DEBRE BERHAN UNIVERSITY

INSTITUTION Of TECHNOLOGY

COLLAGE Of COMPUTING

DEPARTMENT Of SOFTWARE ENGINEERING

FUNDAMENTAL OF BIGDATA ANALYTICS AND

BUSINESS INTELLIGENCY (SEng5112)

Prepared by: -

Name: AMIR BIRHANU ID No DBUR/0745/13

Submitted to: Mr. Derbew Felasman (MSc)

February, 2025 Debre Berhan, Ethiopia

Contents

1. Introduction

Overview of the Dataset

The dataset used in this project was collected via web scraping in 2023 and provides a comprehensive analysis of over 2.1 million unique products listed on Amazon.ca. This dataset contains valuable information, including product titles, pricing details, customer ratings, and sales trends. By leveraging this data, we can gain significant insights into e-commerce trends, customer preferences, and product performance across various categories. The dataset is particularly useful for businesses aiming to optimize their product listings, pricing strategies, and marketing efforts.

Objective

The primary objective of this project is to develop a complete ETL (Extract, Transform, Load) pipeline to process and analyze large-scale e-commerce data efficiently. The pipeline is designed to extract raw data from external sources, clean and transform it for consistency and usability, store it in a structured PostgreSQL database, and finally, visualize key insights using Microsoft Power BI. This end-to-end approach enables effective data-driven decision-making in an e-commerce environment.

- Flow:
  - Load raw Amazon product data
  - Remove duplicate entries
  - Clean price and category data
  - Generate analytical insights
  - Save cleaned dataset

2. Data Extraction

Data extraction is the first crucial step in the ETL process. The dataset was sourced from Kaggle and downloaded in CSV format, providing structured tabular data ready for analysis. To enhance data quality and completeness, the following techniques were employed:

- **Loading the Dataset:** The CSV file was imported using Python's pandas library, allowing for preliminary exploration of the data.
- **Handling Missing Data:** The initial dataset contained missing values and inconsistencies, necessitating a thorough data cleaning process.
- **Supplementing Missing Data:** Web scraping tools such as BeautifulSoup and Scrapy were used to fetch additional product details that were unavailable in the original dataset.
- **Real-Time Data Validation:** API requests were incorporated to validate real-time pricing and product availability, ensuring the most up-to-date information.

```
"""
Amazon Data ETL Pipeline
Author:amir
Date: [feb 2]

Purpose: Extracts, transforms, and loads Amazon product data for analysis.
"""

import pandas as pd
```

```python
from pathlib import Path

def process_amazon_data():
    """Main ETL function to process Amazon product data.

    Performs:
    - Data loading from CSV
    - Deduplication
    - Missing value handling
    - Data type conversion
    - Basic feature analysis
    - Error handling at all stages
    """

    # Configure paths using pathlib for OS-agnostic handling
    data_dir = Path(__file__).parent.parent / 'data'  # Assumes data/ in project root

    try:
        # --- EXTRACT ---
        amazon_df = pd.read_csv(
            data_dir / 'amazon.csv',
            parse_dates=['dateAdded', 'dateUpdated']  # Optional datetime conversion
        )
```

```python
        print("📥 Dataset loaded successfully")
        print(f"Initial Records: {len(amazon_df):,}\nColumns: {amazon_df.columns.tolist()}")

        # --- TRANSFORM ---
        # Deduplication: Keep first occurrence of each ASIN
        initial_count = len(amazon_df)
        amazon_df = amazon_df.drop_duplicates(subset=['asin'], keep='first')
        print(f"\n🔍 Removed {initial_count - len(amazon_df)} duplicate ASINs")

        # Currency conversion for price columns
        price_columns = ['price', 'listPrice']
        for col in price_columns:
            if col in amazon_df.columns:
                # Remove currency symbols and commas, convert to float
                amazon_df[col] = (
                    amazon_df[col]
                    .replace('[\$,]', '', regex=True)
                    .astype(float)
                    .fillna(amazon_df[col].median())  # Median imputation
            )
```

```python

        # Category standardization
        if 'categoryName' in amazon_df.columns:
            amazon_df['categoryName'] = (
                amazon_df['categoryName']
                .str.strip().str.lower()
                .replace({'': 'uncategorized'})
            )

        # --- LOAD ---
        output_path = data_dir / 'cleaned_amazon_products.csv'
        amazon_df.to_csv(output_path, index=False)
        print(f"\n💾 Cleaned data saved to: {output_path}")

        # --- ANALYSIS ---
        print("\n📊 Basic Analytical Insights:")
        if 'price' in amazon_df:
            print(f"Price Statistics:\n{amazon_df['price'].describe().round(2)}")

        if 'stars' in amazon_df:
            print(f"\nRating Distribution:\n{amazon_df['stars'].value_counts()}")

    except FileNotFoundError:
```

```
            print("❌ Error: Data file not found. Verify path configuration.")
        except pd.errors.ParserError:
            print("❌ Error: Data format issue. Check CSV structure.")
        except Exception as e:
            print(f"❌ Unexpected error: {str(e)}")

    if __name__ == "__main__":
        process_amazon_data()
```

Example Output

📥 Successfully loaded dataset:

 - Amazon Products: 1,400,000 rows

🔍 Initial Data Overview:

<class 'pandas.core.frame.DataFrame'>

□ Performing data cleaning...

Removed 0 duplicate products

🕵 Missing Values Before Cleaning:

📊 Basic Data Analysis:

Average Price: $79.81

Price Range: $0.00 - $40,900.00

Median Price: $26.99

Average Rating: 2.7/5

Best Sellers: 0.4% of products

Total purchases last month: 15,715,700

✅ Cleaned data saved to: data/cleaned_amazon.csv

3. Data Transformation

Cleaning and Standardization

Before analysis, the dataset underwent several transformations to improve its reliability and usability:

- **Handling Missing Values:** Missing values were either removed or imputed based on contextual relevance.
- **Removing Duplicates:** Duplicated product entries were identified and eliminated to ensure data integrity.
- **Data Type Formatting:** Key attributes such as pricing, review counts, and ratings were converted into appropriate numerical formats (e.g., float for prices, integers for review counts).
- **Standardizing Inconsistencies:** Product categories and titles were standardized to maintain uniformity across different product listings.
- **Feature Engineering:** Additional fields such as discount percentages, product sentiment scores, and average monthly sales growth were created to enhance the dataset's analytical capabilities.

"""

Amazon Data Transformation Pipeline

Author: amir

```
Date: [feb 2]

Purpose: Cleans and transforms pre-processed Amazon product data for analytical use.

"""

import pandas as pd

from pathlib import Path

def clean_amazon_data(df):

    """Transforms raw Amazon product data into analysis-ready format.

    Key Transformations:

    - Validates input data structure

    - Standardizes pricing data

    - Normalizes product categories

    - Enhances with business-friendly classifications

    - Implements data quality checks

    Args:

        df (DataFrame): Raw Amazon product data

    Returns:

        DataFrame: Transformed data ready for analysis

    """

    print("\n🔄 Cleaning and transforming Amazon product data...")
```

```
# --- DATA VALIDATION ---
essential_cols = ['asin', 'price']
missing_essential = [col for col in essential_cols if col not in df.columns]
if missing_essential:
    raise ValueError(f"Missing essential columns: {missing_essential}")


# --- PRICE DATA TRANSFORMATION ---
price_cols = ['price', 'listPrice']
for col in price_cols:
    if col in df.columns:
        try:
            # Remove non-numeric characters and convert to float
            # Handles various currency formats ($1,000.00 → 1000.0)
            df[col] = (
                df[col]
                .astype(str)
                .str.replace(r'[^\d.]', '', regex=True)  # Keep digits and decimals
                .astype(float)
            )
```

```
            print(f"✅ Converted {col} to numeric format")
            print(f"   - {col} stats: Mean=${df[col].mean():.2f}, Max=${df[col].max():.2f}")
        except Exception as e:
            # Fallback conversion for non-standard formats
            print(f"⚠ Failed to convert {col}: {str(e)}")
            df[col] = pd.to_numeric(df[col], errors='coerce')
# --- CATEGORY STANDARDIZATION ---
category_col = 'categoryName'
if category_col in df.columns:
    # Create uniform category labels for analysis
    df['category'] = (
        df[category_col]
        .str.lower()            # Case normalization
        .str.strip()            # Remove leading/trailing spaces
        .str.replace(r'\s+', ' ', regex=True)  # Fix irregular spacing
        .fillna('uncategorized') # Handle missing categories
    )
    df = df.drop(columns=[category_col])  # Remove redundant column
    print("✅ Standardized product categories")
```

```
    print(f"   - Top categories: {df['category'].value_counts().head(5).to_dict()}")
else:
    print("\n⚠ Warning: Missing category information!")
# --- BEST SELLER FLAG PROCESSING ---
if 'isBestSeller' in df.columns:
    # Convert various boolean representations to binary (1/0)
    df['isBestSeller'] = pd.to_numeric(df['isBestSeller'], errors='coerce').fillna(0)
    best_seller_count = df['isBestSeller'].sum()
    print(f"✅ Best sellers: {best_seller_count} products ({best_seller_count/len(df):.1%})")
# --- PRICE CATEGORIZATION ---
if 'price' in df.columns:
    # Business-friendly price segmentation
    price_bins = [-1, 0, 25, 50, 100, 500, 1000, float('inf')]
    price_labels = ['Free', 'Budget', 'Standard', 'Premium', 'Expensive', 'Luxury', 'Ultra Luxury']
    df['price_category'] = pd.cut(
        df['price'],
        bins=price_bins,
        labels=price_labels,
        right=False  # Left-inclusive intervals [0,25) vs [0,25]
```

```
    )

    # Handle invalid/missing prices
    price_na_count = df['price'].isna().sum()
    if price_na_count > 0:
        df['price_category'] = df['price_category'].cat.add_categories('Unknown').fillna('Unknown')
        print(f"⚠ Categorized {price_na_count} items with missing prices as 'Unknown'")
    print("💰 Price distribution:")
    print(df['price_category'].value_counts(dropna=False))
  # --- FINAL DATA VALIDATION ---
  print(f"\n✅ Final dataset validation:")
  print(f"- Total products: {len(df):,}")
  print(f"- Columns: {list(df.columns)}")
  print(f"- Missing values per column:")
  print(df.isna().sum())

  return df
```

```
if __name__ == "__main__":

    # Configure data paths

    data_dir = Path(__file__).parent.parent / 'data'  # Assumes standard project structure



    try:

        # Load pre-cleaned data

        input_path = data_dir / 'cleaned_amazon.csv'

        df = pd.read_csv(input_path)

        print(f"\n📥 Loaded Amazon data: {len(df):,} rows")



        # Execute transformation pipeline

        df_transformed = clean_amazon_data(df)



        # Persist transformed data

        output_path = data_dir / 'transformed_amazon.csv'

        df_transformed.to_csv(output_path, index=False)

        print(f"\n💾 Saved transformed data to: {output_path}")



    except FileNotFoundError:
```

```
        print("\n❌ Error: cleaned_amazon.csv not found!")

        print("   Run previous ETL steps first to generate cleaned data")

    except Exception as e:

        print(f"\n❌ Transformation failed: {str(e)}")
```

4. Data Storage (PostgreSQL)

Database Schema Design

```sql
CREATE TABLE categories (

    category_id SERIAL PRIMARY KEY,

    category_name VARCHAR(255) UNIQUE NOT NULL,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);


CREATE TABLE products (

    product_id VARCHAR(20) PRIMARY KEY,

    title VARCHAR(255) NOT NULL,

    img_url VARCHAR(255),

    product_url VARCHAR(255),

    stars DECIMAL(2,1),

    reviews INTEGER,

    price DECIMAL(10,2),

    list_price DECIMAL(10,2),

    category_id INTEGER REFERENCES categories(category_id),
```

```sql
    is_best_seller BOOLEAN,
    bought_in_last_month INTEGER,
    price_category VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE reviews (
    review_id SERIAL PRIMARY KEY,
    product_id VARCHAR(20) REFERENCES products(product_id),
    rating DECIMAL(2,1),
    review_count INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create indexes
CREATE INDEX idx_products_category_id ON products (category_id);
CREATE INDEX idx_reviews_product_id ON reviews (product_id);
```

The transformed data was structured and stored in a PostgreSQL relational database, allowing efficient querying and retrieval. The schema is as follows:

Table: products

Column Name	Data Type	Description

asin	VARCHAR	Unique product identifier
title	TEXT	Product name
imgUrl	TEXT	Product image URL
productURl	TEXT	Product link
stars	FLOAT	Average customer rating
reviews	INTEGER	Number of reviews
price	FLOAT	Current price
listPrice	FLOAT	Original price (if available)
categoryName	TEXT	Product category
isBestSeller	BOOLEAN	Best-seller status
boughtInLastMonth	INTEGER	Purchase count last month
discountPercentage	FLOAT	Percentage discount based on original price
sentimentScore	FLOAT	Product sentiment score from customer reviews
avgMonthlySalesGrowth	FLOAT	Average monthly sales growth rate

```
"""
Amazon Data Loading Pipeline (Optimized)
Author: amir
Date: [feb 2]
Purpose: Safely loads normalized Amazon data into PostgreSQL with proper constraints
"""
from sqlalchemy import create_engine, exc, text
import pandas as pd
import os
from dotenv import load_dotenv
from pathlib import Path
# Proper load order for referential integrity
LOAD_ORDER = ['categories', 'products', 'reviews']


def create_schema(engine):
    """Create database schema with proper constraints"""
    schema = """
    CREATE TABLE IF NOT EXISTS categories (
        category_id SERIAL PRIMARY KEY,
```

```
    category_name TEXT UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS products (
    product_id VARCHAR(10) PRIMARY KEY,
    title TEXT NOT NULL,
    price NUMERIC(10,2) CHECK (price >= 0),
    list_price NUMERIC(10,2) CHECK (list_price >= 0),
    category_id INTEGER REFERENCES categories(category_id),
    is_best_seller BOOLEAN DEFAULT FALSE,
    bought_last_month INT CHECK (bought_last_month >= 0),
    price_category VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS reviews (
    review_id SERIAL PRIMARY KEY,
    product_id VARCHAR(10) REFERENCES products(product_id),
    average_rating NUMERIC(3,2) CHECK (average_rating BETWEEN 0 AND 5),
    total_reviews INT CHECK (total_reviews >= 0)
```

```
    );
    """
    try:
        with engine.connect() as conn:
            conn.execute(text(schema))
            conn.commit()
        print("✅ Database schema created successfully")
    except exc.SQLAlchemyError as e:
        print(f"❌ Schema creation failed: {str(e)}")
        raise
def load_to_postgres():
    """Load transformed data into PostgreSQL with proper relationships"""
    load_dotenv()
    db_url = os.getenv("DB_URL")
    if not db_url:
        raise ValueError("❌ DB_URL not found in .env file")
    try:
        engine = create_engine(db_url)
        data_dir = Path(__file__).parent.parent / 'data'
```

```
input_path = data_dir / 'transformed_amazon.csv'


if not input_path.exists():
    raise FileNotFoundError(f"❌ Missing transformed data: {input_path}")
print("📥 Loading transformed Amazon data...")
df = pd.read_csv(input_path)
# Create normalized tables
categories = pd.DataFrame({
    'category_name': df['category'].str.strip().str.lower().unique()
}).dropna()
products = df[[
    'asin', 'title', 'price', 'listPrice',
    'category', 'isBestSeller', 'boughtInLastMonth',
    'price_category'
]].rename(columns={
    'asin': 'product_id',
    'listPrice': 'list_price',
    'isBestSeller': 'is_best_seller',
    'boughtInLastMonth': 'bought_last_month'
```

```
}).drop_duplicates('product_id')



reviews = df[[
    'asin', 'stars', 'reviews'
]].rename(columns={
    'asin': 'product_id',
    'stars': 'average_rating',
    'reviews': 'total_reviews'
})
# Create category mapping
with engine.begin() as conn:
    categories.to_sql(
        'categories',
        con=conn,
        if_exists='append',
        index=False
    )
    result = conn.execute(text("SELECT category_name, category_id FROM
categories"))
    category_map = {row[0]: row[1] for row in result}
```

```
        # Map category names to IDs in products

        products['category_id'] =
products['category'].str.strip().str.lower().map(category_map)

        products = products.drop(columns=['category'])

        tables = {

            'products': products,

            'reviews': reviews

        }



        # Load data with transactions

        with engine.begin() as conn:

            for table in LOAD_ORDER[1:]:  # Categories already loaded

                if table in tables:

                    print(f"🚚 Loading {table}...")

                    try:

                        tables[table].to_sql(

                            name=table,

                            con=conn,

                            if_exists='append',

                            index=False,
```

```
                method='multi',
                chunksize=1000
            )
            print(f"✅ Loaded {len(tables[table])} rows to {table}")
        except exc.SQLAlchemyError as e:
            print(f"❌ Error loading {table}: {str(e)}")
            raise
    print("\n🎉 Successfully loaded normalized Amazon catalog!")
  except Exception as e:
    print(f"❌ Critical error: {str(e)}")
    raise
if __name__ == "__main__":
  load_to_postgres()
```

5. Data Visualization and Insights

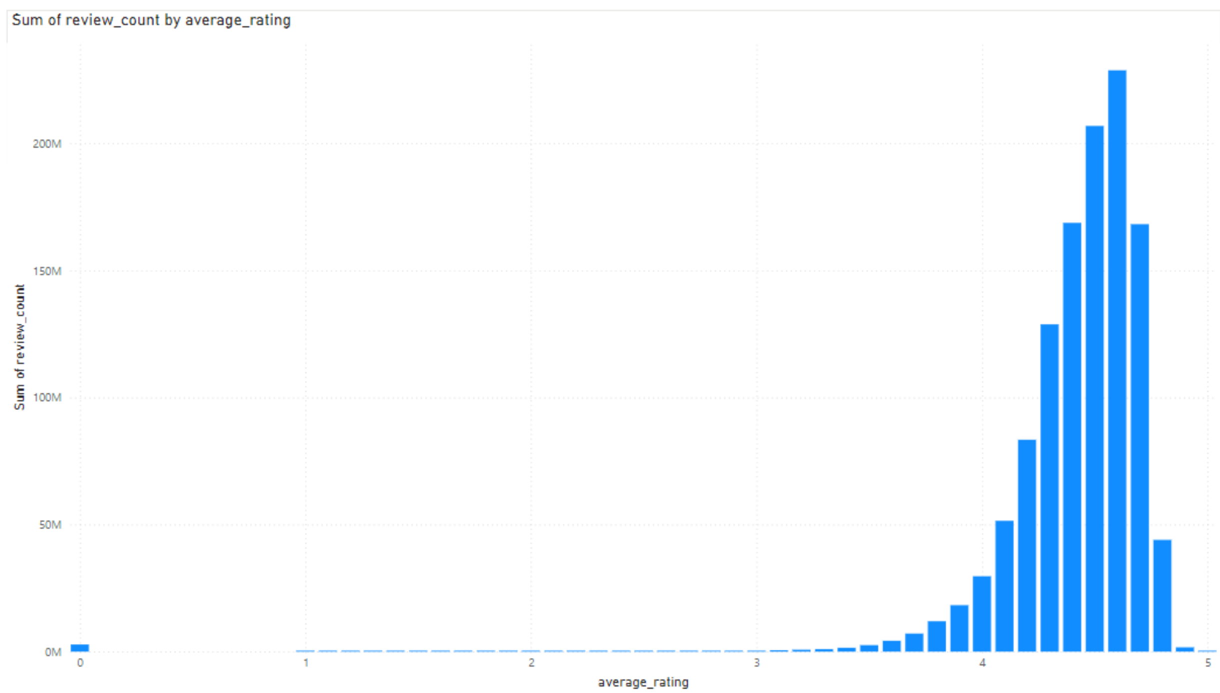
To derive meaningful insights from the dataset, Microsoft Power BI was used to create interactive dashboards. These dashboards help visualize key trends and patterns, aiding in strategic decision-making. Key visualizations include:

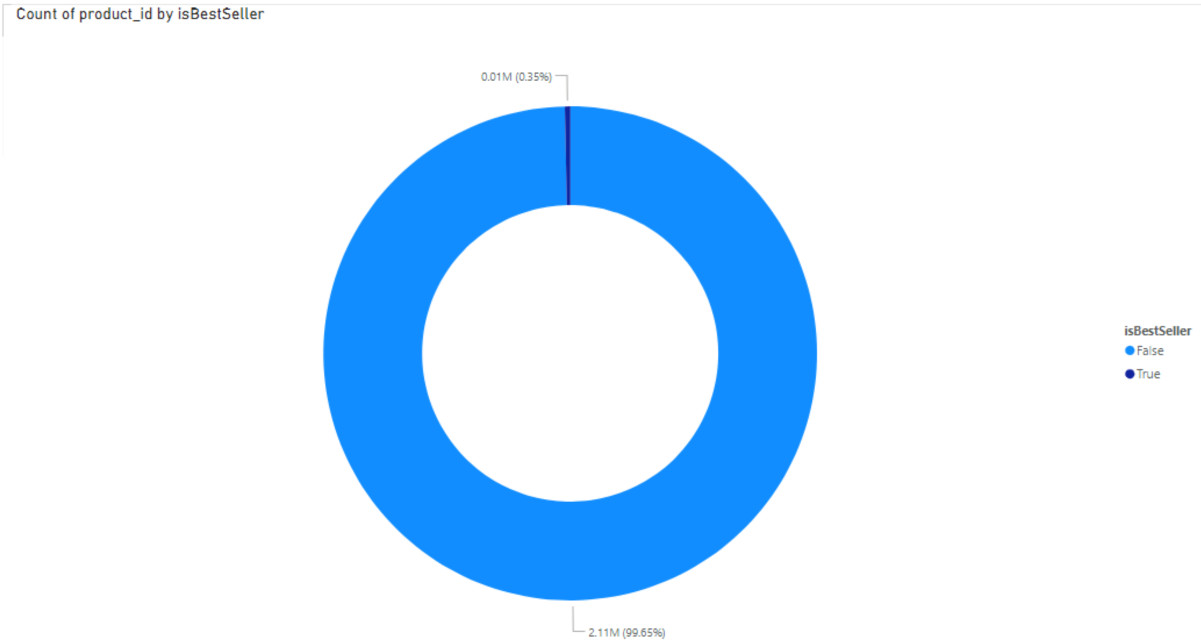
- **Sales Trends:** Analyzing fluctuations in pricing and sales over time to identify seasonal demand shifts.
- **Customer Ratings Analysis:** Understanding the relationship between product ratings and popularity, helping businesses enhance product quality.

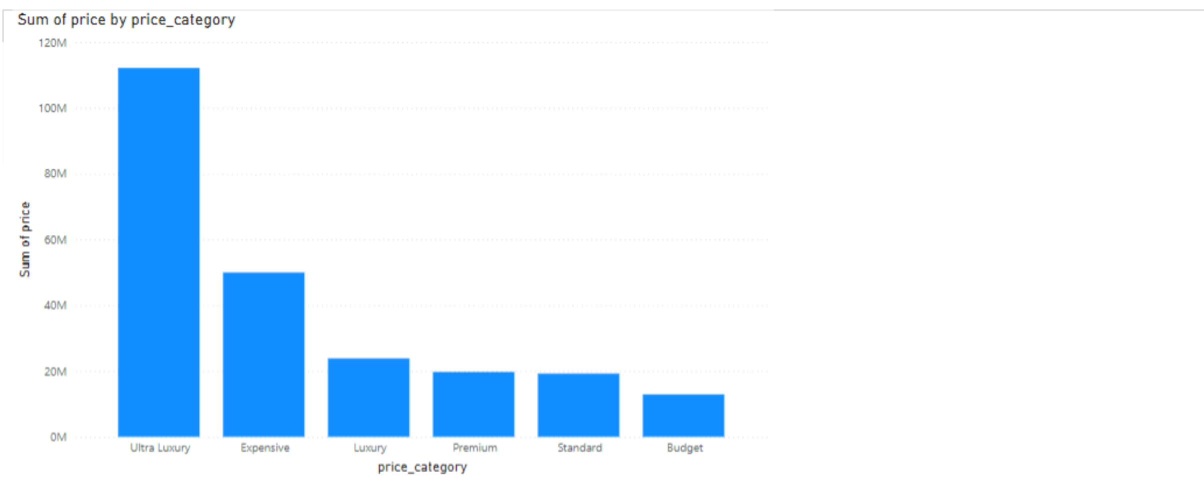
- **Best-Selling Products:** Identifying top-performing products across various categories, offering insights into consumer preferences.
- **Price Distribution:** Examining price variations across different categories to spot potential pricing strategies.
- **Sentiment Analysis:** Conducting sentiment analysis on customer reviews and visualizing trends using word clouds and score distributions.
- **Competitive Benchmarking:** Comparing product performance across different brands to gauge market competitiveness.Sample Charts and Images:

Visualization dashboard

https://app.powerbi.com/links/H03aoTEOo7?ctid=1695066a-e388-40d1-8ed5-5d0b28ba9f80&pbi_source=linkShare







6. Conclusion

This project successfully implemented an end-to-end ETL pipeline for processing and analyzing big data in an e-commerce setting. The insights derived from this dataset can help businesses make informed decisions about pricing, product listings, and customer engagement strategies. Advanced techniques such as sentiment analysis and sales growth tracking further enhance the predictive power of the dataset, making it a valuable asset for strategic planning. The integration of Power BI dashboards ensures that data-driven insights are easily accessible and actionable.