Technical Report: Final Project DS 5110: Introduction to Data Management and Processing

Rayan Hassan, Soumyae Tyagi
Khoury College of Computer Sciences
Data Science Program
hassan.ray@northeastern.edu, tyagi.so@northeastern.edu

December 3, 2024

Contents

1 Introduction

The project consists of designing and implementing a scalable data warehousing solution for a large e-commerce company. It aims to manage and analyze vast amounts of product reviews and metadata, enabling more effective data analysis, business intelligence, and reporting capabilities. The solution handles both structured and unstructured data, ensuring data integrity, scalability, and performance for future growth. The project includes a database implementation that allows the company to access product metadata and modify it by adding or removing data to or from it. On the other hand, it encompasses a ChatBot that can be used by clients or users to help them in their search for specific products and answer any questions they might have. That way, we would have satisfied both clients' and companies' needs.

2 Literature Review

The design and implementation of scalable data warehousing solutions for e-commerce platforms is a key area of research. Traditional databases like relational systems have been optimized for structured data, while NoSQL databases have been used with unstructured data, such as product reviews and user-generated content. Recent advancements focus on hybrid architectures that combine these two types of databases for comprehensive data management (Stonebraker et al., 2018 C). Additionally, the integration of ETL (Extract, Transform, Load) pipelines has been central to automating the data processing flow. According to Faisal et al. (2020) C, automated ETL processes can significantly reduce manual errors, optimize query performance, and improve operational efficiency. Our project leverages these concepts, utilizing Apache Airflow for task orchestration to keep the data pipeline consistent and efficient.

However, gaps remain in achieving real-time data processing, as many current systems struggle to scale effectively with growing data volumes. Another challenge in the industry is the lack of truly interactive business intelligence (BI) dashboards. While traditional BI tools are useful, they often lack real-time interactivity and require manual updates, limiting their potential for timely decision-making. Finally, the integration of AI-powered tools, such as chatbots, still poses challenges in deploying smoothly and handling diverse customer queries (Chen et al., 2012 C). This project contributes by focusing on a system that efficiently processes large volumes of data, and offers interactive dashboards that provide valuable visualizations and allow for better user engagement.

3 Methodology

This section describes the steps taken towards achieving our final goal.

3.1 Dataset Information

The first step was choosing a dataset to work with. The latter consists of reviews and metadata for video games and toys, sourced from a comprehensive collection of Amazon product reviews data that are recorded from May 1996 to September 2023. This data helps improve our eCommerce chatbot by giving it better insights into customer opinions and preferences, allowing it to answer questions with more relevant, context-specific information.

The dataset (C) originates from several trusted sources, including:

- Hugging Face: Amazon Reviews 2023 Dataset
- Academic Paper: Bridging Language and Items for Retrieval and Recommendation
- GitHub Repository: Amazon Reviews 2023 Project

User Reviews Dataset

This dataset contains detailed information on user reviews for various products. The key features are listed in table 1.

Item Metadata Dataset

This dataset provides product-related metadata, with key features listed in table 2.

3.2 ERD Design

We have designed the ERD (Entity-Relationship Diagram) to reflect the structure and relationships within the database, which is shown in Figure 1. The key entities in the design include:

- **User Reviews:** Contains information on user-generated content such as ratings, text, and helpful votes. Foreign key is parent_asin which references Product Metadata.
- **Product Metadata:** Stores product-related data, including titles, pricing, and product features. Primary key is parent_asin.
- **Product Images:** Contains URLs for images associated with each product. Primary key is pimage_id and foreign key is parent_asin which references Product Metadata.
- **Categories:** Stores information about product categories. Foreign key is parent_asin which references Product Metadata.

3.3 Data Pipeline

3.3.1 Data Pipeline Components Summary

- download_data.py (Listing 1): This code downloads the data from a specified URL, extract the contents and uploads the extracted file to a cloud storage bucket.
- bucket_connection.py (Listing 2): This module is responsible for establishing a connection to the Google Cloud Storage bucket using environment variables.
- json_to_csv.py (Listing 3): This module downloads the JSON file from the cloud storage bucket, converts the JSON data to CSV format using pandas and uploads the CSV back to cloud storage.

- db_connection.py (Listing 4): This module is essential for establishing a secure connection to a Google Cloud SQL database using the necessary configurations.
- CSV_to_DB.py (Listing 5): This code is responsible for creating the user review and metadata tables and uses Google Cloud CLI to import metadata CSV from GCS into the PostgreSQL tables.
- db_to_schema.py (Listing 6): This module is responsible for cleaning the metadata table by removing invalid image URLs and splits metadata and user review tables into four different table.

Please note that the data pipeline folder is uploaded on our GitHub repository with all of these files, as well as more code and other details about the project. Find the link below:

Our GitHub Page

3.3.2 ETL Process

The data pipeline automates the ETL (Extract, Transform, Load) process, which is central to transforming the raw data into a format suitable for analysis, querying, and later machine learning. Here's a more detailed breakdown of how the ETL process works within our pipeline.

Extract: The first step in the ETL pipeline is data extraction. The download_data.py script is responsible for fetching raw product review and metadata data from a specified URL, which comes in JSON format. The Extract phase is triggered by scheduled jobs in Apache Airflow, keeping the pipeline up-to-date with the latest data. Once downloaded, the raw JSON files are placed in a Google Cloud Storage where they can be easily accessed by other components in the pipeline for further processing. After which, they are converted into CSV files using the json_to_csv.py script.

Transform: After extraction, the Transform phase focuses on cleaning and and normalizing the raw data. This is done by several Python scripts (such as db_to_schema.py), which ensure that the data is ready for loading into the database.

Load: Once the data has been cleaned and transformed, the Load phase involves inserting the processed data into a PostgreSQL database. This is done through the CSV_to_DB.py file, which uses SQL queries to insert the CSV data into the appropriate tables.

For the User Reviews Dataset, the CSV_to_DB.py script ensures that each review entry is stored in a table that includes metadata about the product (e.g., product ID, title, rating, helpful votes, etc.).

For the Item Metadata Dataset, a separate table is created for each product, storing all relevant metadata (e.g., price, categories, features, descriptions).

The data is inserted into the PostgreSQL database using SQL commands. This also includes indexing the tables to optimize query performance.

3.3.3 Automation and Scheduling for Adding Data

The entire ETL pipeline is orchestrated using Apache Airflow, which automates the scheduling and execution of all tasks involved in the process. Airflow ensures that each step in the ETL process is performed in the correct order, with error handling and retries built-in to manage failures properly. The workflow includes:

- **Airflow DAG (Directed Acyclic Graph)**: The ETL pipeline is defined as a DAG in Airflow. The DAG includes tasks such as downloading raw data, converting it into CSV format, cleaning the data, and loading it into the database. Each task depends on the successful completion of the previous one.

- **Task Scheduling**: Airflow allows tasks to be scheduled to run at specified intervals (e.g., daily, weekly), ensuring that new data is automatically ingested and processed regularly.

The Airflow.py file responsible for this automation is included in the GiHub repository. It uses Python Operators to execute Python functions (ingest_data_meta(), ingest_data_review(), etc.) for each step in the pipeline.

Figures 2 and 3 show how Airflow modularize our data pipeline to upload/add data.

3.3.4 Dockerization

To ensure a consistent execution environment across different stages of development and deployment, the entire ETL pipeline is containerized using Docker. Each module of the pipeline (data extraction, transformation, and loading) is packaged into its own Docker container, ensuring that all dependencies (e.g., Python libraries, database connectors) are bundled together in a self-contained environment.

Each part of the ETL process runs within a separate Docker container. This allows the pipeline to be easily deployed on different machines or cloud environments without worrying about inconsistencies in dependencies or system configurations.

3.3.5 SQL Queries for Data Removal

After the data is loaded into the database, SQL queries are used to aggregate and integrate the data from multiple tables. For instance, JOIN queries are used to link all tables based on the asin field, allowing for comprehensive queries that return both customer feedback and product information. We have provided an example of Data Removal. Figure 4 shows an SQL query that retrieves information for a specific product. One can see the corresponding data shown in the terminal. Figure 5 shows the SQL query for data removal from all tables. Finally, figure 6 depicts the result of data removal, which in fact deletes all content corresponding to the targeted product, as expected.

3.4 ChatBot

The VaultBot chatbot is an intelligent eCommerce assistant designed to help users search for products, review feedback, and make informed decisions. It leverages a combination of advanced AI and streamlined data workflows to provide accurate and efficient responses to user queries. Below is an explanation of the technologies used.

3.4.1 Data Management

VaultBot integrates with the PostgreSQL database to fetch product information, ratings, and reviews. This database acts as the foundation for providing users with relevant and reliable data about the items they are interested in. Secure connections are established using cloud-based configurations to maintain data integrity and accessibility.

3.4.2 Product Search and Recommendations

When a user searches for a product, VaultBot utilizes structured queries to identify the best-rated items matching the user's search terms. The chatbot displays the top options ranked by average customer ratings, allowing users to quickly discover suitable choices without sifting through irrelevant information.

3.4.3 Review Analysis

VaultBot processes customer reviews to deliver meaningful insights. These reviews are loaded into a structured format, enabling the chatbot to extract key points, trends, and user sentiments. This analysis helps users make more informed decisions based on collective feedback from other customers.

3.4.4 AI-Driven Responses

VaultBot employs advanced AI models such as ChatGroq and embedding techniques. These technologies enable the chatbot to understand user queries, summarize large volumes of data, and provide precise answers. By using language models, VaultBot can effectively interpret both product metadata and customer reviews to address user questions.

3.4.5 Interactive User Interface

Built with Streamlit, VaultBot features a dynamic web interface that offers a seamless user experience. Key functionalities include:

- Searching for products by name or description.
- Clicking on items to explore detailed product metadata and customer reviews.
- Asking specific questions about selected products for tailored insights.

3.4.6 Technical Workflow

- **Database Connectivity:** Secure connections are established using tools like SQLAlchemy and Google Cloud Connector to access PostgreSQL data.
- **Data Loading and Processing:** Customer reviews and metadata are converted into usable formats through tools like DataFrameLoader.
- **AI-Powered Retrieval:** The system uses FAISS (a vector database) and HuggingFace embeddings to match user queries with relevant product information or reviews.

- **Response Generation:** A retrieval-augmented generation (RAG) pipeline combines AI-driven query understanding with data retrieval for producing contextually accurate responses.

3.5 Dashboards

The final step consisted of building interactive dashboards using Flask for the e-commerce Data Vault, and Streamlit for the VaultBot, both which will be discussed in Section 4

3.5.1 E-commerce Data Vault

This dashboard was made to help the company access and visualize data, which includes both user reviews and product metadata. The first page loads and displays a control panel (Figure 7) that allows the company to choose one of four options: Access User Reviews (Figure 8), Access Product Metadata (Figure 9), User Reviews Visualization (Figure 10), and Product Metadata Visualization (Figure11). Accessing either user reviews or product metadata allows the company to look up any review or product by their respective Asin values. One only has to enter or choose from a list of Asins, and the corresponding information will automatically be filtered and displayed on the screen. As for the visualization tabs, these allow for better analysis of the data, and will be discussed in detail in the upcoming section (4).

3.5.2 ChatBot UI

The user interface (UI) of the VaultBot chatbot is designed with simplicity and interactivity at its core, ensuring a seamless experience for users. Built using Streamlit, the UI offers a modern, web-based platform for easy navigation and engagement. As shown in Figure 12, the UI initial screen displays a search bar where users can search for any product. After the latter is chosen, top 5 products recommendations get displayed, which is shown in Figure 13. Once an option is selected, a second search bar renders on the screen to prompt the user to enter any question they would like to ask (Figure 14). The bot will then provide the answer to that question based on all the information held in the dataset.

4 Results

4.1 Data Pipeline Testing

In order to show that all tasks are performing properly we have tested the Airflow with 1% of data to show that all the dags are working as expected. This can be visualized in the following Figures: 2 and 3. The time-frame to run the dags get_review_data, get_metadata, json_to_csv_review_data, and json_to_csv_meta_data operators is highly dependent on the original data size.

4.2 Data Visualization

As part of the ongoing analysis of the product metadata and user reviews, we have generated meaningful visualizations that reveal key insights. These visualizations use a sample of our dataset, and serve as an essential tool in understanding both user behavior and product performance. The visualizations were created using data from both the user reviews and product metadata datasets. Please refer to the visualization code: 7, which includes all the functions discussed below.

Review Trends

When choosing "User Reviews Visualizations" on E-Commerce Data Vault Dashboard, one can see Review Trends associated with the User Reviews dataset (Figure 10). Two line plots were developed to analyze trends in user reviews over time. The first plot tracks the number of reviews per month, providing an overview of how the volume of reviews has evolved. The second plot displays the average rating per month, giving insights into how the ratings of products have changed over time. This data can be useful for identifying seasonal trends or shifts in product sentiment.

Rating Distribution by Category

The first figure in the Product Metadata Visualization (Figure 11) tab depicts a box-plot that shows how ratings are distributed within different product categories. This allows for an in-depth look at how ratings vary across categories, highlighting the spread and potential outliers in each category's review ratings.

Most Common Categories

This figure consists of a pie chart that shows the percentage of each category in product metadata and compares them to one another. This allows the company to note the category distribution within the dataset and act accordingly by maybe adding or removing products that belong to a certain category to ensure balance and equal distribution. In our plot, one can notice that the most common category in the sample dataset used is "Toys & Games".

Top 20 Products by Average Rating

The next figure is a bar chart showing the top 20 products based on average rating. This type of graph gives great insights on user satisfaction by showing the most highly rated products. In other words, products that have the highest satisfaction rate and therefore more likely to be purchased again.

Product Price Distribution

The last graph presents a box-plot that shows the price distribution among all products. Looking at the plot, it is clear that most prices range from 10 to 60 US dollars approximately. The dataset has a few outliers which include the maximum price at around 420 US dollars. This type of graph gives great insights on price range and allows for a general understanding of price spread.

4.3 ChatBot Testing

Multiple product selections and questions have been tested on the ChatBot. Once a product is selected, the interface dynamically updates to show relevant metadata and enable question submission. This is clearly shown in Figure 13. If no relevant products are found, the chatbot displays a warning message prompting the user to refine their search. Once a product is selected, users can ask their question. Figure 14 shows how that works. If no information is available to answer that question, the model will display a message to highlight that, as shown in Figure 15.

5 Discussion

One of the key challenges we encountered was maintaining real-time data processing. Despite PostgreSQL's ability to handle large-scale data efficiently, the integration of real-time ETL pipelines proved to be complex. However, using Apache Airflow for task orchestration helped streamline this process, which was consistent with previous studies (Faisal et al., 2020 C). Another critical aspect addressed in this project was the creation of interactive dashboards for business intelligence (BI). With the help of Flask, we were able to generate important visualizations that help the company get a better understanding of the data and its distribution. However, despite these improvements, there is still room for further enhancement in terms of user interaction and dynamic visualization. In fact, we had a limited amount of credits to be spent in Google Cloud Platform (GCP) which prevented us from working with the entire dataset for visualization. Last but not least, we were able to integrate a chatbot to assist users in navigating product data, and improve the general user experience.

6 Conclusion

This project successfully implemented a scalable data warehousing solution that integrates both structured and unstructured data, with a focus on performance and scalability. The use of automated ETL processes and hybrid database architecture proved effective in handling large volumes of data. While the interactive dashboards and ChatBot added value, there is still room for improvement in terms of real-time interactivity and AI capabilities. Additionally, limited GCP credits made it hard for us to work with the entire dataset for visualizations. Future work could focus on enhancing these aspects, addressing the gaps identified in scalability, user interface customization, and AI integration.

7 References

References

A Appendix A: Code

Here are the chunks of code mentioned in the document

```
    import os
import requests
import zipfile
import shutil
from src.bucket_connection import *
def ingest_data_meta(file_url):
    response = requests.get(file_url, timeout=30)
    filename = os.path.basename(file_url)
    zipfile_path = os.path.join(’Data’, ’temp1’, filename)
    extract_to = os.path.join(’Data’, ’temp1’)
    if response.status_code == 200:
        with open(zipfile_path, "wb") as file:
            file.write(response.content)
    else:
        print(f"Failed to download the file. Status code: {response.status_code}")
    extracted_files = []
    try:
        with zipfile.ZipFile(zipfile_path, ’r’) as zip_ref:
            zip_ref.extractall(extract_to)
            extracted_files = zip_ref.namelist()
    except zipfile.BadZipFile:
        print(f"Failed to unzip {zipfile_path}. It may not be a zip file.")
    nfile_name = str(extracted_files[0])
    temp_csv_file_path = os.path.join(’Data’, ’temp1’, nfile_name)
    destination_blob_directory = "Data/Raw/TEST/"
    destination_blob_name = os.path.join(destination_blob_directory, nfile_name)
    upload_blob(temp_csv_file_path, destination_blob_name)
    folder_path=’Data/temp1’
    shutil.rmtree(folder_path)
    os.makedirs(folder_path)
    message = "Done"
    return message
def ingest_data_review(file_url):
    response = requests.get(file_url, timeout=30)
    filename = os.path.basename(file_url)
    zipfile_path = os.path.join(’Data’, ’temp2’, filename)
    extract_to = os.path.join(’Data’, ’temp2’)
    if response.status_code == 200:
        with open(zipfile_path, "wb") as file:
            file.write(response.content)
    else:
        print(f"Failed to download the file. Status code: {response.status_code}")
    extracted_files = []
    try:
        with zipfile.ZipFile(zipfile_path, ’r’) as zip_ref:
            zip_ref.extractall(extract_to)
            extracted_files = zip_ref.namelist()
    except zipfile.BadZipFile:
        print(f"Failed to unzip {zipfile_path}. It may not be a zip file.")
    nfile_name = str(extracted_files[0])
    temp_csv_file_path = os.path.join(’Data’, ’temp2’, nfile_name)
    destination_blob_directory = "Data/Raw/TEST/"
    destination_blob_name = os.path.join(destination_blob_directory, nfile_name)
    upload_blob(temp_csv_file_path, destination_blob_name)
    folder_path=’Data/temp2’
    shutil.rmtree(folder_path)
    os.makedirs(folder_path)
    message = "Done"
    return message
```

Listing 1: download_data.py

```
    import os
from google.cloud import storage
from dotenv import load_dotenv

load_dotenv()

def connect_to_bucket():
    bucket_name = os.getenv("GCS_BUCKET_NAME")
    client = storage.Client()
    bucket = client.bucket(bucket_name)
    return bucket

def upload_blob(source_file_name: str, destination_blob_name: str):
    bucket = connect_to_bucket()
    blob = bucket.blob(destination_blob_name)
    blob.upload_from_filename(source_file_name)

def download_blob(source_blob_name: str, destination_file_name: str):
    bucket = connect_to_bucket()
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)

def list_blobs():
    bucket = connect_to_bucket()
    blobs = bucket.list_blobs()
    print("Blobs in the bucket:")
    for blob in blobs:
        print(blob.name)
```

Listing 2: bucket_connection.py

```
    import os
import pandas as pd
from dotenv import load_dotenv
# from bucket_connection import upload_blob

from src.bucket_connection import *

load_dotenv()

def json_to_csv_meta(source_blob_name, destination_blob_directory):
    file_name = os.path.splitext(os.path.basename(source_blob_name))[0]
    temp_json_file_path = f'Data/temp/{file_name}.json'
    temp_csv_file_path = f'Data/temp/{file_name}.csv'
    download_blob(source_blob_name, temp_json_file_path)
    data = pd.read_json(temp_json_file_path)
    data.to_csv(temp_csv_file_path, index=False)
    destination_blob_name = os.path.join(destination_blob_directory, f'{file_name}.csv')
    upload_blob(temp_csv_file_path, destination_blob_name)
    os.remove(temp_json_file_path)
    os.remove(temp_csv_file_path)
    message = f'Successfully converted {source_blob_name} to CSV and uploaded to {destination_blob_name}'
    return message
```

```python

def json_to_csv_review(source_blob_name, destination_blob_directory):
    file_name = os.path.splitext(os.path.basename(source_blob_name))[0]
    temp_json_file_path = f'Data/temp/{file_name}.json'
    temp_csv_file_path = f'Data/temp/{file_name}.csv'
    download_blob(source_blob_name, temp_json_file_path)
    data = pd.read_json(temp_json_file_path)
    data.to_csv(temp_csv_file_path, index=False)
    destination_blob_name = os.path.join(destination_blob_directory, f'{file_name}.csv')
    upload_blob(temp_csv_file_path, destination_blob_name)
    os.remove(temp_json_file_path)
    os.remove(temp_csv_file_path)
    message = f'Successfully converted {source_blob_name} to CSV and uploaded to {destination_blob_name}'
    return message
```

Listing 3: json_to_csv.py

```python
import os
import pg8000
import sqlalchemy
from dotenv import load_dotenv
from google.cloud.sql.connector import Connector, IPTypes

load_dotenv()

def connect_with_db() -> sqlalchemy.engine.base.Engine:
    instance_connection_name = os.getenv("INSTANCE_CONNECTION_NAME")
    db_user = os.getenv("DB_USER")
    db_pass = os.getenv("DB_PASS")
    db_name = os.getenv("DB_NAME")
    ip_type = IPTypes.PRIVATE if os.getenv("PRIVATE_IP") else IPTypes.PUBLIC
    connector = Connector()
    def getconn() -> pg8000.dbapi.Connection:
        conn: pg8000.dbapi.Connection = connector.connect(
            instance_connection_name,
            "pg8000",
            user=db_user,
            password=db_pass,
            db=db_name,
            ip_type=ip_type,
        )
        return conn
    pool = sqlalchemy.create_engine(
        "postgresql+pg8000://",
        creator=getconn,
    )
    return pool
```

Listing 4: db_connection.py

```python
import os
from airflow import DAG
from datetime import datetime
from google.cloud import storage
from dotenv import load_dotenv
from sqlalchemy import text
```

```
import subprocess

from src.db_connection import *
load_dotenv()

def create_table_user_review():
    postgres_conn_string = os.getenv("postgres_conn_string")
    engine = connect_with_db()
    with engine.begin() as connection:
        try:
            query = text("""CREATE TABLE IF NOT EXISTS user_reviews ( rating TEXT, title TEXT, text TEXT, images TEXT, asin TEXT, parent_asin TEXT, user_id TEXT, timestamp TEXT, helpful_vote TEXT, verified_purchase TEXT ); """)
            result = connection.execute(query)
        except Exception as e:
            message = f"Error during insert: {e}"

def create_table_meta_data():
    postgres_conn_string = os.getenv("postgres_conn_string")
    engine = connect_with_db()
    with engine.begin() as connection:
        try:
            query = text("""CREATE TABLE IF NOT EXISTS metadata ( main_category TEXT, title TEXT, average_rating TEXT, rating_number TEXT, features TEXT, description TEXT, price TEXT, images TEXT, videos TEXT, store TEXT, categories TEXT, details TEXT, parent_asin TEXT, bought_together TEXT, subtitle TEXT, author TEXT );  """)
            result = connection.execute(query)
        except Exception as e:
            message = f"Error during insert: {e}"

def add_meta_data():
    bucket_name = "mlops_data_pipeline/Data/Raw_CSV/TEST"
    file_name = ’test_metadata.csv’
    postgres_conn_string = os.getenv("postgres_conn_string")
    table_name = ’metadata’
    transfer_command = f"""
    yes | gcloud sql import csv data-wharehousing \
    gs://{bucket_name}/{file_name} \
    --project=dockdecoder \
    --database=postgres \
    --table={table_name}
    """
    try:
        # Run the command
        result = subprocess.run(transfer_command, shell=True, check=True, capture_output=True, text=True)
        print("Import successful:", result.stdout)
    except subprocess.CalledProcessError as e:
        print("Error during import:", e.stderr)

def add_review_data():
    bucket_name = "mlops_data_pipeline/Data/Raw_CSV/TEST"
    file_name = ’test_user_reviews.csv’
    postgres_conn_string = os.getenv("postgres_conn_string")
    table_name = ’user_reviews’
    transfer_command = f"""
    yes | gcloud sql import csv data-wharehousing \
    gs://{bucket_name}/{file_name} \
    --project=dockdecoder \
    --database=postgres \
    --table={table_name}
    """
    try:
        # Run the command
        result = subprocess.run(transfer_command, shell=True, check=True, capture_output=True, text=True)
        print("Import successful:", result.stdout)
    except subprocess.CalledProcessError as e:
        print("Error during import:", e.stderr)
```

Listing 5: CSV_to_DB.py

```
    import os
from airflow import DAG
from datetime import datetime
from google.cloud import storage
from dotenv import load_dotenv
from sqlalchemy import text

from src.db_connection import *

load_dotenv()

def db_to_schema():
    postgres_conn_string = os.getenv("postgres_conn_string")
    engine = connect_with_db()
    with engine.begin() as connection:
        try:
            query = text("""CREATE TABLE IF NOT EXISTS productimages ( parent_asin TEXT, thumb TEXT, hi_res TEXT, large_res TEXT );  """)
            result = connection.execute(query)
            query = text("""CREATE TABLE IF NOT EXISTS productmetadata ( parent_asin TEXT, title TEXT, average_rating TEXT, rating_number TEXT, features TEXT, description TEXT, price TEXT, store TEXT, details TEXT, main_category TEXT );""")
            result = connection.execute(query)
            query = text("""CREATE TABLE IF NOT EXISTS productcategories ( parent_asin TEXT, categories TEXT );""")
            result = connection.execute(query)
            query = text("""CREATE TABLE IF NOT EXISTS userreviews ( rating TEXT, title TEXT, text TEXT, asin TEXT, parent_asin TEXT, user_id TEXT, timestamp TEXT, helpful_vote TEXT, verified_purchase TEXT );""")
            result = connection.execute(query)


            query = text("""DELETE FROM public.metadata m WHERE images NOT LIKE '%https://%';""")
            result = connection.execute(query)
            query = text("""UPDATE public.metadata SET images = REPLACE(images, '''', '"') WHERE images IS NOT NULL;""")
            result = connection.execute(query)
            query = text("""UPDATE public.metadata SET images = REPLACE(images, 'None', 'null') WHERE images IS NOT NULL;""")
            result = connection.execute(query)
            query = text("""INSERT INTO productimages (parent_asin, thumb, hi_res, large_res) SELECT parent_asin, COALESCE(images::jsonb -> 0 ->> 'thumb', '') AS thumb, COALESCE(images::jsonb -> 0 ->> 'hi_res', '') AS hi_res, COALESCE(images::jsonb -> 0 ->> 'large', '') AS large_res FROM public.metadata;""")
            result = connection.execute(query)

            query = text("""INSERT INTO productmetadata ( parent_asin, title, average_rating, rating_number, features, description, price, store, details, main_category ) SELECT parent_asin, title, average_rating, rating_number, features, description, price, store, details, main_category FROM public.metadata;""")
            result = connection.execute(query)
            query = text("""INSERT INTO productcategories ( parent_asin, categories ) SELECT parent_asin, categories FROM public.metadata;""")
            result = connection.execute(query)
            query = text("""INSERT INTO userreviews ( rating, title, text, asin, parent_asin, user_id, timestamp, helpful_vote, verified_purchase ) SELECT rating, title, text, asin, parent_asin, user_id, timestamp, helpful_vote, verified_purchase FROM public.user_reviews;""")
            result = connection.execute(query)

        except Exception as e:
            message = f"Error during insert: {e}"
```

Listing 6: db_to_schema.py

```
def review_trends_volume(df_ur):
    monthly_review_counts = df_ur.groupby('year_month').size()
    plt.figure(figsize=(12, 6))
    # Number of Reviews per month
    monthly_review_counts.plot(kind='line', marker='o', color='b')
    plt.title('Number of Reviews per Month')
    plt.xlabel('Month')
    plt.ylabel('Number of Reviews')
    plt.grid(True)
    plt.tight_layout()
    image_path = os.path.join('static', 'review_trends_volume.png')
    plt.savefig(image_path)
    plt.close()
    return image_path

def review_trends_avgRating(df_ur):
    monthly_avg_rating = df_ur.groupby('year_month')['rating'].mean()
    plt.figure(figsize=(12, 6))
    # Average Rating per Month
    monthly_avg_rating.plot(kind='line', marker='o', color='r')
    plt.title('Average Rating per Month')
    plt.xlabel('Month')
    plt.ylabel('Average Rating')
    plt.grid(True)
    plt.tight_layout()
    image_path = os.path.join('static', 'review_trends_avgRating.png')
    plt.savefig(image_path)
    plt.close()
    return image_path

```

```python
def products_price_distribution(df_pm):
    df_pm['price'] = pd.to_numeric(df_pm['price'], errors='coerce')
    df_pm = df_pm[df_pm['price'].apply(lambda x: x.is_integer() if pd.notnull(x) else False)]
    plt.figure(figsize=(15, 6))
    sns.boxplot(x=df_pm['price'], palette='viridis')
    plt.xlim(0, 700)
    plt.title('Price Distribution of Products')
    plt.xlabel('Price ($)')
    plt.grid(True)
    plt.tight_layout()

def most_common_categories(df_pm):
    df_pm['categories'] = df_pm['categories'].apply(lambda x: ast.literal_eval(x) if isinstance(x, str) else x)
    df_pm['categories'] = df_pm['categories'].apply(lambda x: x if x != '[]' else None)
    category_counts = df_pm['categories'].explode().value_counts()
    plt.figure(figsize=(10, 6))
    category_counts.head(10).plot(kind='pie', autopct='%1.1f%%', startangle=90, cmap='tab20')
    plt.title('Distribution of Main Product Categories')
    plt.ylabel('')
    plt.tight_layout()
    image_path = os.path.join('static', 'most_common_categories.png')
    plt.savefig(image_path)
    plt.close()
    return image_path

def rating_distribution_by_category(df_ur,df_pm):
    df_merged = pd.merge(df_ur, df_pm[['parent_asin', 'main_category']], left_on='parent_asin', right_on='parent_asin', how='left')
    df_merged = df_merged.dropna(subset=['main_category'])
    plt.figure(figsize=(12, 8))
    sns.boxplot(x='main_category', y='rating', data=df_merged, palette='viridis')
    plt.xticks(rotation=45, ha='right')
    plt.title('Rating Distribution by Product Category', fontsize=16)
    plt.xlabel('Product Category', fontsize=12)
    plt.ylabel('Rating', fontsize=12)
    plt.grid(True)
    plt.tight_layout()
    image_path = os.path.join('static', 'rating_distribution_by_category.png')
    plt.savefig(image_path)
    plt.close()
    return image_path

def avg_rating_per_product(df_pm):
    product_ratings = df_pm.groupby('parent_asin')['average_rating'].mean().sort_values(ascending=False)
    # Bar chart of average ratings for the top 20 products
    plt.figure(figsize=(12, 6))
    sns.barplot(x=product_ratings.head(20).index, y=product_ratings.head(20).values, color='purple')
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Product Asin')
    plt.ylabel('Average Rating')
    plt.title('Top 20 Products by Average Rating')
    plt.tight_layout()
    image_path = os.path.join('static', 'top20products.png')
    plt.savefig(image_path)
    plt.close()
    return image_path
```

Listing 7: Visualization

B Appendix B: Tables and Figures

Feature Name	**Feature Data Type**	**Feature Description**
rating	Float	Rating of the product
title	String	Title of the user review
text	String	Main body of the user review
images	List	URL of images posted by the user
asin	String	ID of the product
parent_asin	String	Parent ID of the product
user_id	String	ID of the user who submitted the review
timestamp	Integer	Time of submission of the review
verified_purchase	Boolean	User purchase verification
helpful_vote	Integer	Number of users who found the review helpful

Table 1: User Review Dataset

Feature Name	**Feature Data Type**	**Feature Description**
main_category	String	Domain of the product; in this case, handmade products.
title	String	Name of the product.
average_rating	Float	Average rating of the product based on all user reviews.
rating_number	Integer	Total number of ratings the product has received.
features	List	List of key features or selling points of the product.
description	List	Detailed description of the product.
price	Float	Price in US dollars.
images	List	URL of images of the product.
videos	List	URL of videos of the product.
store	String	Name of the store or brand that sells the product.
categories	List	List of categories under which the product is classified.
details	Dictionary	Product details including materials, brand, sizes, etc.
parent_asin	String	Parent ID of the product.
bought_together	List	Suggestions for similar products.

Table 2: Item Metadata Dataset

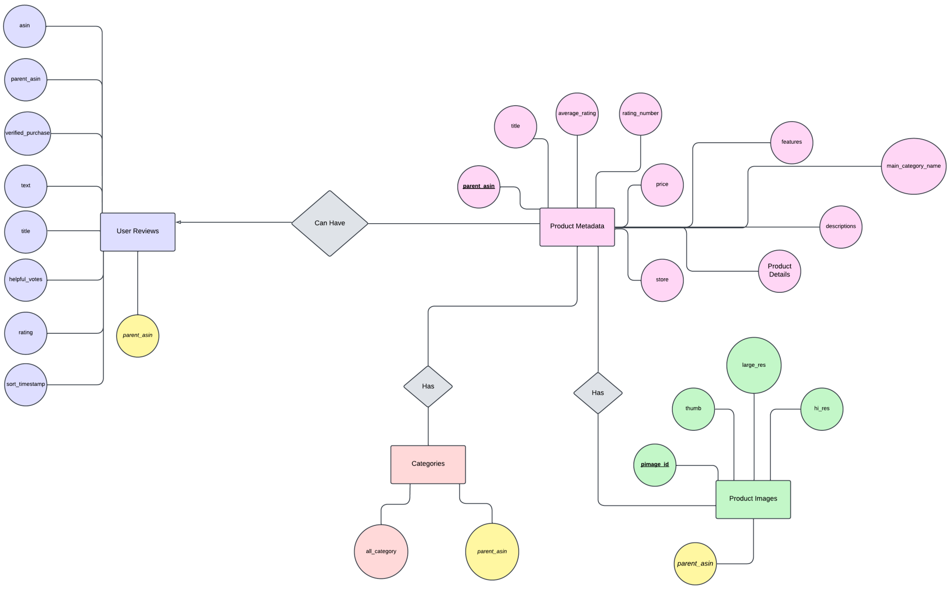


Figure 1: Entity-Relationship Diagram (ERD)

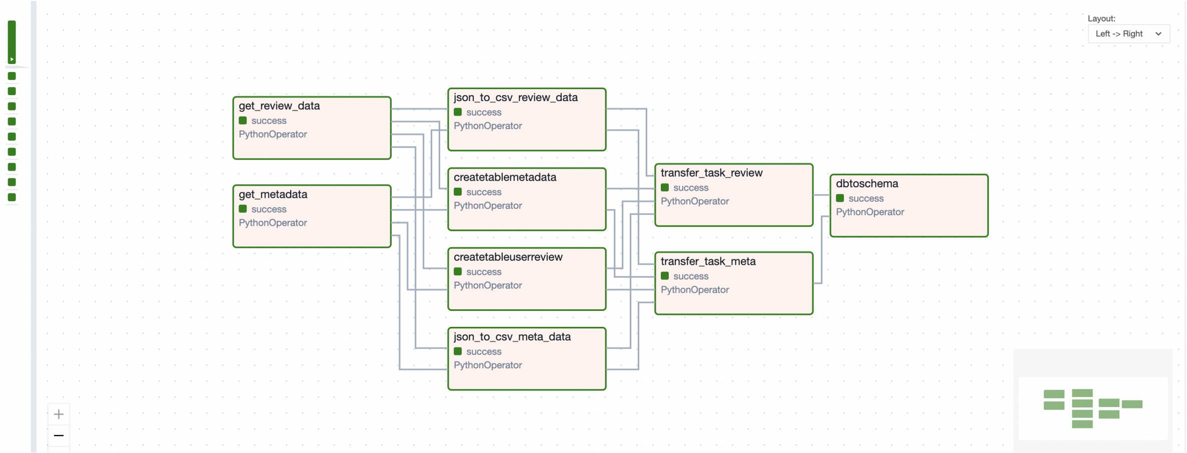


Figure 2: Data Pipeline Diagram

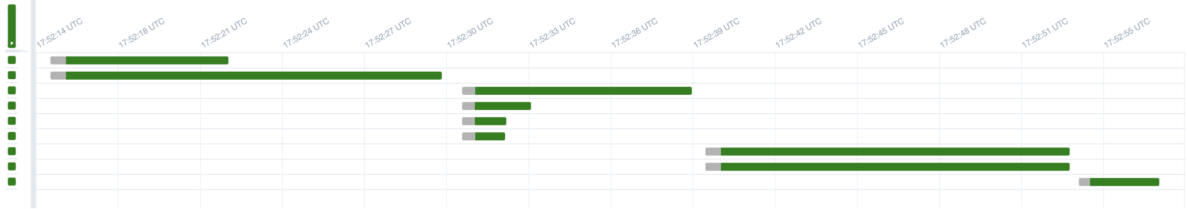
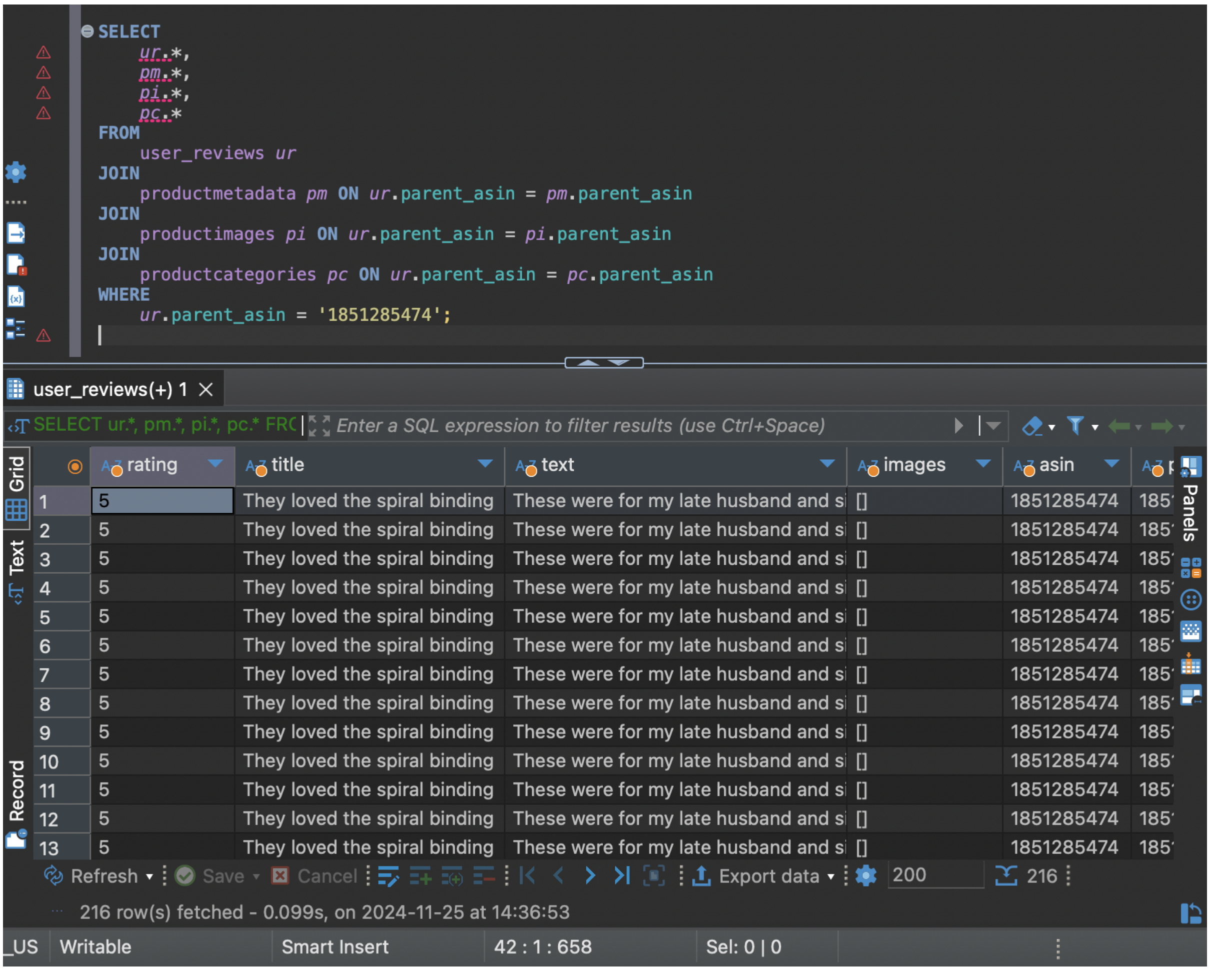


Figure 3: Data Pipeline Gantt Chart

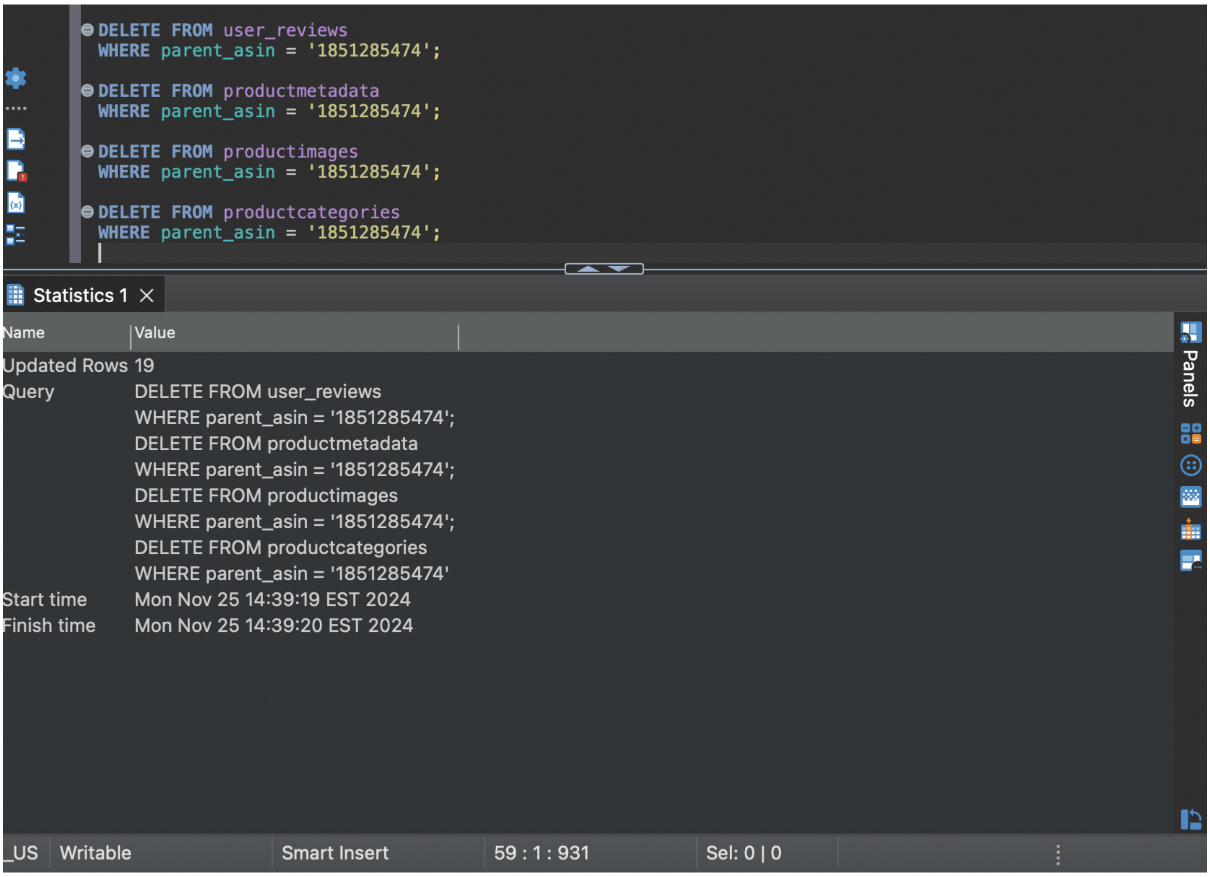


```sql
SELECT
    ur.*,
    pm.*,
    pi.*,
    pc.*
FROM
    user_reviews ur
JOIN
    productmetadata pm ON ur.parent_asin = pm.parent_asin
JOIN
    productimages pi ON ur.parent_asin = pi.parent_asin
JOIN
    productcategories pc ON ur.parent_asin = pc.parent_asin
WHERE
    ur.parent_asin = '1851285474';
```

	rating	title	text	images	asin	p
1	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
2	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
3	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
4	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
5	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
6	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
7	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
8	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
9	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
10	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
11	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
12	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185
13	5	They loved the spiral binding	These were for my late husband and s	[]	1851285474	185

216 row(s) fetched - 0.099s, on 2024-11-25 at 14:36:53

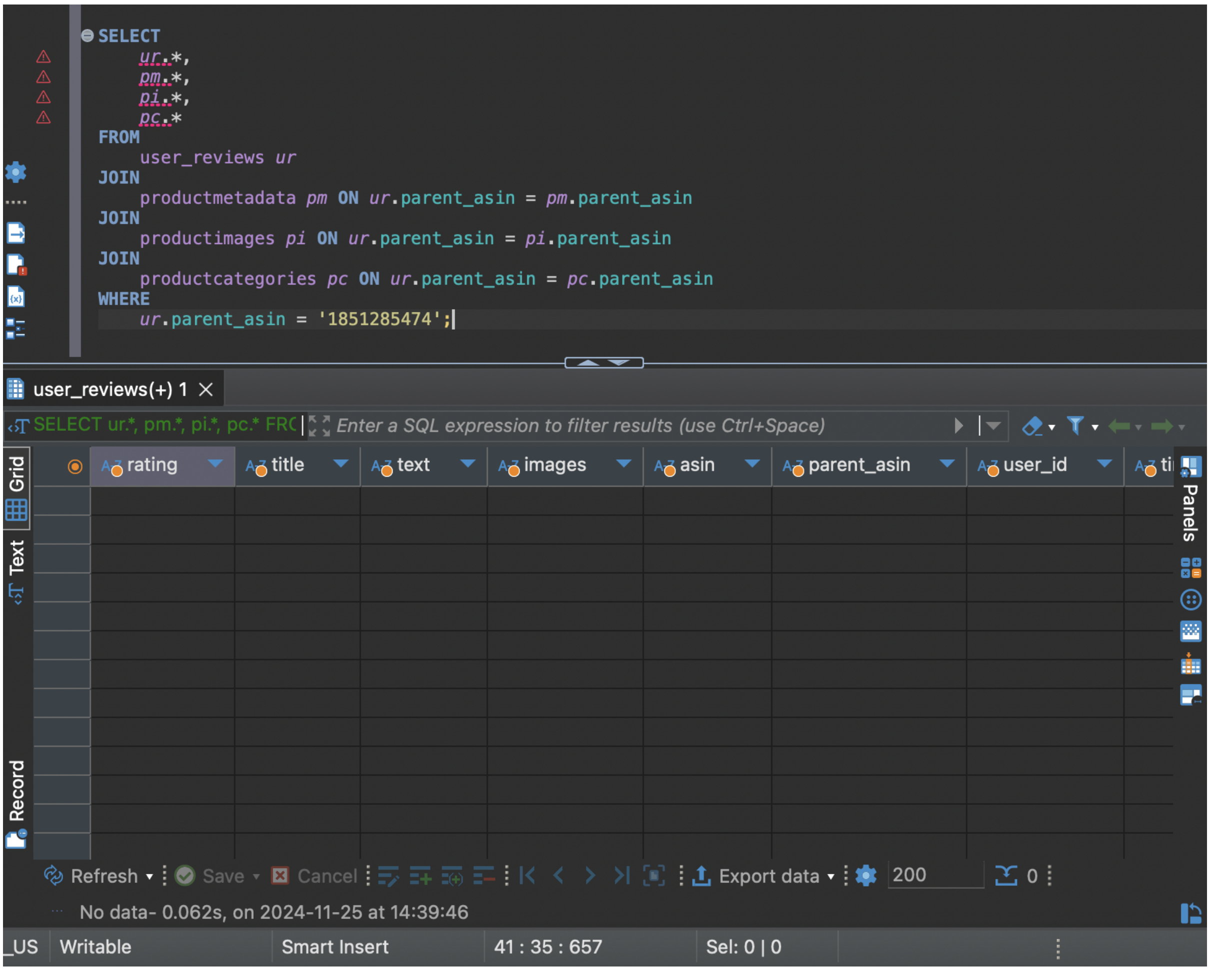
Figure 4: SQL Query - Before Removing Data



```sql
DELETE FROM user_reviews
WHERE parent_asin = '1851285474';

DELETE FROM productmetadata
WHERE parent_asin = '1851285474';

DELETE FROM productimages
WHERE parent_asin = '1851285474';

DELETE FROM productcategories
WHERE parent_asin = '1851285474';
```

```
Name         Value
Updated Rows 19
Query        DELETE FROM user_reviews
             WHERE parent_asin = '1851285474';
             DELETE FROM productmetadata
             WHERE parent_asin = '1851285474';
             DELETE FROM productimages
             WHERE parent_asin = '1851285474';
             DELETE FROM productcategories
             WHERE parent_asin = '1851285474'
Start time   Mon Nov 25 14:39:19 EST 2024
Finish time  Mon Nov 25 14:39:20 EST 2024
```

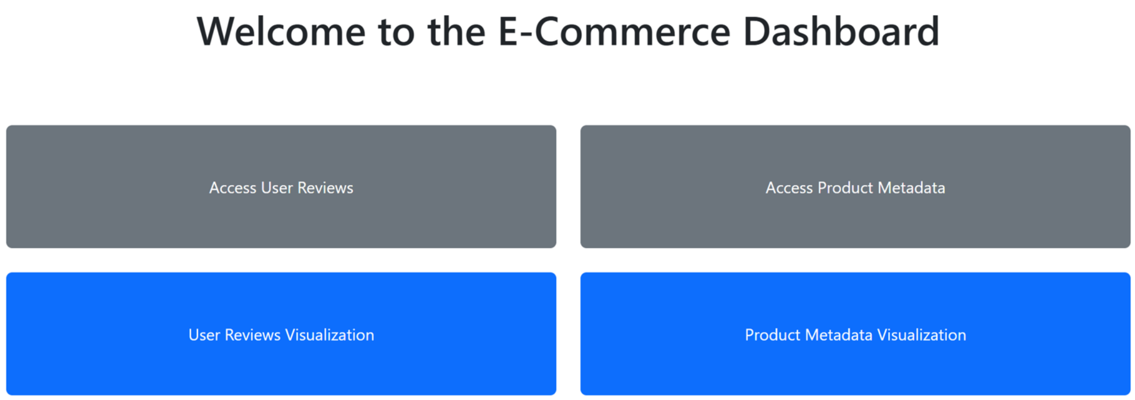
Figure 5: SQL Query - Removing Data



```
SELECT
    ur.*,
    pm.*,
    pi.*,
    pc.*
FROM
    user_reviews ur
JOIN
    productmetadata pm ON ur.parent_asin = pm.parent_asin
JOIN
    productimages pi ON ur.parent_asin = pi.parent_asin
JOIN
    productcategories pc ON ur.parent_asin = pc.parent_asin
WHERE
    ur.parent_asin = '1851285474';
```

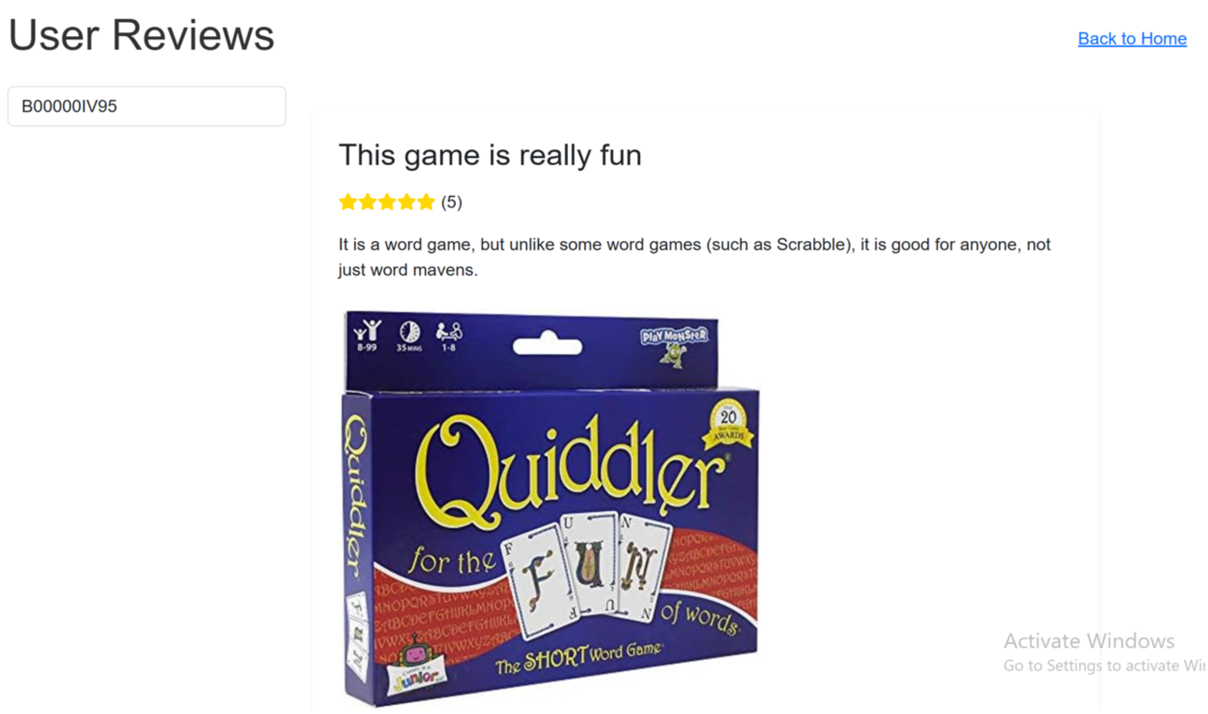
```
No data- 0.062s, on 2024-11-25 at 14:39:46
```

Figure 6: SQL Query - After Removing Data



Welcome to the E-Commerce Dashboard

Figure 7: Control Panel



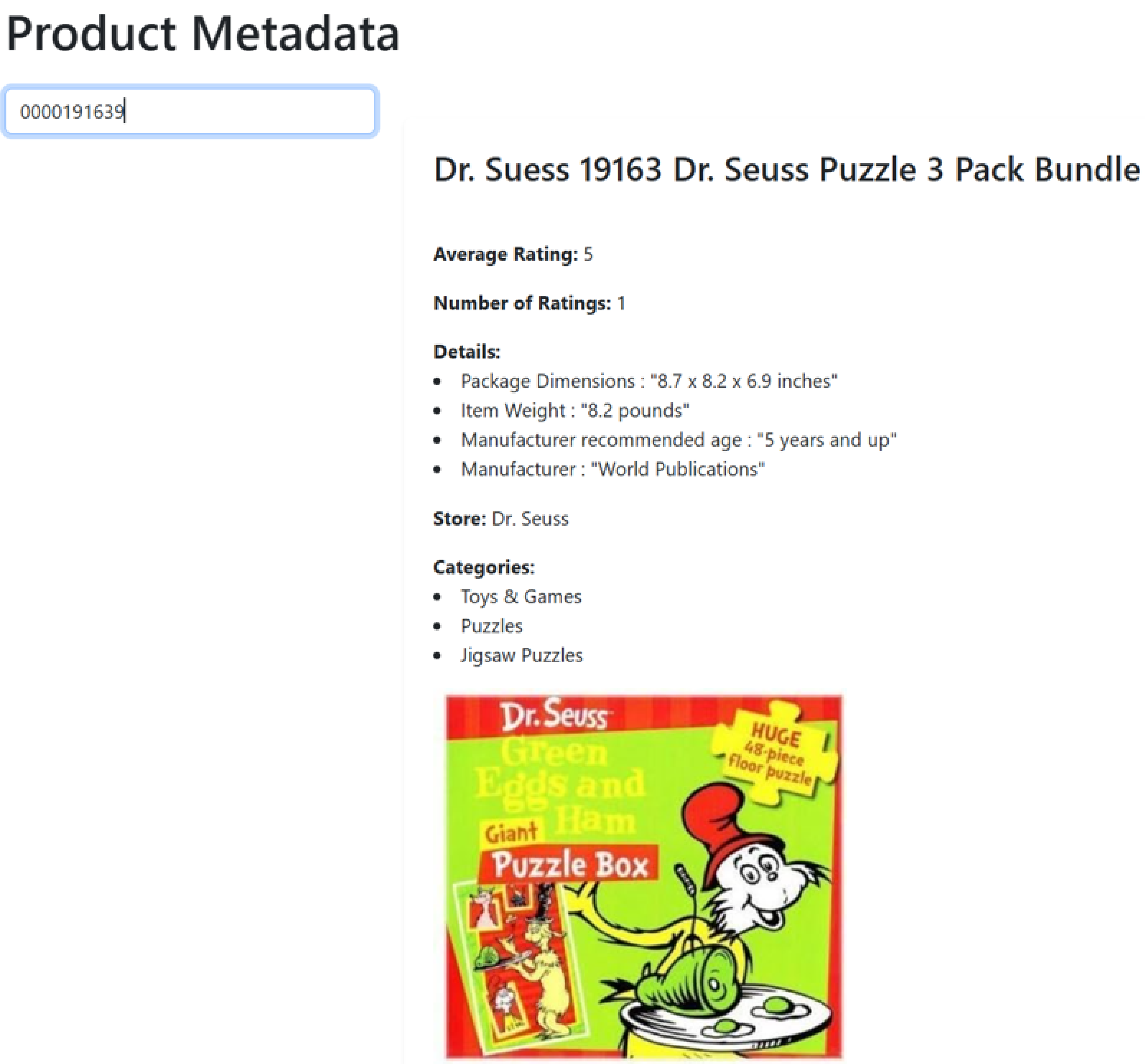
User Reviews

This game is really fun

★★★★★ (5)

It is a word game, but unlike some word games (such as Scrabble), it is good for anyone, not just word mavens.

Figure 8: Access User Reviews



Product Metadata

0000191639

Dr. Suess 19163 Dr. Seuss Puzzle 3 Pack Bundle

Average Rating: 5

Number of Ratings: 1

Details:

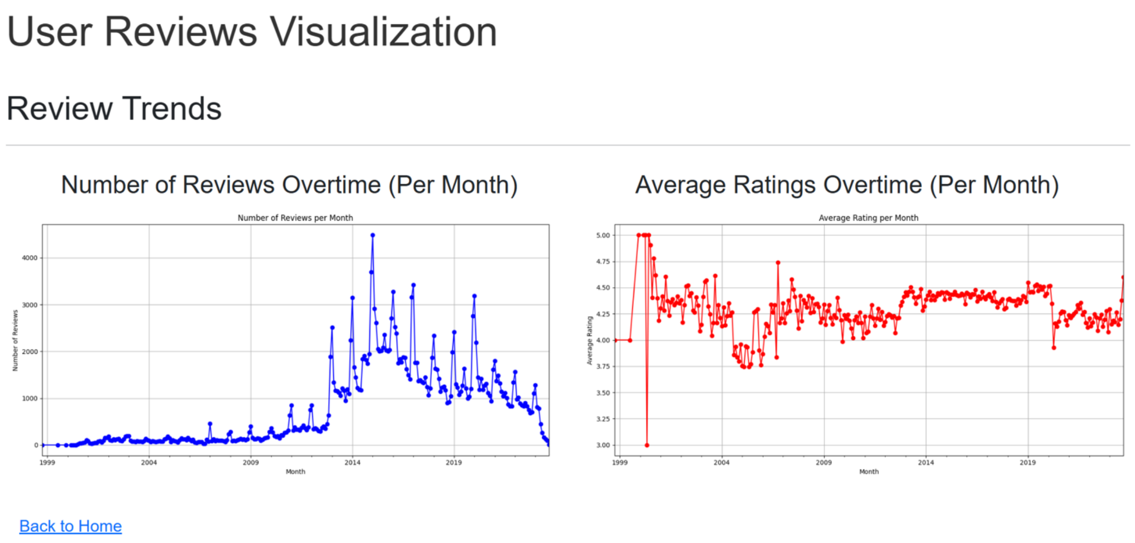
- Package Dimensions : "8.7 x 8.2 x 6.9 inches"
- Item Weight : "8.2 pounds"
- Manufacturer recommended age : "5 years and up"
- Manufacturer : "World Publications"

Store: Dr. Seuss

Categories:

- Toys & Games
- Puzzles
- Jigsaw Puzzles

Figure 9: Access Product Metadata



User Reviews Visualization

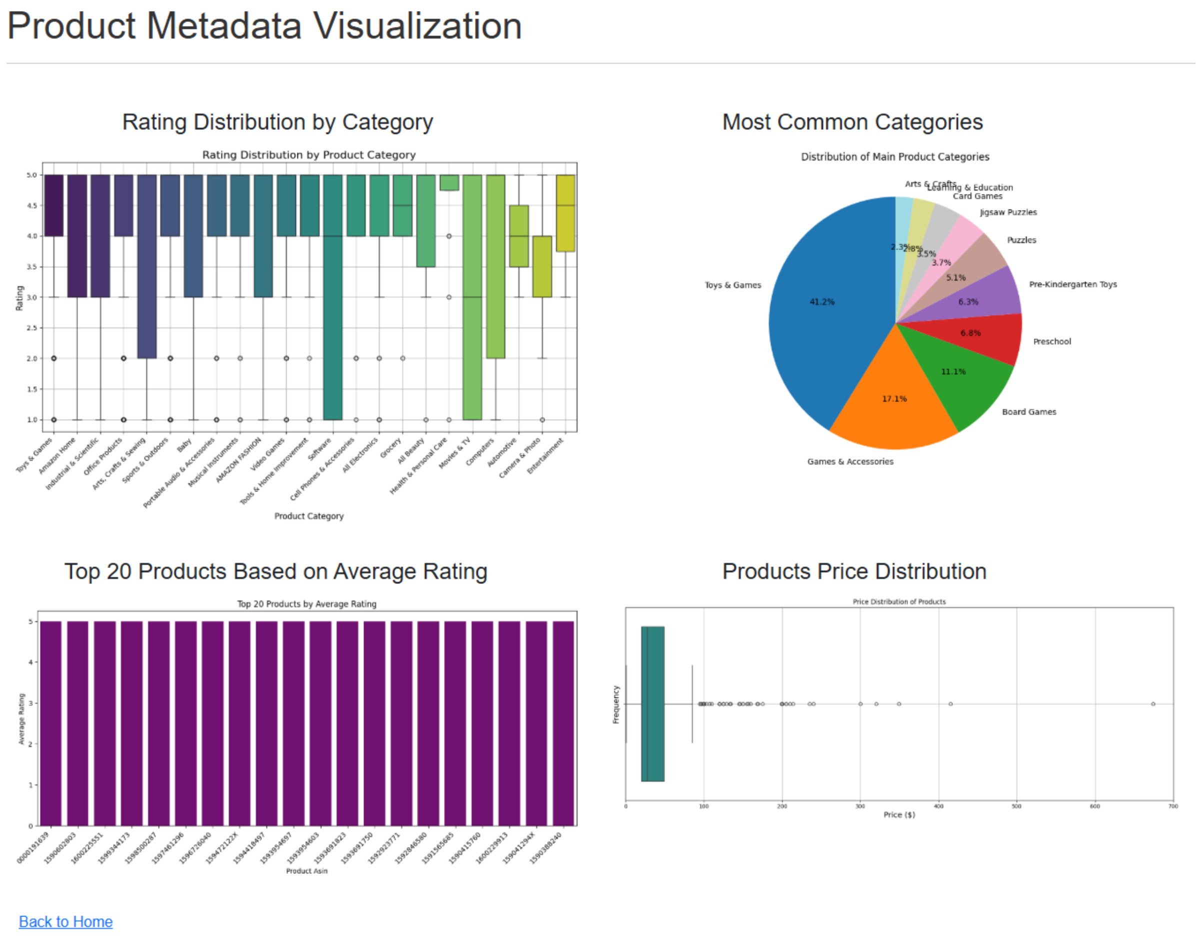
Review Trends

Number of Reviews Overtime (Per Month)

Average Ratings Overtime (Per Month)

Back to Home

Figure 10: User Reviews Visualization



Product Metadata Visualization

Rating Distribution by Category

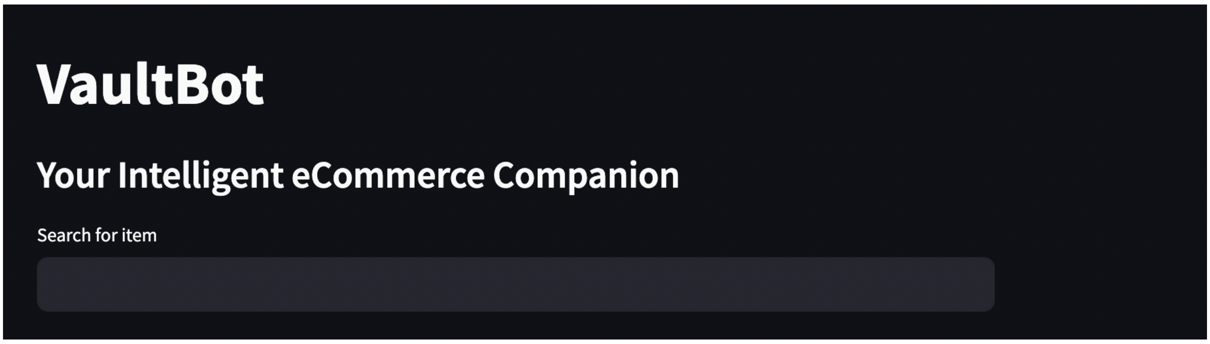
Most Common Categories

Top 20 Products Based on Average Rating

Products Price Distribution

Back to Home

Figure 11: Product Metadata Visualization

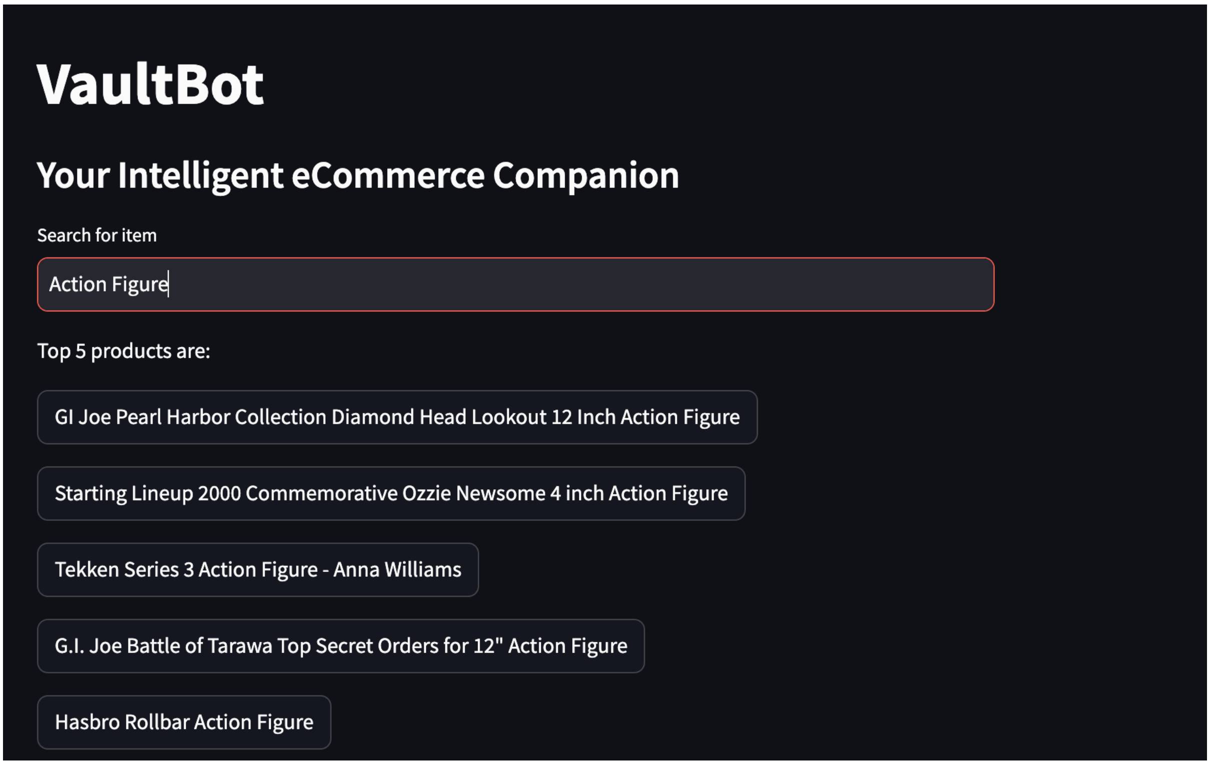


VaultBot

Your Intelligent eCommerce Companion

Search for item

Figure 12: VaultBot Initial Screen



VaultBot

Your Intelligent eCommerce Companion

Search for item

Action Figure

Top 5 products are:

GI Joe Pearl Harbor Collection Diamond Head Lookout 12 Inch Action Figure

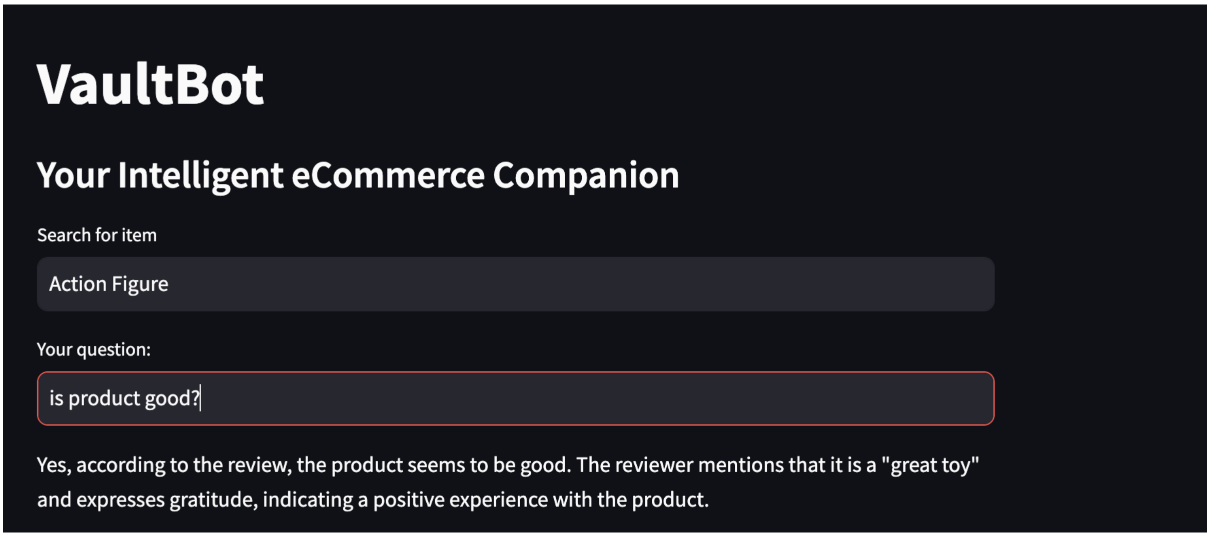
Starting Lineup 2000 Commemorative Ozzie Newsome 4 inch Action Figure

Tekken Series 3 Action Figure - Anna Williams

G.I. Joe Battle of Tarawa Top Secret Orders for 12" Action Figure

Hasbro Rollbar Action Figure

Figure 13: VaultBot Item Search



VaultBot

Your Intelligent eCommerce Companion

Search for item

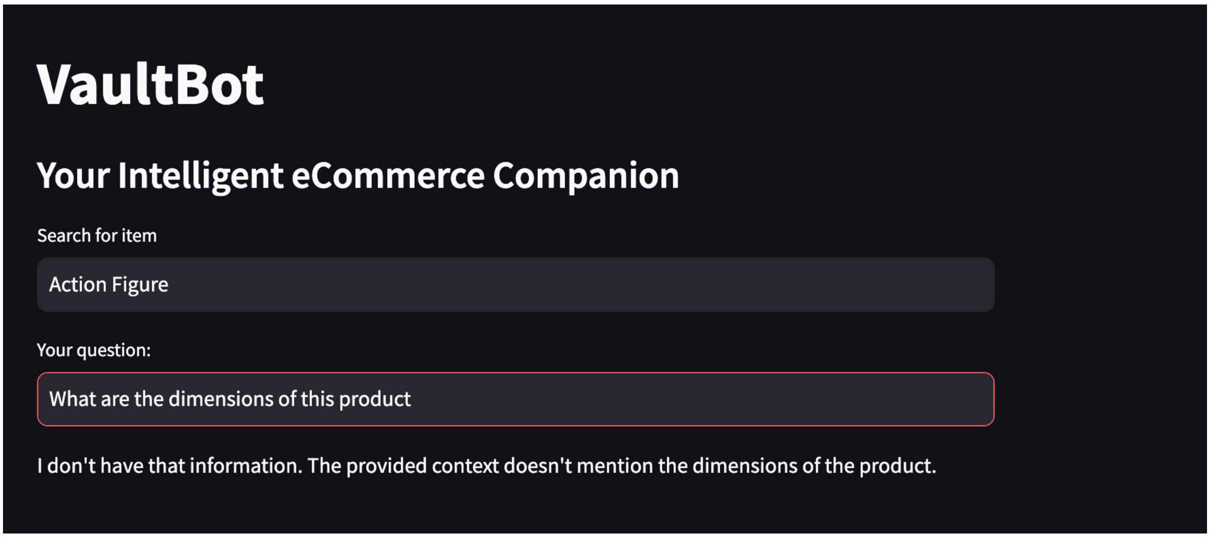
Action Figure

Your question:

is product good?

Yes, according to the review, the product seems to be good. The reviewer mentions that it is a "great toy" and expresses gratitude, indicating a positive experience with the product.

Figure 14: VaultBot Asking Question



VaultBot

Your Intelligent eCommerce Companion

Search for item

Action Figure

Your question:

What are the dimensions of this product

I don't have that information. The provided context doesn't mention the dimensions of the product.

Figure 15: VaultBot Asking Question No Answer

C Appendix C: Citations

The dataset is referenced in the following academic publication:

- Hou, Yupeng, Li, Jiacheng, He, Zhankui, Yan, An, Chen, Xiusi, and McAuley, Julian. ”Bridging Language and Items for Retrieval and Recommendation.” arXiv preprint arXiv:2403.03952 (2024).

  @article{hou2024bridging,

  title={Bridging Language and Items for Retrieval and Recommendation},

  author={Hou, Yupeng and Li, Jiacheng and He, Zhankui and Yan, An and Chen, Xiusi and McAuley, Julian},

  journal={arXiv preprint arXiv:2403.03952},

  year={2024} }

- Chen, H., Chiang, R. H., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. MIS Quarterly, 36(4), 1165–1188. https://doi.org/10.2307/41703503

  @articlechen2012business, author = Chen, H. and Chiang, R. H. and Storey, V. C., title = Business intelligence and analytics: From big data to big impact, journal = MIS Quarterly, volume = 36, number = 4, pages = 1165–1188, year = 2012, doi = 10.2307/41703503

- Faisal, M. I., Hossain, M. A., & Rahman, M. S. (2020). Automated ETL pipeline for data integration in cloud data warehousing. Journal of Cloud Computing: Advances, Systems and Applications, 9(1), 34-47. https://doi.org/10.1186/s13677-020-00214-w @articlefaisal2020automated, author = Faisal, M. I. and Hossain, M. A. and Rahman, M. S., title = Automated ETL pipeline for data integration in cloud data warehousing, journal = Journal of Cloud Computing: Advances, Systems and Applications, volume = 9, number = 1, pages = 34-47, year = 2020, doi = 10.1186/s13677-020-00214-w

- Stonebraker, M., Çetintemel, U., & Zdonik, S. (2018). The design of modern data management systems. ACM Computing Surveys, 50(2), 1-41. https://doi.org/10.1145/3184738 @articlestonebraker2018design, author = Stonebraker, M. and Çetintemel, U. and Zdonik, S., title = The design of modern data management systems, journal = ACM Computing Surveys, volume = 50, number = 2, pages = 1–41, year = 2018, doi = 10.1145/3184738