DEBRE BERHAN UNIVERSITY

College of Computing

Department of Software Engineering

Fundamental OF Big Data Analytics And BI
End To End Pipeline

Course Code:SEng5112

Prepared by:

Yohanes Tamirat………DBUR/0786/13

Githublink:[click Here]

Submited To - Derbew Felasman (MSC)

Submition Date - Feb - 14 - 2025 GC.

Table of Contents

1.Introduction

This project focuses on building an end-to-end ETL pipeline for an e-commerce dataset using Python, PostgreSQL, and Power BI. The pipeline extracts data from raw CSV files, performs cleaning and transformation, loads it into a PostgreSQL database, and finally visualizes the insights using Power BI dashboards.By implementing this pipeline, businesses can gain valuable insights into sales performance, customer behavior, product trends, and operational efficiency. The final dashboard provides an intuitive way to analyze key metrics, detect patterns, and make data-driven decisions.

This documentation details the entire ETL workflow, including data extraction, cleaning, transformation, and loading into PostgreSQL, along with error handling, security considerations, and future enhancements. Additionally, it covers how the cleaned data is visualized in Power BI to create meaningful reports and dashboards.

2.Objective

The objective of this project is to design and implement an end-to-end ETL (Extract, Transform, Load) pipeline for an e-commerce dataset, enabling efficient data processing and visualization. The project aims to:

- ➢ Extract raw data from CSV files.
- ➢ Clean and transform the data to handle missing values, inconsistencies, and unnecessary columns.
- ➢ Load the processed data into a PostgreSQL database for structured storage and easy access.
- ➢ Visualize the key insights using Power BI, providing dashboards for business intelligence and decision-making.
- ➢ Optimize performance by implementing best practices for data handling and storage.
- ➢ Ensure security by managing database access and handling errors efficiently.

3.System requirement

Before setting up and running this ETL pipeline, ensure the following prerequisites are met:

1. Software Requirements

- Python 3.17-for data extraction, transformation, and loading.
- PostgreSQL -for storing the cleaned data.
- Power BI -for data visualization.
- VS Code with jupyter extension.

2. Python Libraries

Ensure you have the following libraries installed:

- pandas for data cleaning and transformation
- sqlalchemy for connecting to PostgreSQL
- psycopg2 for interacting with the PostgreSQL database

3. Database Setup

PostgreSQL must be installed and running.

A database named EcommerceDataset should be created in PostgreSQL.

4. CSV Dataset

The raw e-commerce dataset should be available as Pakistan Largest Ecommerce Dataset.csv.click here to see the sourse of data set Pakistan's Largest E-Commerce Dataset

The cleaned dataset will be stored as cleaned.csv before loading into PostgreSQL.

5. Hardware Requirements

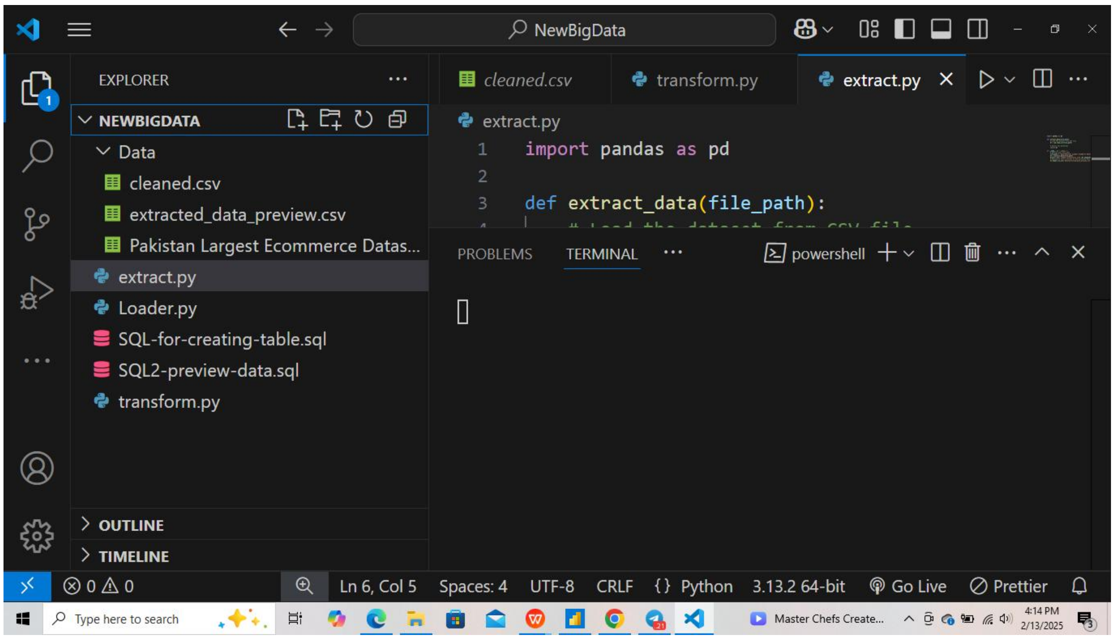
- Processor: Intel Core i5 or higher (or AMD equivalent)
- RAM: Minimum 8 GB (16 GB recommended for large datasets)

- ➢ Storage: At least 10 GB of free space for dataset storage and database files
- ➢ Operating System: Windows 10/11, macOS, or Linux

4.ETL Process Description

The ETL (Extract, Transform, Load) process is a crucial step in data analytics and business intelligence. It ensures that raw data is collected, cleaned, and stored efficiently for further analysis. This project follows a structured ETL process using Python and PostgreSQL, focusing on three main stages: Extraction, Transformation, and Loading.

This is the general structure of the Project folder



```
NEWBIGDATA
  Data
    cleaned.csv
    extracted_data_preview.csv
    Pakistan Largest Ecommerce Datas...
  extract.py
  Loader.py
  SQL-for-creating-table.sql
  SQL2-preview-data.sql
  transform.py
```

```
import pandas as pd

def extract_data(file_path):
```

4.1. Extraction (Extracting Data)

The extraction phase is responsible for retrieving raw data from the source. In this project, the data is extracted from a CSV file named "Pakistan Largest Ecommerce Dataset.csv" using the extract.py script.

Steps in Extraction:

- ➢ The script reads the CSV file into a Pandas DataFrame.
- ➢ It verifies that the file exists and loads it into memory.

- After successful extraction, the first few rows are saved as a preview to ‘extracted_data_preview.csv’ for verification.

This is the implementation extract.py

```python
import pandas as pd

def extract_data(file_path):
    # Load the dataset from CSV file
    df = pd.read_csv(file_path)

    # Return the DataFrame
    return df
```

```python
if __name__ == "__main__":
    # Specify your file path
    file_path = 'Data/Pakistan Largest Ecommerce Dataset.csv'
    df = extract_data(file_path)
    print(f"Data loaded successfully with {df.shape[0]} rows and
{df.shape[1]} columns.")
    # Optionally save a preview of the extracted data
    df.head().to_csv('Data/extracted_data_preview.csv', index=False)
```

4.2. Transformation (Cleaning and Transforming Data)

The Transformation phase is where the raw data is cleaned, processed, and prepared for loading into the database. This involves handling missing values, fixing inconsistencies, converting data types, and removing unnecessary columns.

Steps in Transformation:

Handle Missing Values:

- Replace \N and #REF! with 'Unknown' or appropriate placeholders.

- Fill missing numeric values with the median or mean.

Convert Data Types:

- ➢ Convert numeric columns (e.g., price, grand_total) to appropriate types (e.g., float or int).
- ➢ Convert date columns (e.g., created_at, Working Date) to datetime format.

Remove Unnecessary Columns:

- ➢ Drop columns that are not required for analysis(e.g., MV, Year, Month, Customer Since).

Remove Duplicates:

- ➢ Drop duplicate rows to ensure data integrity.

Strip Spaces from String Values:

- ➢ Remove leading/trailing spaces from string columns to ensure consistency.

Implementation of transform.py:

```
import pandas as pd

def clean_data():
    print("Loading the dataset...")
    raw_df = pd.read_csv('Data/Pakistan Largest Ecommerce Dataset.csv',
dtype=str, low_memory=False)
    # --- Fix Column Names ---
    raw_df.columns = raw_df.columns.str.strip()  # Remove spaces around
column names
    # --- Handle Missing Values ---
    print("Handling missing values...")
    raw_df['sales_commission_code'] =
raw_df['sales_commission_code'].replace('\\N',
'Unknown').fillna('Unknown')
    raw_df['BI Status'] = raw_df['BI Status'].replace('#REF!',
'Unknown').fillna('Unknown')
    # --- Convert Numeric Columns ---
    print("Converting numeric columns...")
    num_cols = ['price', 'grand_total', 'discount_amount',
'qty_ordered']
```

```
    for col in num_cols:
        if col in raw_df.columns:
            raw_df[col] = raw_df[col].astype(str).str.replace(',',
'').astype(float)
            raw_df[col] = raw_df[col].fillna(raw_df[col].median())  #
Use median for missing values
    # --- Convert Integer Columns ---
    int_cols = ['item_id', 'qty_ordered', 'Customer ID', 'increment_id']
    for col in int_cols:
        if col in raw_df.columns:
            raw_df[col] = pd.to_numeric(raw_df[col],
errors='coerce').fillna(0).astype(int)
    # --- Handle Date Columns ---
    print("Fixing date columns...")
    date_cols = ['created_at', 'Working Date', 'M-Y']
    for col in date_cols:
        if col in raw_df.columns:
            raw_df[col] = pd.to_datetime(raw_df[col], errors='coerce')
    # --- Remove Unnecessary Columns ---
    print("Removing unnecessary columns...")
    cols_to_drop =  [
        'MV', 'Year', 'Month', 'Customer Since',
        'Unnamed: 19', 'Unnamed: 20', 'Unnamed: 21',
        'Unnamed: 22', 'Unnamed: 23', 'Unnamed: 24', 'Unnamed:
25','sales_commission_code'
    ]
    cols_to_drop = [col for col in cols_to_drop if col in
raw_df.columns]
    raw_df = raw_df.drop(columns=cols_to_drop)
    # --- Remove Duplicates ---
    print("Removing duplicates...")
    raw_df = raw_df.drop_duplicates()
    # --- Strip Spaces from String Values ---
    raw_df = raw_df.applymap(lambda x: x.strip() if isinstance(x, str)
else x)
    # --- Save Cleaned Data ---
    print("Saving cleaned data to 'cleaned.csv'...")
    raw_df.to_csv('Data/cleaned.csv', index=False)
```

```python
    print("Data cleaning completed successfully!")
if __name__ == "__main__":
    clean_data()
```

4.3. Loading (Storing Data in PostgreSQL)

The Loading phase involves storing the cleaned and transformed data into a PostgreSQL database for structured storage and efficient access. This step ensures that the data is readily available for further analysis and visualization in Power BI.

Steps in Loading:

1. Establish a Connection to PostgreSQL

The script connects to a PostgreSQL database using SQLAlchemy.

The connection string includes the database name (ecommerce_db), username (postgres), and password (122313).

2. Create Table (If Not Exists)

The script first creates an empty table structure by writing only the column names from the cleaned dataset.

This ensures that the table is structured properly before inserting data.

3. Load Data Efficiently

The cleaned data is inserted into the database in chunks of rows to handle large datasets efficiently.

If the table already exists, the data is appended rather than replaced.

4. Print Success Message

After successfully loading the data, the script prints "Data successfully loaded to PostgreSQL!" to indicate completion.

Implementation of loader.py

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

```
def load_to_postgres(cleaned_df):
    """
    Load cleaned data into a PostgreSQL database.
    Parameters:
        cleaned_df (DataFrame): Cleaned e-commerce data.
    """
    # Database connection
    engine =
create_engine('postgresql://postgres:122313@localhost:5432/ecommerce_db
')
    # Create table if not exists (for fresh table)
    cleaned_df.head(0).to_sql('ecommerce_table', engine,
if_exists='replace', index=False)
    # Load in chunks for large datasets
    cleaned_df.to_sql('ecommerce_table', engine, if_exists='append',
index=False, chunksize=10000)
    print("Data successfully loaded to PostgreSQL!")
if __name__ == "__main__":
    # Load cleaned data from 'Data/cleaned.csv'
    cleaned_df = pd.read_csv('Data/cleaned.csv')
    # Load data to PostgreSQL
    load_to_postgres(cleaned_df)
```

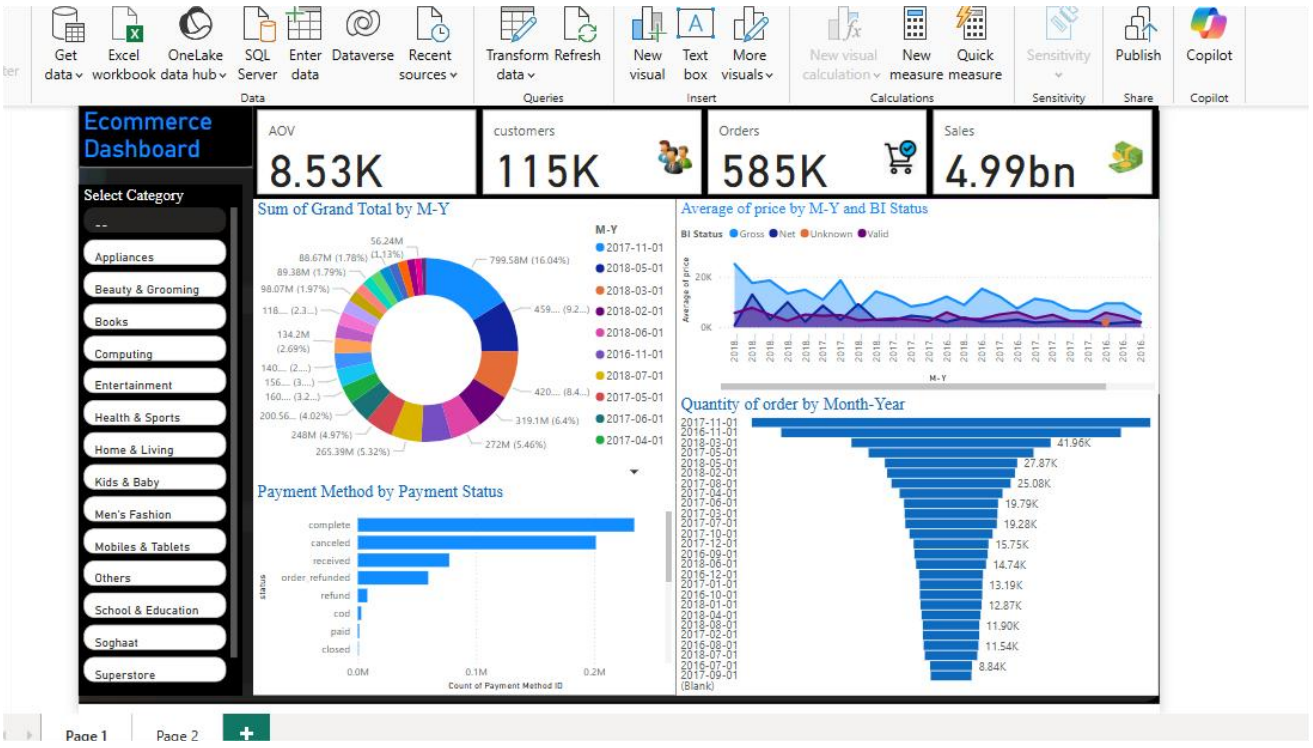
4.Creating table in Postgress Sql

```
CREATE TABLE ecommerce_table(
    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),
    discount_amount FLOAT,
    payment_method VARCHAR(50),
    working_date TIMESTAMP,
    bi_status VARCHAR(50),
    m_y TIMESTAMP
);
```

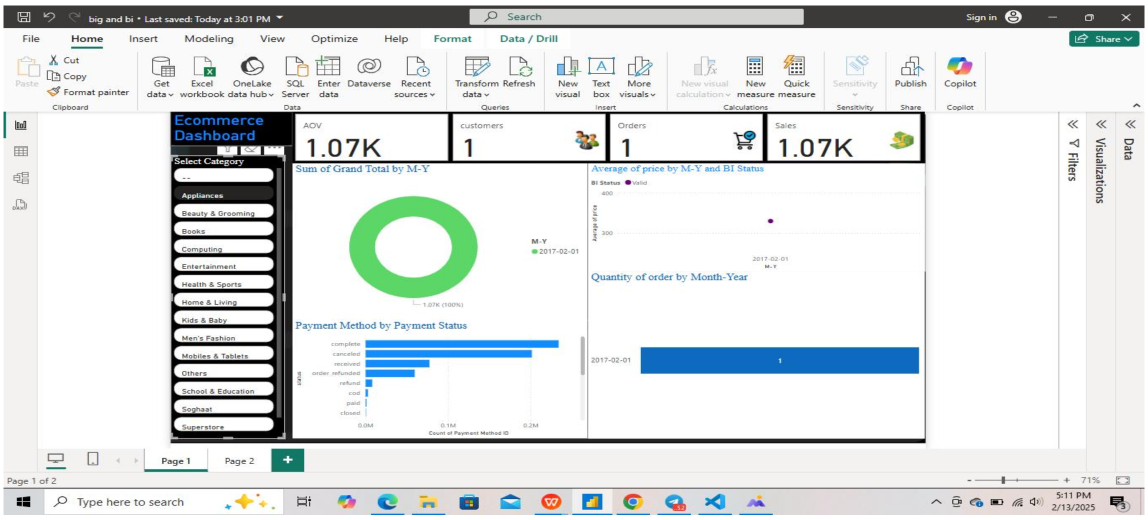
A code for Previewing First 10 Rows of loaded data

```
SELECT * FROM ecommerce_table LIMIT 10;
```

5.Visualization(Dashboard)



This is the page apears when you click the buttons



big and bi • Last saved: Today at 3:01 PM
Search
File
Home
Insert
Modeling
View
Optimize
Help
Format
Data / Drill
Ecommerce Dashboard
AOV 1.07K
customers 1
Orders 1
Sales 1.07K
Sum of Grand Total by M-Y
Average of price by M-Y and BI Status
Quantity of order by Month-Year
Payment Method by Payment Status
Page 1
Page 2
Page 1 of 2

6.Performance Optimization

Optimizing performance is crucial for handling large datasets efficiently. The following best practices are implemented or can be further improved in the ETL pipeline:

1. Batch Processing

Instead of loading the entire dataset at once, data is inserted into the PostgreSQL database in chunks.

This reduces memory consumption and prevents potential timeout or performance bottlenecks.

The chunksize parameter in to_sql() ensures data is processed in smaller, manageable portions (e.g., 10,000 rows per batch).

2. Indexing

Indexing key columns (such as Customer ID, item_id, or created_at) significantly improves query performance.

Indexing allows faster retrieval of data, reducing the time taken for searching, filtering, and aggregation.

3. Logging

Implementing logging ensures that the ETL pipeline is monitored and issues are detected early.

Logging can be implemented using Python's logging module to record events such as:

- Successful data extraction, transformation, and loading
- Errors encountered during execution
- Performance metrics (e.g., processing time for each stage)

6.1Security Considerations

Since this ETL pipeline handles potentially sensitive business data, it is essential to enforce security measures to protect the database and stored information.

1. Database Credentials

Hardcoding database credentials (username and password) in the script is a security risk.

Instead, credentials should be stored securely using environment variables or a secrets management tool like AWS Secrets Manager or HashiCorp Vault.

2. Data Privacy

If the dataset contains personally identifiable information (PII) such as customer names, email addresses, or phone numbers, it is crucial to anonymize or encrypt sensitive fields.

3. Access Control

Database access should be restricted to authorized users only.

It is recommended to create separate user roles for different operations.

6.2Future Enhancements

As the dataset grows, additional features can be integrated into the ETL pipeline for better automation, reliability, and efficiency.

1. Automated Scheduling

Instead of manually running the script, scheduling can be automated using cron jobs (Linux/macOS), Task Scheduler (Windows), or Apache Airflow.

2. Data Validation

Before loading data into PostgreSQL, validation checks can ensure data integrity.

Example validation rules:

- Ensure price and grand_total are positive values.
- Check that required fields (e.g., Customer ID, order_id) are not null.
- Verify that created_at dates fall within a valid range.

3. Incremental Data Loading

Instead of replacing or appending the entire dataset, the pipeline can be optimized to only load new or updated records.

7.Conclusion

This end-to-end ETL pipeline streamlines data extraction, transformation, and loading into a PostgreSQL database, enabling businesses to efficiently process large e-commerce datasets. The integration with Power BI ensures that key insights are visualized effectively for decision-making.

With performance optimizations such as batch processing and indexing, the system is designed to handle large datasets efficiently. Additionally, security measures including credential management and access control help protect sensitive business data.