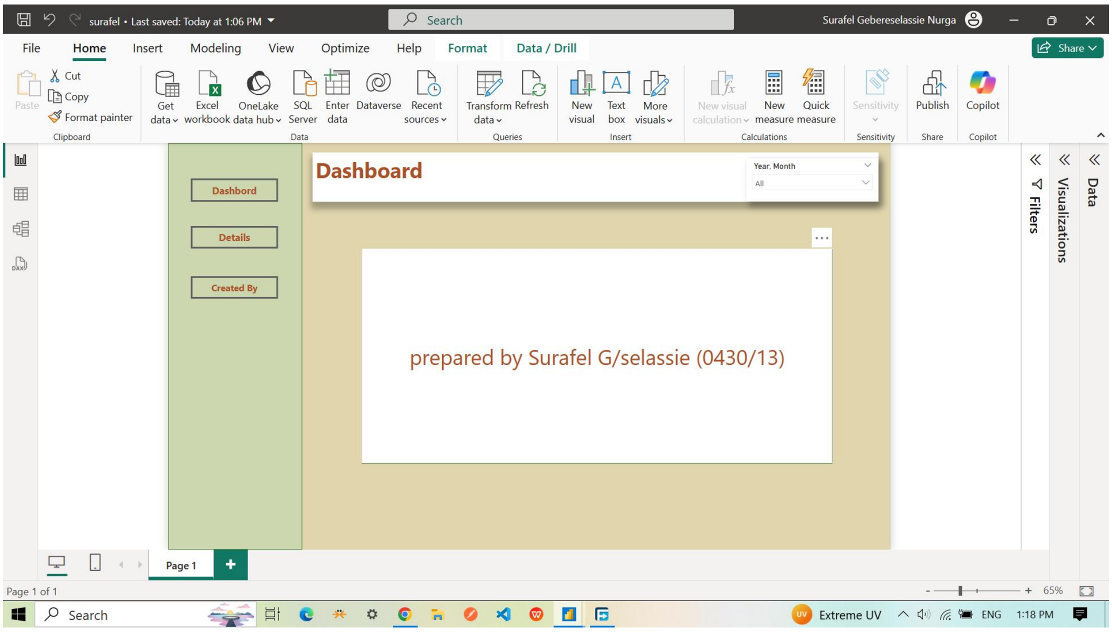
Dashboard

Year, Month

2016

Year	Quarter	Month	Day	Count of Customer ID	Sum of grand_total	Sum of price	status	Sum of qty_ordered
2016	Qtr 3	July	1	1445	21,840,900.15	13,826,005.15	canceled	181200.00%
2016	Qtr 3	July	1	18	31,888.00	40,897.00	closed	1800.00%
2016	Qtr 3	July	1	5877	11,628,592.27	8,108,972.70	complete	659000.00%
2016	Qtr 3	July	1	1240	9,046,880.25	8,336,534.25	order_refunded	131100.00%
2016	Qtr 3	July	1	28	60,340.00	17,150.00	received	3100.00%
2016	Qtr 3	July	1	229	683,346.60	426,619.80	refund	26100.00%
2016	Qtr 3	August	1	1381	23,073,138.32	13,130,508.25	canceled	161300.00%
2016	Qtr 3	August	1	52	247,503.00	137,099.00	closed	5800.00%
2016	Qtr 3	August	1	7497	17,190,084.23	14,503,366.90	complete	842800.00%
2016	Qtr 3	August	1	1	86,250.00	86,250.00	fraud	100.00%
2016	Qtr 3	August	1	2266	14,405,755.90	10,756,965.15	order_refunded	268300.00%
2016	Qtr 3	August	1	19	150,892.00	72,478.00	received	2000.00%
2016	Qtr 3	August	1	319	1,082,107.25	723,904.80	refund	33600.00%
2016	Qtr 3	September	1	5492	48,028,732.63	41,271,672.00	canceled	624600.00%
2016	Qtr 3	September	1	65	1,092,201.00	1,137,075.00	closed	6600.00%
2016	Qtr 3	September	1	8117	25,805,070.96	23,710,370.85	complete	900500.00%
2016	Qtr 3	September	1	6	11,244.00	1,874.00	holded	600.00%
2016	Qtr 3	September	1	1297	10,991,375.85	9,282,143.30	order_refunded	158400.00%
2016	Qtr 3	September	1	74	606,396.10	238,798.65	received	7900.00%
2016	Qtr 3	September	1	382	3,774,311.80	1,128,550.50	refund	43000.00%
2016	Qtr 4	October	1	3486	59,637,159.33	42,825,419.25	canceled	441600.00%
2016	Qtr 4	October	1	28	17,849.00	22,897.00	closed	2900.00%
2016	Qtr 4	October	1	7561	23,247,096.90	16,406,059.31	complete	895300.00%
2016	Qtr 4	October	1	1	120,000.00	120,000.00	fraud	100.00%
2016	Qtr 4	October	1	1641	12,268,117.55	11,374,249.55	order_refunded	175200.00%
Total				133909	645,585,271.30	523,001,339.17		14852000.00%

3. Prepaerd by page



Dashboard

Year, Month

All

prepared by Surafel G/selassie (0430/13)

Performance Optimization

- **Batch Processing**: Data is inserted into PostgreSQL in chunks to handle large datasets efficiently.
- **Indexing**: Indexing key columns in the database improves query performance.
- **Logging**: Implementing logging for monitoring ETL operations.

Security Considerations

- **Database Credentials**: Store credentials securely using environment variables.
- **Data Privacy**: Ensure sensitive customer data is anonymized if necessary.
- **Access Control**: Restrict database access to authorized users only.

Future Enhancements

- **Automated Scheduling**: Implementing cron jobs or Airflow for scheduled ETL runs.
- **Data Validation**: Adding data validation checks before inserting into the database.

Notes

- Ensure PostgreSQL is running and accessible at localhost:5432.
- Update database credentials as needed.
- Modify cols_to_drop if additional columns should be removed.

Conclusion

This ETL pipeline automates data cleaning and loading into a PostgreSQL database, making it efficient for processing large e-commerce datasets. With improvements in performance optimization and security, the pipeline can scale for larger datasets and enterprise applications.