DS5110: Essentials of Data Science

E-commerce Product Recommendations and Pattern Analysis

Keith Fernandes, Rutuja Jadhav, Siddhi Nirmale

GitHub Repository:
https://github.com/siddhi07/E-commerce-Product-Recommendations-and-Pattern-Analysis.

INDEX

List of Figures

SQL Queries

1. Abstract

Traditional retail recommendation systems rely primarily on purchase history without incorporating customer satisfaction metrics, often resulting in suggestions that include poorly reviewed products. This project addresses this limitation by developing an integrated analytical framework that combines market basket analysis with sentiment analysis to generate intelligent, quality-filtered product recommendations.

The methodology employs a two-pronged approach: first, the Apriori algorithm identifies frequent itemset associations from 38,765 grocery transactions spanning 167 product categories, generating association rules evaluated through support, confidence, and lift metrics. Second, a pre-trained DistilBERT transformer model classifies sentiment from 4,166 Amazon product reviews, achieving meaningful discrimination across positive (70.6%), negative (29.4%), and neutral categories. Data integration is accomplished through a unified SQLite database schema linking transactional patterns with product-level sentiment scores via item ID mapping.

Results demonstrate that the hybrid approach successfully identifies co-purchase patterns while filtering recommendations by customer satisfaction. Cross-validation between star ratings and BERT-predicted sentiment confirms model reliability, with average sentiment scores increasing monotonically from -0.87 for 1-star reviews to +0.72 for 5-star reviews. The analysis reveals category-level sentiment variations, with cereals and coffee achieving consistently positive sentiment while certain categories exhibit sentiment-rating misalignment.

The final deliverable is an interactive dashboard enabling non-technical stakeholders to explore product associations, sentiment distributions, and personalized recommendations. This work demonstrates the value of combining transactional behavior analysis with natural language processing to create more nuanced recommendation systems than either approach achieves independently.

Keywords: *Market Basket Analysis, Sentiment Analysis, Apriori Algorithm, DistilBERT, Product Recommendations, Association Rule Mining*

2. Problem Statement and Objectives

2.1. Problem Statement

Retailers face significant challenges in understanding customer purchasing behavior and providing personalized product recommendations. Traditional recommendation systems often rely solely on purchase history without considering customer satisfaction with specific products or brands. This gap results in recommendations that may include poorly reviewed products, diminishing customer trust and satisfaction.

2.2. Project Objectives

The primary objectives of this project are:

1. **Identify purchasing patterns** using association rule mining to discover which products are frequently bought together
2. **Analyze customer sentiment** from product reviews to understand brand-level satisfaction
3. **Generate targeted recommendations** by combining transactional behavior with sentiment scores
4. **Deliver actionable insights** through an intuitive, interactive dashboard accessible to non-technical users

2.3. Expected Outcome

An interactive dashboard displaying product association networks, key metrics (support, confidence, lift), and sentiment-filtered recommendations. The dashboard enables effortless navigation and interpretation of insights for users without technical backgrounds.

3. Data Description

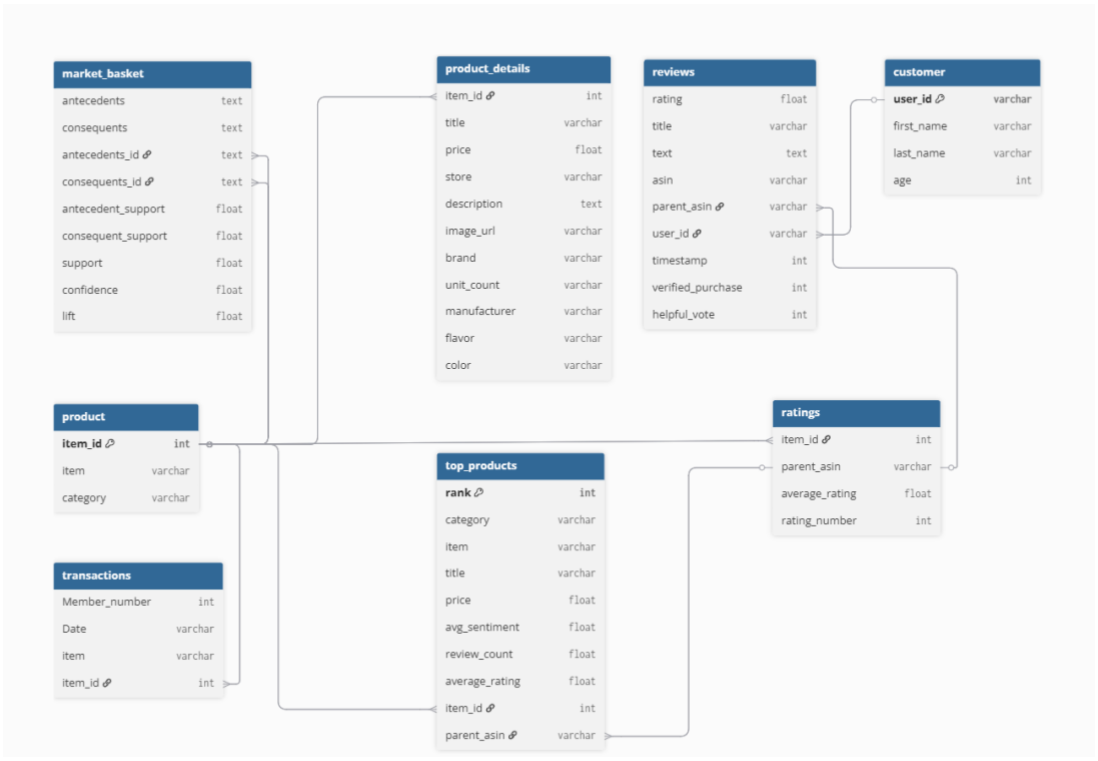
3.1. Dataset Sources: The project utilizes two primary datasets:

1. **Groceries Dataset (Kaggle)**
   - **Source:** https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset
   - **Size:** 38,765 rows of transactional data
   - **Time Period:** 2015
   - **Structure**: Three columns (Member_number, Date, itemDescription)
   - Each row represents a single item purchase; shopping trips span multiple rows

2. **Amazon Product Reviews Dataset**
   - **Contains:** Product IDs, customer reviews, ratings, and user information
   - **Tables:** reviews, product, product_details, ratings, transactions, customer
   - **Reviews extracted:** 4,166 rows
   - **Products catalogued:** 1,604 items

3.2. Database Schema: The project integrates data across six relational tables:

- **reviews:** text, title, rating, user_id, parent_asin
- **product:** item_id, item, category
- **ratings:** parent_asin, item_id (bridge table)
- **transactions:** Member_number, Date, item, item_id
- **customer:** customer demographic information
- **product_details:** extended product attributes



4. Data Preprocessing and Feature Engineering

4.1. Data Extraction: Data was extracted from a SQLite database using SQL queries and loaded into pandas
DataFrames:

```
reviews_df = pd.read_sql_query(
    "SELECT text, title, rating, user_id, parent_asin FROM reviews",
    conn
)
product_df = pd.read_sql_query(
    "SELECT item_id, item, category FROM product",
    conn
)
```

4.2. Missing Value Analysis: A systematic check for data quality issues revealed:

- **Missing values:** 0 null values across all columns
- **Empty strings:** 0 empty reviews
- **Whitespace-only entries:** 1 row identified (retained since title contained valid text)

4.3. Text Preprocessing Pipeline: A comprehensive text cleaning function was developed to prepare review text for sentiment analysis:

```
def clean_text(text):
    # Convert to lowercase
    text = text.lower()
    # Handle emoticons (convert to sentiment words)
    emoticon_dict = {
        r':\)': ' positive ', r':-\)': ' positive ', r':D': ' positive ',
        r':\(': ' negative ', r':-\(': ' negative '
    }
    for emoticon, sentiment in emoticon_dict.items():
        text = re.sub(emoticon, sentiment, text, flags=re.IGNORECASE)
    # Remove special characters and punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

4.4. Feature Engineering

Combined Text Feature: Title and text fields were merged for comprehensive sentiment analysis, ensuring no data loss when one field was empty:

```
reviews_df['combined_text'] = reviews_df['cleaned_title'] + " " +
reviews_df['cleaned_text']
```

Data Linkage: Reviews were linked to products through a multi-step join process:

- reviews.parent_asin → ratings.parent_asin (obtain item_id)
- ratings.item_id → product.item_id (obtain product information)

5. Data Cleaning and Database Creation

5.1. Overview and Objectives

This notebook performs comprehensive data preprocessing, cleaning, and database creation to prepare the foundational data infrastructure for the E-commerce Product Recommendations project. The notebook integrates multiple data sources grocery transaction data and Amazon product metadata into a unified SQLite database that supports both market basket analysis and sentiment analysis components.

5.2. Input Datasets

Three primary datasets are required to run this notebook:

Dataset	Description	Purpose
Groceries_dataset.csv	Transaction records from grocery store	Market basket analysis
meta_Grocery_and_Gourmet_Food.csv	Amazon product metadata	Product details & sentiment analysis
Reviews_Grocery_and_Gourmet_Food.csv	Amazon customer reviews	Sentiment analysis

5.3. Data Loading and Initial Exploration

1. Groceries Transaction Dataset

```
df_market_basket = pd.read_csv('/content/Groceries_dataset.csv')
```

Dataset Characteristics:

- **Total Records:** 38,765 rows
- **Columns:** Member_number, Date, itemDescription
- **Unique Customers:** 3,898 members
- **Unique Items:** 167 product categories
- **Date Range:** Throughout 2015 (728 unique dates)

Sample Items: tropical fruit, whole milk, pip fruit, other vegetables, rolls/buns, pot plants, citrus fruit, beef, frankfurter, chicken, butter, yogurt, sausage, brown bread, root vegetables, coffee, pastry, berries, and 150+ more categories.

2. Amazon Product Metadata Dataset

```
df_sentiment = pd.read_csv('/content/meta_Grocery_and_Gourmet_Food.csv')
```

Dataset Characteristics (Before Filtering):

- **Total Records:** 5,672 rows
- **Key Columns:** parent_asin, main_category, title, average_rating, rating_number, price, store, description, brand, manufacturer, flavor, color

3. Amazon Reviews Dataset

```
df_sentiment_reviews = pd.read_csv('/content/Reviews_Grocery_and_
Gourmet_Food.csv')
```

Dataset Characteristics:

- **Total Records:** 20,011 reviews

- **Columns:** rating, title, text, images, asin, parent_asin, user_id, timestamp, verified_purchase, helpful_vote

5.4. Data Preprocessing

Category Extraction from Amazon Data

The hierarchical category path was parsed to extract items and category names:

```
df_sentiment['item'] = df_sentiment['categories'].str.split('>').str[-
1].str.strip()
df_sentiment['category'] = df_sentiment['categories'].str.split('>').str[-
2].str.strip()
df_sentiment = df_sentiment.drop('categories', axis=1)
```

This transformation converts paths like *"Grocery & Gourmet Food > Beverages > Coffee"* into:

- **item:** "Coffee"
- **category:** "Beverages"

Dataset Integration via Partial Matching

A critical step was filtering the Amazon product data to retain only items that correspond to categories in the grocery transaction dataset. This was achieved using partial string matching:

```
def matches_any(item, items_list):
    item_lower = str(item).lower()
    return any(item.lower() in item_lower or item_lower in item.lower()
               for item in items_list)
market_basket_items = df_market_basket['itemDescription'].unique()
df_sentiment = df_sentiment[df_sentiment['item'].apply(
    lambda x: matches_any(x, market_basket_items))]
```

Filtering Results:

- **Before Filtering:** 5,672 rows
- **After Filtering:** 1,995 rows
- **Unique Items Retained:** 220 product types

Item ID Mapping System

A unified *item_id* system was created to link transaction data with sentiment data:

```
# Step 1: Create ID mapping from market basket items
unique_items = df_market_basket['itemDescription'].unique()
item_to_id = {item: idx for idx, item in enumerate(unique_items)}
# Step 2: Apply to market basket DataFrame
df_market_basket['item_id'] = df_market_basket['itemDescription'].map(item_to_id)
# Step 3: Apply to sentiment data with partial matching
def get_item_id(item, item_dict, items_list):
    item_lower = str(item).lower()
    for basket_item in items_list:
        basket_lower = basket_item.lower()
        if basket_lower in item_lower or item_lower in basket_lower:
            return item_dict.get(basket_item)
    return None
df_sentiment['item_id'] = df_sentiment['item'].apply(
    lambda x: get_item_id(x, item_to_id, unique_items))
```

Mapping Results:

- **Unique item_ids in Market Basket:** 167
- **Unique item_ids in Sentiment Data:** 67 (subset with matching products)

This mapping enables joining association rules with sentiment scores for integrated recommendations.

5.5. Table Structure Design

DataFrame Separation for Normalization

The combined sentiment data was split into normalized tables to reduce redundancy:

```
# Product catalog
df_product = df_sentiment[['item_id', 'item', 'category']]
df_product = df_product.drop_duplicates(subset='item_id')

# Ratings information
df_ratings = df_sentiment[['item_id', 'parent_asin', 'average_rating', 'rat-
ing_number']]

# Detailed product information
df_product_details = df_sentiment[['item_id', 'title', 'price', 'store',
    'description', 'image_url', 'brand', 'unit_count', 'manufacturer',
    'flavor', 'color']]
df_product_details = df_product_details.drop_duplicates(subset='item_id')
```

Customer Data Generation

Since the reviews dataset contained only anonymous user_ids, synthetic customer data was generated using the Faker library for demonstration purposes:

```python
from faker import Faker
import random

fake = Faker()
unique_user_ids = df_sentiment_reviews['user_id'].unique()

df_customer = pd.DataFrame({
    'user_id': unique_user_ids,
    'first_name': [fake.first_name() for _ in range(len(unique_user_ids))],
    'last_name': [fake.last_name() for _ in range(len(unique_user_ids))],
    'age': [random.randint(10, 100) for _ in range(len(unique_user_ids))]
})
```

5.6. Database Creation

PySpark Database Creation

The tables were first created using PySpark for scalability:

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Creating Databases") \
    .getOrCreate()

# Create database
spark.sql("CREATE DATABASE IF NOT EXISTS Products")

# Convert pandas to PySpark DataFrames and save as tables
df_transaction_spark.write.mode("overwrite").saveAsTable("Products.transactions")
df_product_spark.write.mode("overwrite").saveAsTable("Products.product")
df_ratings_spark.write.mode("overwrite").saveAsTable("Products.ratings")
df_sentiment_reviews_spark.write.mode("overwrite").saveAsTable("Products.re-
views")
df_product_details_spark.write.mode("overwrite").saveAsTable("Products.prod-
uct_details")
df_customer_spark.write.mode("overwrite").saveAsTable("Products.customer")
```

SQLite Export

The database was exported to SQLite format for portability:

```
import sqlite3

conn = sqlite3.connect('/content/products.db')

tables = ['customer', 'product', 'product_details', 'ratings', 'reviews', 'trans-
actions']

for table in tables:
    df = spark.sql(f"SELECT * FROM Products.{table}")
    df_pandas = df.toPandas()
    df_pandas.to_sql(table, conn, if_exists='replace', index=False)
    print(f"Exported {table} to products.db")

conn.close()
```

Output File: *products.db*

5.7. Final Database Schema

The products.db database contains six tables:

Table: *customer*

Column	Type	Description
user_id	TEXT	Unique customer identifier
first_name	TEXT	Customer first name
last_name	TEXT	Customer last name
Age	INTEGER	Customer age (10-100)

Table: *product*

Column	Type	Description
item_id	INTEGER	Unique product identifier
item	TEXT	Product name
category	TEXT	Product category

Table: *product_details*

Column	Type	Description
item_id	INTEGER	Foreign key to product
title	TEXT	Full product title
price	REAL	Product price
store	TEXT	Seller/store name
description	TEXT	Product description
image_url	TEXT	Product image URL
brand	TEXT	Brand name

unit_count	TEXT	Package size
manufacturer	TEXT	Manufacturer name
flavor	TEXT	Product flavor
color	TEXT	Product color

Table: *ratings*

Column	Type	Description
item_id	INTEGER	Foreign key to product
parent_asin	TEXT	Amazon product ID
average_rating	REAL	Average customer rating (1-5)
rating_number	INTEGER	Number of ratings

Table: *reviews*

Column	Type	Description
rating	REAL	Individual review rating
title	TEXT	Review title
text	TEXT	Review text content
asin	TEXT	Product ASIN
parent_asin	TEXT	Parent product ASIN
user_id	TEXT	Reviewer user ID
timestamp	REAL	Review timestamp
verified_purchase	BOOLEAN	Purchase verification status
helpful_vote	REAL	Helpful vote count

Table: *transactions*

Column	Type	Description
Member_number	INTEGER	Customer ID
Date	TEXT	Transaction date
item	TEXT	Item purchased
item_id	INTEGER	Foreign key to product

5.8. Data Quality Summary

Metric	Value
Total Transaction Records	38,765
Unique Customers (Transactions)	3,898
Unique Product Categories	167
Amazon Products (After Filtering)	1,995
Matched Product Categories	67
Customer Reviews	20,011

6. Market Basket Analysis using Apriori Algorithm

This notebook implements association rule mining using the Apriori algorithm to discover purchasing patterns within the Groceries dataset. The analysis identifies which products are frequently bought together, enabling data-driven product recommendations and retail optimization strategies.
The Apriori algorithm was selected for market basket analysis due to its clear rule generation and explicit calculation of interpretable metrics. Implementation used Python's mlxtend library.

6.1. Business Applications: The association rules support several retail optimization strategies

- **Product Placement:** Items with high association can be placed on the same shelf to prompt cross-purchases
- **Promotional Strategy:** Discounts can be applied strategically to one item in a frequently co-purchased pair
- **Targeted Advertising:** Advertisements for item X can target buyers who purchase item Y
- **Product Bundling:** Highly associated items can be combined into new product offerings

6.2. Methodology

Affinity analysis is a data mining technique that discovers co-occurrence relationships among activities performed by specific individuals or groups. In retail, this technique performs market basket analysis to understand customer purchase behavior and identify relationships between items that people buy together.

Association Rule Mining Metrics

Three key metrics are used to evaluate association rules:

Metric	**Formula**	**Interpretation**
Support	$P(X \cap Y)$	How popular an itemset is, measured by the proportion of transactions containing the itemset
Confidence	$P(Y\|X) = P(X \cap Y) / P(X)$	How likely item Y is purchased when item X is purchased
Lift	$P(X \cap Y) / (P(X) \times P(Y))$	How likely item Y is purchased when item X is purchased, controlling for Y's popularity. Lift > 1 indicates positive association; Lift < 1 indicates negative association; Lift = 1 indicates no association

6.4. Data Preparation

Database Connection

The analysis begins by connecting to the SQLite database (products.db), which contains the following tables:

- ***customer* -** Customer information
- ***product* -** Product catalog
- ***product_details*** - Extended product information
- ***ratings* -** Product ratings data
- ***reviews* -** Customer review text
- ***transactions* -** Purchase transaction records

Transaction Data Loading

```
df_transactions = pd.read_sql("SELECT * FROM transactions", conn)
```

Date Conversion

The Date column was converted from object format to proper datetime objects for temporal analysis:

```
df_transactions['Date'] = pd.to_datetime(df_transactions['Date'], format='mixed')
```

Transaction Grouping

Individual item purchases were grouped into transaction baskets by combining purchases made by the same member on the same date:

```
transactions = df_transactions.groupby(['Member_number', 'Date']
)['item'].apply(list).tolist()
```

Result: 14,963 unique transactions were created from the grouped data.

Transaction Encoding

The TransactionEncoder from mlxtend was used to convert the list-based transaction data into a binary matrix suitable for the Apriori algorithm:

```
te = TransactionEncoder()
te_arry = te.fit(transactions).transform(transactions)
transactions_df = pd.DataFrame(te_arry, columns=te.columns_)
```

This encoding transforms transactions like *['milk', 'bread', 'butter']* into a binary row where columns for milk, bread, and butter are True and all other item columns are False.

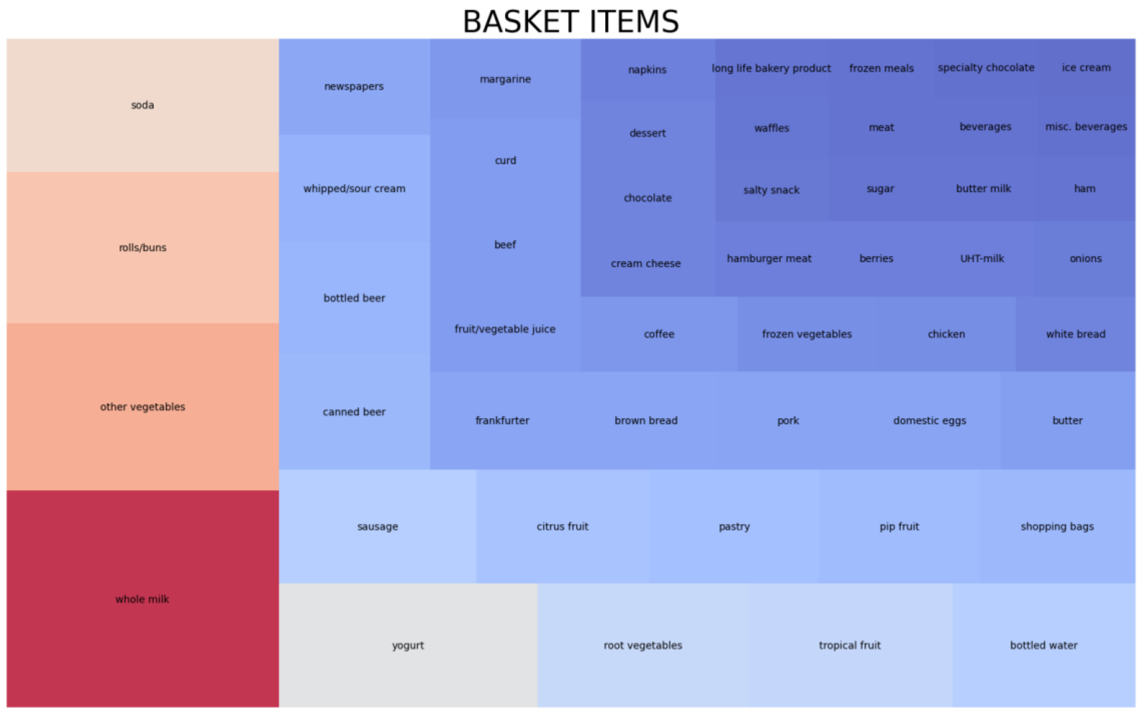
6.5. Exploratory Analysis

Item Purchase Frequency

A descriptive analysis was performed on the encoded transaction matrix to count individual item purchases across all 167 product categories. The purchase count for each item was calculated by subtracting the frequency of False values (item not purchased) from the total transaction count.

Top 50 Items Visualization

A TreeMap visualization was created using the Squarify library to display the relative popularity of the top 50 most frequently purchased items. The visualization uses a coolwarm color scheme where warmer colors indicate higher purchase frequencies.



Key Findings from Item Frequency Analysis:

- The visualization reveals which product categories dominate customer purchases
- High-frequency items like whole milk, vegetables, and rolls/buns appear prominently
- This baseline understanding informs interpretation of association rules

6.6. Apriori Algorithm Implementation

Parameter Configuration

```
frequent_itemsets = apriori(transactions, min_support=0.001, use_colnames=True,
max_len=5)
```

Parameter Justification:

- **min_support = 0.001 (0.1%)**
  A low threshold was selected to capture itemsets appearing in at least 0.1% of transactions, ensuring discovery of meaningful but less common associations
- **max_len = 5**
  Limits itemsets to a maximum of 5 items, balancing computational efficiency with discovery of complex purchasing patterns
- **use_colnames = True**
  Preserves item names for interpretability

Itemset Length Analysis

A length column was added to analyze the distribution of itemset sizes:

```
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
```

6.7. Association Rule Generation

Rule Extraction

Association rules were generated from the frequent itemsets using lift as the primary metric:

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=0.001)
```

Total Transactions: 14,963

Rule Structure

Each generated rule contains:

- ***Antecedents:*** Items that, when purchased, predict the consequent
- ***Consequents:*** Items predicted to be purchased given the antecedents
- ***Antecedent Support:*** Proportion of transactions containing the antecedent
- ***Consequent Support:*** Proportion of transactions containing the consequent
- ***Support:*** Proportion of transactions containing both antecedent and consequent
- ***Confidence:*** Probability of consequent given antecedent
- ***Lift:*** Strength of association controlling for item popularity

6.8. Database Integration

New Database Creation

A new database was created containing all original tables plus the association rules:

```
conn_new = sqlite3.connect('product_apriori.db')
```

Item ID Mapping

To enable integration with the sentiment analysis component, item names in the association rules were mapped to their corresponding item_id values from the transactions table:

```
def map_items_to_ids(items_frozenset):
    if isinstance(items_frozenset, frozenset):
        item_list = list(items_frozenset)
        ids = [item_to_id.get(item) for item in item_list if item in item_to_id]
        return ', '.join(map(str, ids)) if ids else None
    return None
```

Market Basket Table Schema

The final *market_basket* table includes:

- ***antecedents -*** Item names in the rule's left-hand side
- ***consequents -*** Item names in the rule's right-hand side
- ***antecedents_id -*** Mapped item IDs for antecedents

- ***consequents_id* -** Mapped item IDs for consequents
- ***antecedent support* -** Support value for antecedents
- ***consequent support* -** Support value for consequents
- ***support* -** Combined support value
- ***confidence* -** Rule confidence
- ***lift* -** Rule lift value

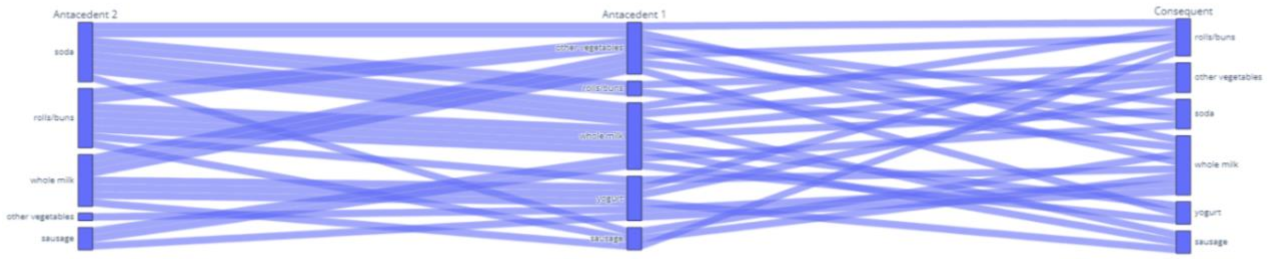
6.9. Visualization with PyARMViz

Rule Visualization Preparation

The association rules were prepared for visualization using the PyARMViz library:

```
rules_dict = []
for rule in rules_data:
    diction = {
        'lhs': tuple(antecedents),
        'rhs': tuple(consequents),
        'count_full': combined_count,
        'count_lhs': antecedent_count,
        'count_rhs': consequent_count,
        'num_transactions': 14963
    }
    rules_dict.append(diction)
```

The generate_rule_from_dict function from PyARMViz creates rule objects suitable for network visualization, enabling interactive exploration of product associations.



6.10. Key Results Summary

Metric	**Value**
Total Transactions Analyzed	14,963
Unique Product Categories	167
Minimum Support Threshold	0.001 (0.1%)
Maximum Itemset Length	5
Lift Threshold	0.001

6.11. Integration with Sentiment Analysis

The item_id mapping created in this notebook enables seamless connection between association rules and sentiment data from Part 3.

For each discovered rule:

- Antecedent and consequent items are linked to their respective sentiment scores via *item_id*
- Recommendations can prioritize not just frequently co-purchased items, but those with positive customer sentiment
- This creates a comprehensive system that combines transactional behavior patterns with qualitative consumer opinions

6.12. Limitations and Future Enhancements

Current Limitations:

- Low support threshold may generate some spurious associations
- Temporal patterns (seasonal purchasing behavior) not incorporated
- Customer segmentation not applied (rules represent aggregate behavior)

Planned Enhancements:

- Filter rules by higher confidence thresholds for production use
- Incorporate time-series analysis to detect seasonal rule variations
- Apply customer clustering to generate segment-specific recommendations
- Integrate lift filtering to focus on rules with strong positive associations (lift > 1)

7. Sentiment Analysis using DistilBERT

7.1. Overview and Objectives

This notebook performs sentiment analysis on customer reviews using a pre-trained DistilBERT transformer model. The analysis classifies customer feedback as Positive, Negative, or Neutral, enabling the recommendation system to suggest products with favorable customer sentiment alongside the market basket association rules.

7.2. Data Extraction

Database Connection

```
conn = sqlite3.connect('products.db')
```

The notebook connects to the *products.db* database, which contains 6 tables: customer, product, product_details, ratings, reviews, and transactions.

Reviews Data Extraction

```
reviews_df = pd.read_sql_query(
    "SELECT text, title, rating, user_id, parent_asin FROM reviews",
    conn)
```

Extracted Columns:

- ***text* -** Full review text content
- ***title* -** Review title/headline
- ***rating* -** Star rating (1-5)
- ***user_id* -** Reviewer identifier
- ***parent_asin* -** Product identifier for joining with other tables

7.3. Handling Missing Values

Real-world data quality checks were performed:

```
# Check for null values
reviews_df.isnull().sum()

# Check for empty strings
empty_count = (reviews_df['text'] == '').sum()

# Check for whitespace-only entries
whitespace_count = (reviews_df['text'].str.strip() == '').sum()
```

Decision: Rows with missing text were retained since sentiment analysis uses both text and title fields combined, preventing data loss.

7.4. Text Preprocessing

Cleaning Function

A comprehensive text cleaning function was implemented to normalize review text:

```
def clean_text(text):
    # Convert to lowercase
    text = text.lower()

    # Replace emoticons with sentiment words
    emoticon_dict = {
        r':\)': ' positive ', r':-\)': ' positive ', r':D': ' positive ',
        r':\(': ' negative ', r':-\(': ' negative ', r'T_T': ' negative ',
        r':/': ' skeptical ', r':\|': ' neutral ',
    }

    # Replace emojis with sentiment words
    emoji_dict = {
        '😊': ' positive ', '😍': ' positive ', '♡': ' positive ', '👍': ' 
positive ',
        '😢': ' negative ', '😠': ' negative ', '👎': ' negative ',
    }

    # Remove special characters (keep only letters and spaces)
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    return text
```

Text Combination

Both title and text were cleaned and combined for comprehensive analysis:

```
reviews_df['cleaned_text'] = reviews_df['text'].apply(clean_text)
reviews_df['cleaned_title'] = reviews_df['title'].apply(clean_text)
reviews_df['combined_text'] = reviews_df['cleaned_text']
```

7.5. Sentiment Analysis with DistilBERT

Model Loading

```
from transformers import pipeline

sentiment_pipeline = pipeline(
    "sentiment-analysis",
    model="distilbert-base-uncased-finetuned-sst-2-english"
)
```

Model Details:

- **Model:** DistilBERT (distilled version of BERT)
- **Fine-tuned on:** SST-2 (Stanford Sentiment Treebank)
- **Output:** POSITIVE or NEGATIVE label with confidence score (0.0-1.0)

Sentiment Scoring Function

```
def get_bert_sentiment(text):
    if not text or len(text.strip()) == 0:
        return 0

    # Truncate to BERT's max token limit
    text_truncated = text[:512]

    # Get prediction
    result = sentiment_pipeline(text_truncated)[0]
    label = result['label']
    score = result['score']

    # Convert to -1 to +1 scale
    if label == 'POSITIVE':
        return round(score, 3)
    else:
        return round(-score, 3)
```

Score Interpretation:

- **+1.0:** Strongly positive sentiment
- **0.0:** Neutral sentiment
- **-1.0:** Strongly negative sentiment

Sentiment Classification

```
def classify_bert_sentiment(score):
    if score > 0.05:
        return 'Positive'
    elif score < -0.05:
        return 'Negative'
    else:
        return 'Neutral'
```

7.6. Results Analysis

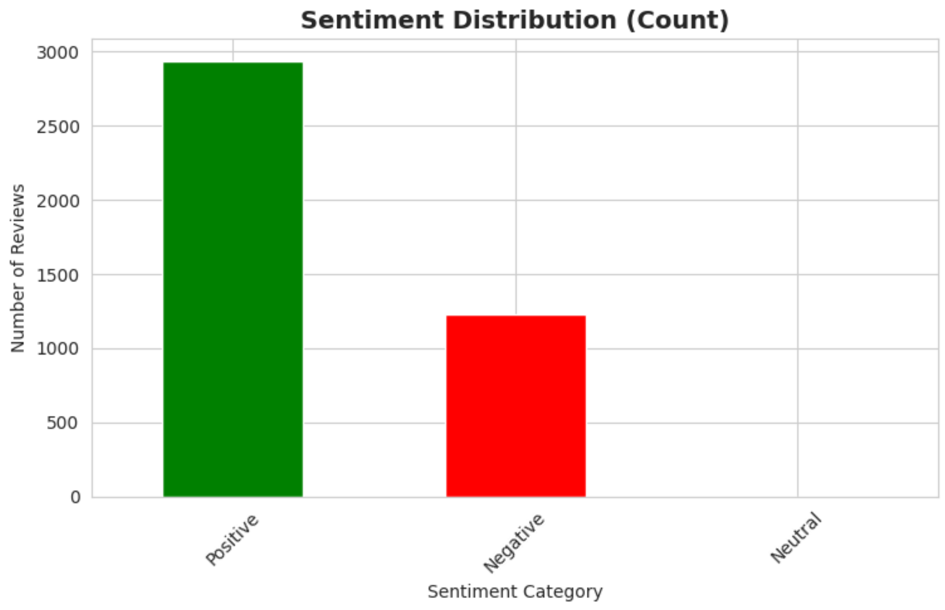
Sentiment Distribution

The analysis produces sentiment counts and percentages across all reviews:

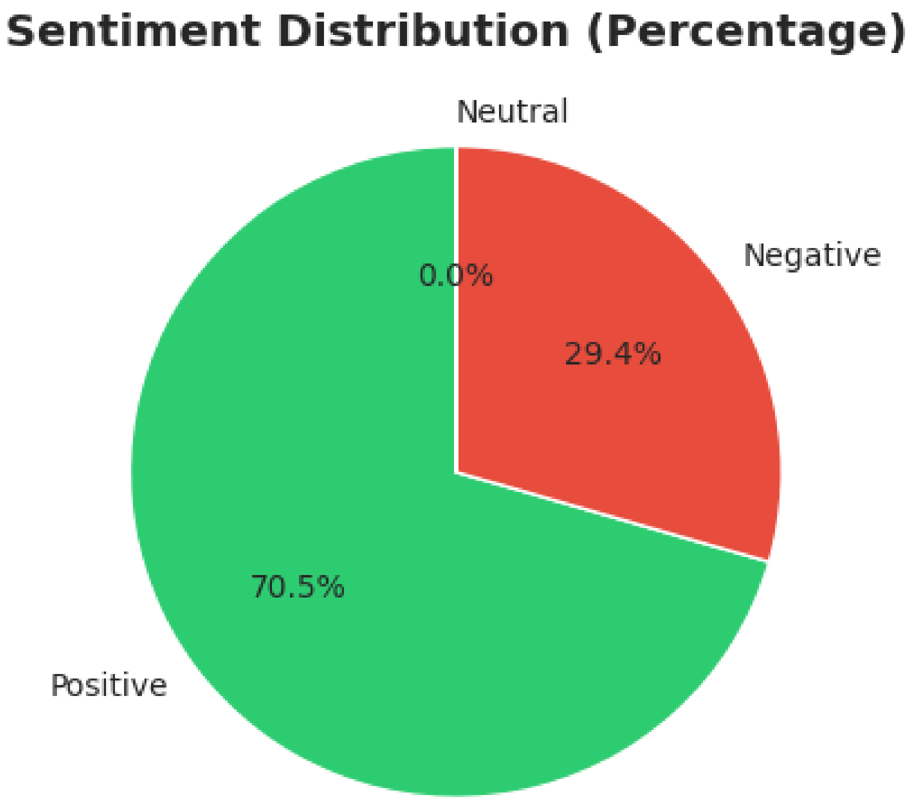
Category	Description
Positive	Score > 0.05
Neutral	-0.05 ≤ Score ≤ 0.05
Negative	Score < -0.05

Visualizations Created

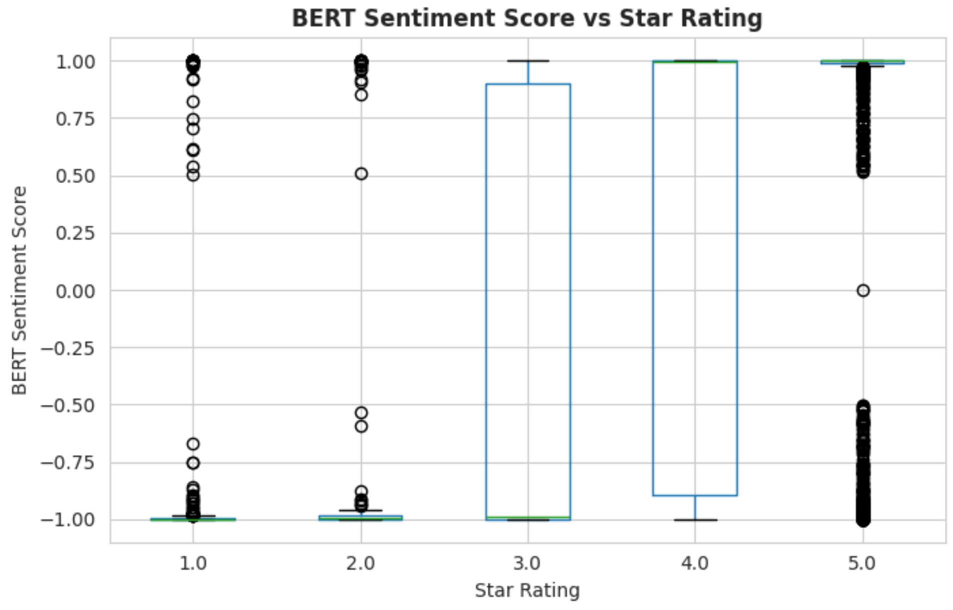
- **Bar Chart:** Sentiment distribution by count



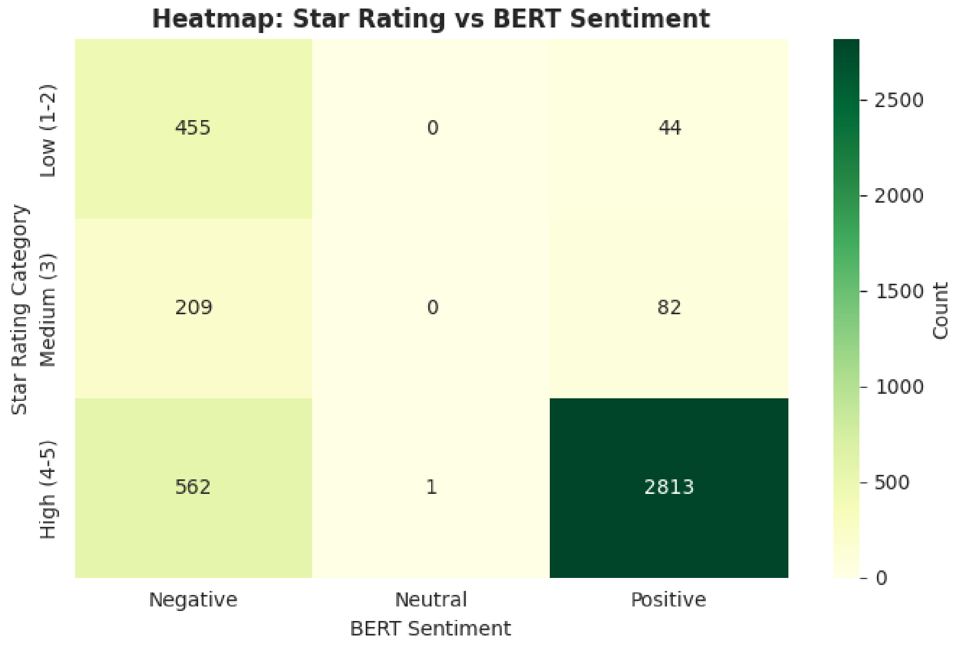
- **Pie Chart:** Sentiment distribution by percentage



- **Box Plot:** BERT sentiment score vs star rating



- **Heatmap:** Cross-tabulation of star rating categories vs sentiment categories



Rating vs Sentiment Comparison
Cross-tabulation analysis compares star ratings with sentiment predictions:

```
reviews_df['rating_category'] = pd.cut(reviews_df['rating'],
                                        bins=[0, 2, 3, 5],
                                        labels=['Low (1-2)', 'Medium (3)', 'High
(4-5)'])

cross_tab = pd.crosstab(reviews_df['rating_category'],
                        reviews_df['bert_sentiment_category'])
```

This reveals whether customers' written words match their star ratings.

Cross-tabulation: Star Rating vs BERT Sentiment

Rating Category	Negative	Neutral	Positive	All
Low (1-2)	455	0	44	499
Medium (3)	209	0	82	291
High (4-5)	562	1	2813	3376
All	1226	1	2939	4166

Average BERT Sentiment Score by Star Rating:

Rating	Mean	STD	Count
1.0	-0.865770	0.471545	344
2.0	-0.721948	0.668663	155
3.0	-0.431237	0.871944	291
4.0	0.367932	0.902085	470
5.0	0.715876	0.670391	2906

7.7. Product-Level Sentiment Aggregation

Grouping by Product

```
product_sentiment = reviews_df.groupby('parent_asin').agg({
    'bert_sentiment_score': ['mean', 'std', 'min', 'max', 'count'],
    'rating': 'mean',
    'bert_sentiment_category': lambda x: (x == 'Positive').sum()
})
```

Aggregated Metrics per Product:

- **avg_sentiment -** Mean sentiment score
- **std_sentiment -** Standard deviation
- **min_sentiment / max_sentiment -** Score range
- **review_count -** Number of reviews
- **avg_rating -** Mean star rating
- **positive_count -** Count of positive reviews

7.8. Product Recommendations Integration

Data Merging Pipeline

The sentiment scores are merged with product information:

```
# Step 1: Map parent_asin to item_id via ratings table
asin_to_item = pd.read_sql_query(
    "SELECT DISTINCT parent_asin, item_id FROM ratings", conn
)

# Step 2: Add category from product table
product_table = pd.read_sql_query(
    "SELECT item_id, category FROM product", conn
)

# Step 3: Add product details (title, price)
product_details = pd.read_sql_query(
    "SELECT item_id, title, price FROM product_details", conn
)
```

Quality Filtering

```
product_sentiment = reviews_df.groupby('parent_asin').agg({
    'bert_sentiment_score': ['mean', 'std', 'min', 'max', 'count'],
    'rating': 'mean',
    'bert_sentiment_category': lambda x: (x == 'Positive').sum()
})MIN_REVIEWS = 5

filtered = product_sentiment_with_category[
    product_sentiment_with_category['review_count'] >= MIN_REVIEWS
]
```

Products with fewer than 5 reviews are excluded to avoid single-review bias.

Top 3 Recommendations per Item Category

```
top_3_final = (df_filtered
    .sort_values(['item', 'avg_sentiment'], ascending=[True, False])
    .groupby('item')
    .head(3)
    .reset_index(drop=True))
```

7.9. Database Export

Table Creation in product_apriori.db

```
tableau_db = sqlite3.connect('product_apriori.db')
create_table_sql = """
CREATE TABLE IF NOT EXISTS top_products (
    rank INTEGER,
    category TEXT,
    item TEXT,
    title TEXT,
    price REAL,
    avg_sentiment REAL,
    review_count REAL,
    average_rating REAL,
    item_id INTEGER,
    parent_asin TEXT,
    PRIMARY KEY (rank, item, parent_asin)
)"""
```

Final Export Schema

Column	Type	Description
rank	INTEGER	Rank within item category (1-3)
category	TEXT	Product category
item	TEXT	Item name
title	TEXT	Full product title
price	REAL	Product price
avg_sentiment	REAL	Average sentiment score (-1 to +1)
review_count	REAL	Number of reviews analyzed
average_rating	REAL	Average star rating
item_id	INTEGER	Foreign key to product table
parent_asin	TEXT	Amazon product identifier

7.10. Key Outputs

- *top_products* table in *product_apriori.db* - Top 3 products per item category ranked by sentiment
- Sentiment scores for all reviews with category labels
- Visualizations showing sentiment distribution and rating correlations
- Product recommendations ready for Tableau dashboard integration

7.11. Sample Predictions

Review Text	Score	Classification
"This product is amazing! Love it!"	1	Positive
"Terrible quality, waste of money"	-1	Negative
"It's okay, nothing special"	-0.808	Negative
"The candy was way too soft and salty"	-0.997	Negative
"I was never a huge fan until I found this one"	0.919	Positive

8. Tools and Technologies

Component	Technology	Purpose
Data Analysis	Python (pandas, numpy)	Data manipulation and analysis
Machine Learning	scikit-learn, mlxtend	Apriori algorithm, evaluation metrics
NLP/Sentiment	HuggingFace Transformers	Deep learning sentiment classification
Database	SQLite, SQL	Data storage and querying
Visualization	Streamlit	Interactive dashboard development
Version Control	GitHub	Collaboration and code management

9. Limitations

- **Dataset alignment issues:** Significant mismatch between review parent_asin values and ratings table reduced linkable records
- **Dataset age:** Amazon Fine Food Reviews dataset from 2012 may not reflect current consumer sentiment patterns
- **Market basket analysis scope:** Association rules generated from single grocery outlet may not generalize to broader retail contexts
- **BERT token limitation:** 512-token maximum truncates longer reviews, potentially losing sentiment-relevant content

10. Future Work and Recommendations

10.1. Immediate Improvements

- **Dataset Enhancement:** Source more recent review datasets to improve sentiment model relevance
- **Data Integration Refinement:** Develop more robust product-review linking mechanisms to reduce data loss
- **Temporal Analysis:** Incorporate time-series analysis to identify seasonal purchasing patterns

10.2. Advanced Extensions

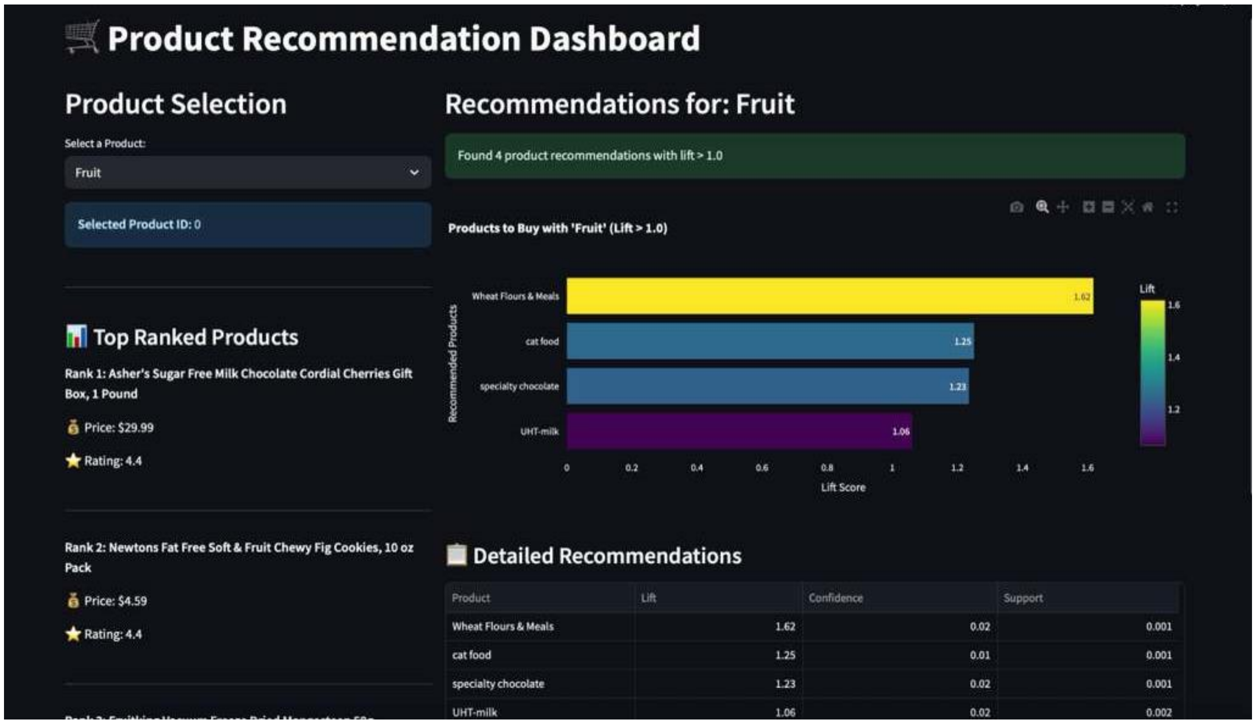
- **Customer Segmentation:** Implement age-based and demographic segmentation for personalized recommendations
- **Advanced NLP Techniques:** Explore aspect-based sentiment analysis to identify specific product attributes driving satisfaction
- **A/B Testing Framework:** Implement controlled experiments to measure recommendation effectiveness

10.3. Dashboard Enhancements

- **Network visualization:** Interactive product association graphs
- **Sentiment heatmaps:** Category-level sentiment visualization
- **Recommendation panels:** Dynamic brand suggestions filtered by sentiment scores
- **SQL connectivity:** Real-time dashboard updates from production database

11. Product Recommendation Dashboard

The ***Product Recommendation Dashboard*** integrates market basket analysis and sentiment analysis results in a unified interface for non-technical stakeholders.



🛒 Product Recommendation Dashboard

Product Selection

Select a Product:

Fruit

Selected Product ID: 0

📊 Top Ranked Products

Rank 1: Asher's Sugar Free Milk Chocolate Cordial Cherries Gift Box, 1 Pound

💰 Price: $29.99

⭐ Rating: 4.4

Rank 2: Newtons Fat Free Soft & Fruit Chewy Fig Cookies, 10 oz Pack

💰 Price: $4.59

⭐ Rating: 4.4

Recommendations for: Fruit

Found 4 product recommendations with lift > 1.0

📋 Detailed Recommendations

Product	Lift	Confidence	Support
Wheat Flours & Meals	1.62	0.02	0.001
cat food	1.25	0.01	0.001
specialty chocolate	1.23	0.02	0.001
UHT-milk	1.06	0.02	0.002

- **Product Selection Panel:** Users select a product category from a dropdown menu to view recommendations.
- **Top Ranked Products:** Displays the top 3 products within the selected category ranked by sentiment score, including price and average rating.
- **Association Rule Visualization:** A horizontal bar chart displays products frequently co-purchased with the selected item, filtered to show only positive associations (lift > 1.0). Bar length and color intensity represent lift strength.
- **Detailed Recommendations Table:** Presents complete metrics for each associated product including lift, confidence, and support values.

Example Output: Selecting *"Fruit"* returns 4 associated products, with Wheat Flours & Meals showing the strongest association (lift = 1.62), indicating customers purchasing fruit are 62% more likely to also purchase flour products. The sentiment panel simultaneously recommends Asher's Sugar Free Chocolate Cherries as the highest-rated fruit-category product (Rating: 4.4).

Value Proposition: The dashboard answers two questions simultaneously: "What products are frequently bought together?" (association rules) and "Which specific products have the best reviews?" (sentiment rankings)—delivering quality-filtered recommendations that address the limitations of traditional systems.

SQL Queries

Query 1: Most Frequently Reviewed & Highly Rated Products

Purpose: Identify products that attract the most reviews while maintaining high ratings.

```
WITH title_counts AS (
    SELECT
        r.parent_asin,
        pd.title,
        COUNT(*) AS title_count
    FROM ratings r
    JOIN product_details pd
        ON pd.item_id = r.item_id
    GROUP BY r.parent_asin, pd.title  ),
best_title AS (
    SELECT parent_asin, title
    FROM (
        SELECT
            parent_asin,
            title,
            title_count,
            ROW_NUMBER() OVER (PARTITION BY parent_asin ORDER BY title_count
DESC) AS rn
        FROM title_counts      )
    WHERE rn = 1   )
SELECT
    r.parent_asin,
    bt.title,
    COUNT(rv.rating) AS review_count,
    r.average_rating,
    r.rating_number
FROM ratings r
JOIN reviews rv
    ON rv.parent_asin = r.parent_asin
JOIN best_title bt
    ON bt.parent_asin = r.parent_asin
GROUP BY
    r.parent_asin, bt.title, r.average_rating, r.rating_number
ORDER BY
    review_count DESC,
    average_rating DESC
LIMIT 5
```

Output:

parent_asin	title	Review_count	Average_rating	Rating_number
B01FW4ZCJM	Hershey's Reese's Crunchy Cookie Peanut butter/Chocolate Candy 1.4 oz.	4	4.6	2104
B0C7RJ16K2	TORTUGA Caribbean Coconut Rum Cake - 32 oz Rum Cake - The Perfect Premium Gourmet Gift for Gift Baskets, Parties, Holidays, and Birthdays	3	4.5	1180
B09HCFRWNP	Dark Roast Pure Coffee	3	4.5	18868
B09KC69RSH	Iberia Organic Cane Sugar, 1.5 lb. (Pack of 3)	2	4.7	112773
B0091YJHUU	Dark Roast Pure Coffee	2	4.7	7794

Query 2: Highest-Rated Products with Minimum Ratings Threshold

Purpose: Find top-rated products with sufficient review volume to ensure reliability.

```sql
SELECT
    r.parent_asin,
    pd.title,
    r.average_rating,
    r.rating_number
FROM ratings r
JOIN product_details pd
    ON pd.item_id = r.item_id
WHERE r.rating_number >= 20
ORDER BY
    r.average_rating DESC,
    r.rating_number DESC
LIMIT 10;
```

Output:

parent_asin	title	Average_rating	rating_number
B07HKWV1R2	Cherry Sours Chewy Candy Balls - 3 lbs of Tart Fresh Delicious Bulk Candy	5	20
B007SAUB36	Del Monte Garlic Pasta Sauce, 24 Ounce (Pack of 12)	5	20
B0748MWKNM	Handlmaier's Sweet Bavarian Mustard-8 oz - 230 g-IMPORTED- Shipping from USA	4.9	274
B07PGK25N5	Handlmaier's Sweet Bavarian Mustard-8 oz - 230 g-IMPORTED- Shipping from USA	4.9	102
B0C15BSMVB	HERB OX Beef Instant Broth & Seasoning, Sodium Free, 8 Count (Pack of 12)	4.9	89
B01M15VPV9	Dark Roast Pure Coffee	4.9	88
B01MYGNRQF	Eden Organic Gomasio, Sesame Seeds and Sea Salt, Umami, Traditional, Furikake, Seasoning, Macrobiotic, 3.5 oz (2-Pack)	4.9	48
B01ID2T660	Eden Organic Gomasio, Sesame Seeds and Sea Salt, Umami, Traditional, Furikake, Seasoning, Macrobiotic, 3.5 oz (2-Pack)	4.9	35
B00LZHU7D2	Grow A Pear Handcrafted Prickly Pear Syrup 9 oz	4.9	24
B0BKN985C5	Sujata Chakki Atta, Whole Wheat Flour, 10-Pound Bag	4.9	24

Key Finding: Condiments, seasonings, and pantry staples dominate with near-perfect ratings.

Query 3: Average Sentiment Score per Product Category

Purpose: Summarize sentiment & ratings by category to identify strongest/weakest categories.

```
SELECT category,
       AVG(avg_sentiment) AS avg_sentiment,
       AVG(average_rating) AS avg_rating,
       SUM(review_count) AS total_reviews,
       COUNT(*) AS num_products
FROM top_products
GROUP BY category
ORDER BY total_reviews DESC
```

Output:

category	avg_ sentiment	avg_ rating	total_ reviews	num_ products
Cookies	0.428	4.42	42	15
Nut & Seed Butters	0.498	4.6	36	9
Cereals	1	4.6	30	15
Soups, Stocks & Broths	1	4.5	30	15
Chocolate	0.601	4.16	27	15
Jams, Jellies & Sweet Spreads	0.832	4.47	21	9
Candy & Chocolate	1	4.27	21	18
Baking Chips	0.749	4.35	18	6
Dairy, Eggs & Plant-Based Alternatives	0.749	4.65	15	6
Salt & Salt Substitutes	0.573	4.4	15	15
Cakes	0.688	4.6	12	6
Coffee	1	4.7	12	12
Food & Beverage Gifts	0.999	4.53	12	9
White Sugars	-0.854	4.7	12	6
Baking Syrups, Sugars & Sweeteners	-0.854	4.7	12	6
Cooking & Baking	0.073	4.7	9	6
Candy & Chocolate Bars	1	4.2	9	6
Breakfast Foods	1	4.6	6	3
Crackers	0.786	4.5	6	6
Beverages	1	4.7	6	6

Query 4: Monthly Trends in Product Ratings

Purpose: Track how ratings and review volume change over time.

```
SELECT date_format(from_unixtime(timestamp / 1000), 'yyyy-MM') AS year_month,
       COUNT(*) AS review_count,
       AVG(rating) AS avg_rating
FROM reviews
GROUP BY year_month
ORDER BY year_month
```

Output:

year_month	review_count	avg_rating
2006-07	1	5
2007-08	1	5
2009-05	1	4
2009-11	1	5
2010-09	1	5
2010-11	1	5
2011-01	2	5
2011-08	1	5
2011-09	1	3
2012-02	1	5
2012-08	1	3
2012-12	1	5
2013-01	3	3.67
2013-02	1	5
2013-04	4	4
2013-05	9	4.56
2013-06	3	5
2013-07	1	5
2013-08	2	5
2013-09	4	4.25

Query 5: Relationship Between Review Length and Rating

Purpose: Analyze whether longer reviews correlate with different sentiment.

```
WITH reviews_with_len AS (
    SELECT rating, LENGTH(text) AS review_len
    FROM reviews WHERE text IS NOT NULL
),
bucketed AS (
    SELECT CASE
        WHEN review_len < 100 THEN 'short (<100 chars)'
        WHEN review_len BETWEEN 100 AND 300 THEN 'medium (100-300 chars)'
        WHEN review_len BETWEEN 301 AND 600 THEN 'long (301-600 chars)'
        ELSE 'very long (>600 chars)'
    END AS length_bucket, rating
    FROM reviews_with_len
)
SELECT length_bucket, COUNT(*) AS num_reviews, AVG(rating) AS avg_rating
FROM bucketed
GROUP BY length_bucket
```

Insight: Shorter reviews tend to have slightly higher ratings; very long reviews tend to be slightly more negative.

Output:

length_bucket	num_reviews	avg_rating
short (<100 chars)	1849	4.38
medium (100–300 chars)	1511	4.25
long (301–600 chars)	551	4.29
very long (>600 chars)	255	4.16

Query 6: Top Keywords in Positive vs Negative Reviews

Purpose: Compare keyword frequency between positive and negative reviews.

```
WITH labelled_reviews AS (
    SELECT
        CASE
            WHEN rating >= 4 THEN 'positive'
            WHEN rating <= 2 THEN 'negative'
            ELSE 'neutral'
        END AS sentiment,
        text
    FROM reviews
    WHERE text IS NOT NULL),
keyword_counts AS (
    SELECT sentiment, 'delicious' AS keyword, COUNT(*) AS occurrences
    FROM labelled_reviews
    WHERE lower(text) LIKE '%delicious%'
    GROUP BY sentiment
    UNION ALL
    SELECT sentiment, 'tasty' AS keyword, COUNT(*) AS occurrences
    FROM labelled_reviews
    WHERE lower(text) LIKE '%tasty%'
    GROUP BY sentiment
    UNION ALL
    SELECT sentiment, 'bad' AS keyword, COUNT(*) AS occurrences
    FROM labelled_reviews
    WHERE lower(text) LIKE '%bad%'
    GROUP BY sentiment
    UNION ALL
    SELECT sentiment, 'disappointed' AS keyword, COUNT(*) AS occurrences
    FROM labelled_reviews
    WHERE lower(text) LIKE '%disappointed%'
    GROUP BY sentiment)
SELECT * FROM keyword_counts
ORDER BY keyword, sentiment
```

Output:

sentiment	keyword	occurrences
negative	bad	29
neutral	bad	11
positive	bad	49
negative	delicious	8
neutral	delicious	7
positive	delicious	446

negative	disappointed	48
neutral	disappointed	17
positive	disappointed	38
negative	tasty	9
neutral	tasty	8
positive	tasty	306

Query 7: Customer Segments by Sentiment Polarity

Purpose: Segment customers based on their overall review behavior.

```
WITH customer_stats AS (
    SELECT
        rv.user_id,
        COUNT(*)            AS num_reviews,
        AVG(rv.rating)      AS avg_rating
    FROM reviews rv
    GROUP BY rv.user_id
),
segmented AS (
    SELECT
        cs.user_id,
        cs.num_reviews,
        cs.avg_rating,
        CASE
            WHEN cs.avg_rating >= 4 THEN 'Positive'
            WHEN cs.avg_rating BETWEEN 2.5 AND 4 THEN 'Neutral'
            ELSE 'Negative'
        END AS sentiment_segment
    FROM customer_stats cs
)
SELECT
    s.sentiment_segment,
    COUNT(*)                    AS num_customers,
    AVG(s.num_reviews)          AS avg_reviews_per_customer
FROM segmented s
GROUP BY s.sentiment_segment
ORDER BY num_customers DESC
```

Insight: Positive customers are more engaged (write more reviews); negative customers tend to be less active.

Output:

sentiment_segment	num_customers	avg_reviews_per_customer
Positive	400	8.19
Neutral	115	6.97
Negative	37	2.38

Query 8: Sentiment Comparison Between Product Categories

Purpose: Compare categories on sentiment, rating, and engagement metrics.

```
SELECT
    category,
    AVG(avg_sentiment)   AS avg_sentiment,
    AVG(average_rating)  AS avg_rating,
    SUM(review_count)    AS total_reviews,
    COUNT(*)             AS num_products
FROM top_products
GROUP BY category
ORDER BY total_reviews DESC
```

Output:

category	avg_ sentiment	avg_ rating	total_ reviews	num_ products
Cookies	0.43	4.42	42	15
Nut & Seed Butters	0.5	4.6	36	9
Cereals	1	4.6	30	15
Soups, Stocks & Broths	1	4.5	30	15
Chocolate	0.6	4.16	27	15
Jams, Jellies & Sweet Spreads	0.83	4.47	21	9
Candy & Chocolate	1	4.27	21	18
Baking Chips	0.75	4.35	18	6
Dairy, Eggs & Plant-Based Alternatives	0.75	4.65	15	6
Salt & Salt Substitutes	0.57	4.4	15	15
Cakes	0.69	4.6	12	6
Coffee	1	4.7	12	12
Food & Beverage Gifts	1	4.53	12	9
White Sugars	-0.85	4.7	12	6
Baking Syrups, Sugars & Sweeteners	-0.85	4.7	12	6
Cooking & Baking	0.07	4.7	9	6
Candy & Chocolate Bars	1	4.2	9	6
Breakfast Foods	1	4.6	6	3
Crackers	0.79	4.5	6	6
Beverages	1	4.7	6	6