A MINI PROJECT REPORT

ON

PRODUCT RECOMMENDATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF
BACHELOR OF COMPUTER ENGINEERING

BY

21 AYUSH ROKADE
22 RAHUL SAHU
23 ANIKET SALVI
24 HARSHIKA SANGAM
25 SAHIL SARAWADE

UNDER GUIDANCE OF

Dr. B. B. MESHRAM

UNIVERSITY OF MUMBAI
DEPARTMENT OF COMPUTER ENGINEERING
DATTA MEGHE COLLEGE OF ENGINEERING
PLOT NO.98 SECTOR-3, AIROLI, NAVI MUMBAI
ACADAMIC YEAR 2025-26

DATTA MEGHE COLLEGE OF ENGINEERING

AIROLI, NAVI MUMBA

CERTIFICATE

This is to certify that the project entitled **PRODUCT RECOMMENDATION** is bonafide work of **Ayush Rokade, Rahul Sahu, Aniket Salvi, Harshika Sangam, Sahil Sarawade** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of **BE** in **"Computer Engineering"**.

Dr. B. B. Meshram	Dr. A. P. Pande	Dr. P. A. Dode
Project Guide	Head of the Department	Principal

DATTA MEGHE COLLEGE OF ENGINEERING

AIROLI, NAVI MUMBAI

MINI PROJECT APPROVAL

This project report entitled **PRODUCT RECOMMENDATION**
of the students **Ayush Rokade, Rahul Sahu, Aniket Salvi, Harshika Sangam, Sahil Sarawade**
approved for the degree of **Computer Engineering.**

Internal Examiner	External Examiner
Date:	Date:
Place:	Place:

ACKNOWLEDGEMENT

Motivation and guidance are the keys towards success. I would like to extend my thanks to all the sources of motivation.

We would like to grab this opportunity to thank **Dr. P. A. Dode, Principal** for encouragement and support he has given for our project.

We express our deep gratitude to **Dr. A. P. Pande, Head of the Department** who has been the constant driving force behind the completion of this project.

We wish to express our heartfelt appreciation and deep sense of gratitude to my project guide **Dr. B. B. Meshram** for his encouragement, invaluable support, timely help, lucid suggestions and excellent guidance which helped us to understand and achieve the project goal. His concrete directions and critical views have greatly helped us in successful completion of this work.

We extend our sincere appreciation to all Professors for their valuable inside and tip during the designing of the project. Their contributions have been valuable in so many ways that we find it difficult to acknowledge of them individually.

We are also thankful to all those who helped us directly or indirectly in completion of this work.

Place:

Date:

Name of the student

Ayush Rokade
Rahul Sahu
Aniket Salvi
Harshika Sangam
Sahil Sarawade

Product Recommendation System

Product Recommendation System is an intelligent application that predicts and suggests products to users based on their preferences, past purchases, or similarity to other products. The goal is to recommend items that users are most likely to be interested in by analyzing user behavior, transaction history, and product associations. Using Machine Learning techniques such as supervised and unsupervised learning and the Apriori algorithm for association rule mining, it helps users discover relevant products efficiently while enhancing the overall shopping experience.

1. Statement of the Problem

In today's e-commerce landscape, users face an overwhelming number of product choices across platforms like Amazon, Flipkart, and other online marketplaces. The vast selection often leads to decision fatigue, where users spend significant time browsing instead of finding the products they truly need. Traditional browsing or search methods are limited in addressing individual user preferences, resulting in repetitive suggestions or irrelevant recommendations.

Existing recommendation systems are often either too generic—relying mainly on popularity or trending items—or too opaque, using complex algorithms without transparent reasoning behind the suggestions. Moreover, many systems fail to dynamically adapt to changing user behavior or provide personalized insights based on real-time transaction patterns.

There is a clear need for an intelligent, interactive, and efficient Product Recommendation System that can generate personalized suggestions based on user behavior and product associations. While some tools provide basic recommendations, few integrate advanced Machine Learning techniques such as association rule mining to identify hidden patterns and meaningful product relationships.

Therefore, this project aims to develop a Product Recommendation System using Machine Learning, specifically leveraging supervised and unsupervised techniques and the Apriori algorithm. The system analyzes historical transaction data, extracts patterns, and generates relevant product recommendations. By doing so, it ensures users receive smarter, personalized suggestions, improving user satisfaction and driving higher engagement and sales in the e-commerce environment.

AIM:

1.1 Statement of the Problem

In today's rapidly growing e-commerce industry, users face significant challenges in discovering products that match their preferences from the vast and continuously expanding catalog of items available on platforms such as Amazon, Flipkart, and other online marketplaces. The overwhelming number of options often leads to decision fatigue, where customers spend more time browsing than actually purchasing products. Traditional browsing and filtering methods based on categories, popularity, or price fail to capture individual preferences, resulting in irrelevant or repetitive recommendations.

Current product recommendation systems are often either too generic—providing trending or best-selling items that may not align with a user's unique tastes—or too complex, requiring technical understanding to interpret how suggestions are generated. Moreover, many existing tools lack personalized insights based on purchase history, user behavior, or item associations, which are crucial for increasing engagement and sales.

There is also a noticeable gap in systems that provide accurate, personalized, and actionable product recommendations. While some engines suggest similar products, few combine effective association rule mining, user-friendly interfaces, and fast backend processing to deliver relevant recommendations seamlessly.

Therefore, there is a growing need for an intelligent and interactive Product Recommendation System that can analyze user behavior, identify product associations, and present personalized suggestions efficiently. Such a system would simplify the product discovery process, improve user satisfaction, and help users find items they are most likely to purchase.

1.2 Aim

To design and develop an intelligent Product Recommendation System that leverages Machine Learning techniques, including association rule mining and similarity-based analysis, to deliver personalized product suggestions efficiently through an interactive interface using Python, Pandas, and ML algorithms like Apriori.

1.3 Scope and Objectives

Scope 1: Intelligent Product Recommendation

- To provide personalized product suggestions based on user transaction history, preferences, and product associations.
- To recommend five to ten products that are most relevant to the user's previous purchases or selected items.
- To utilize preprocessed association rules for efficient and fast recommendation generation.
- To continuously adapt recommendations as user behavior and transaction data evolve over time.

Scope 2: Interactive User Interface

- To develop a simple, clean, and interactive web interface using Python (or Streamlit) for a smooth user experience.
- To allow users to easily select products or browse recommendations and instantly receive suggestions.
- To organize recommended products in a visually structured layout with images, names, and key details.

- To ensure the system runs efficiently without requiring complex database setups or heavy computational resources.

Scope 3: Data Handling and Integration

- To load and manage transaction and product data efficiently using Pandas for smooth data operations.
- To handle missing product information or images gracefully, maintaining consistency in the display.
- To integrate association rule mining through the Apriori algorithm to identify hidden product relationships.
- To use optimized backend operations for fast recommendation generation and improved performance.

Scope 4: Performance and Usability

- To ensure fast recommendation generation using preprocessed rules and similarity metrics.
- To minimize user waiting time by optimizing algorithms and data processing.
- To maintain an intuitive and responsive design suitable for all users, including non-technical audiences.
- To focus on e-commerce products across selected categories to maintain relevance and reliability in recommendations.

1.3 Proposed Architecture

The proposed architecture of the Product Recommendation System is designed to deliver fast, accurate, and personalized product suggestions through a simple web-based interface. The system follows a machine learning-driven recommendation approach, where products are recommended based on user behavior, transaction history, and association rules extracted from historical data.

The architecture is divided into several key components, each responsible for a specific functionality:

1. User Interface Layer

- Developed using Python (or Streamlit) to provide an interactive and user-friendly front-end.
- Allows users to select products, view recommended items, or explore suggested associations instantly.
- Displays the top 5–10 recommended products with their images, names, and key details.
- Uses responsive design elements such as columns, tables, and styled text for a clear visual layout.

2. Data Storage and Loading Layer

- Transaction and product datasets are preprocessed and stored in serialized Pickle (.pkl) files (e.g., products.pkl).
- Precomputed association rules (e.g., Apriori results) are stored in a separate serialized file (association_rules.pkl) for efficient recommendation generation.
- Pandas DataFrame is used to manage and access product and transaction data efficiently.

3. Recommendation Engine

- The system identifies relevant products for the user based on their selected item or transaction history.
- Using the precomputed association rules and similarity metrics, it retrieves the most strongly associated products.

- The top 5–10 products with the highest relevance scores are recommended.
- Sorting and indexing are handled using Python's built-in functions for optimized performance.

4. External Data Integration Layer

- Product images, descriptions, and details are fetched dynamically from available APIs or internal datasets.
- Parallel requests (e.g., using ThreadPoolExecutor) can be implemented to fetch multiple product details concurrently, improving response time and efficiency.
- Missing or unavailable product images are replaced with a placeholder to maintain layout consistency.

5. Output Display Layer

- The final output consists of recommended products displayed side-by-side with their images, names, and relevant details.
- Each recommendation card includes:
    - Product image (fetched dynamically or from the dataset)
    - Product name and key attributes (e.g., category, price)
- The design ensures an engaging and organized presentation of recommendations suitable for a shopping interface.

6. Workflow Summary

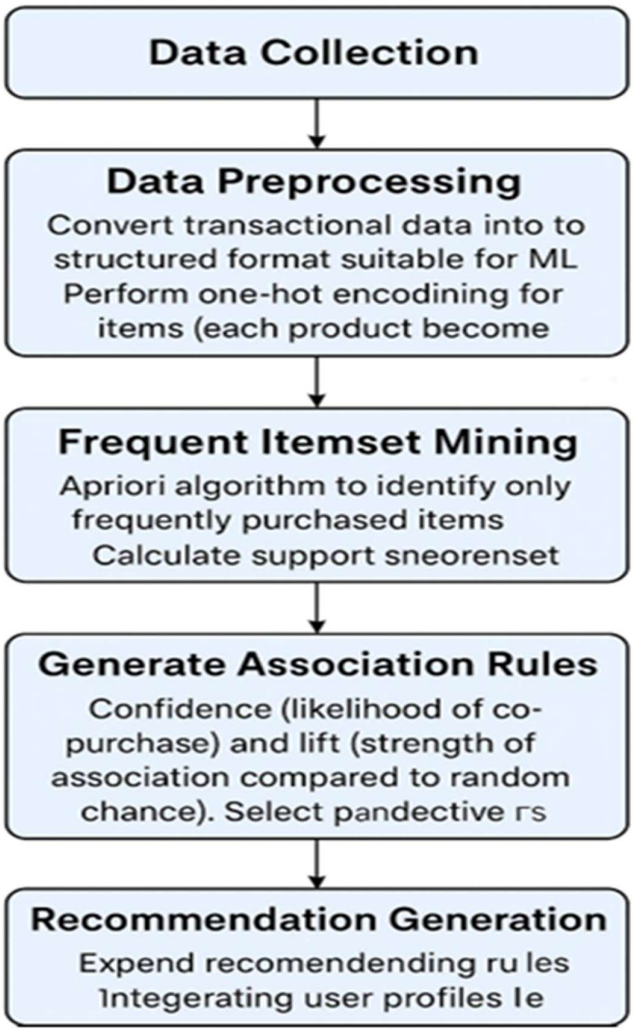
1. The user selects a product from a dropdown list or based on past transactions.
2. The system retrieves relevant association rules or similarity scores from the precomputed data.
3. The top 5–10 associated products are identified.
4. Product images and details are fetched in parallel for efficiency.
5. The recommendations are displayed on the web interface with images, names, and key information.

7. Key Features of the Architecture

- Lightweight and easy to deploy using Python or Streamlit.
- Fast response time due to preprocessed association rules and parallel data fetching.
- No complex database dependencies (all critical data handled through Pickle files).
- Clear separation between frontend (user interface), backend logic (Python + Pandas + ML), and external data sources.
- Scalable and easily extendable for more advanced ML models or collaborative filtering in the future.

2. THEORY : LITERATUTRE SURVEY

System Architecture:



- **Data Collection**

  Collect raw transactional records from sources such as point-of-sale systems, e-commerce order logs, or clickstream data.
  Each transaction should record which items were purchased together and include optional metadata (user ID, timestamp, store/location).
  Aim for clean, representative history covering enough time to capture seasonal and habitual patterns.
  Ensure privacy and permissions are handled (anonymize PII, obey consent and retention policies).

- **Data Preprocessing**

  Transform transactions into a machine-friendly format (one transaction = one row, items as features).
  Perform one-hot / basket encoding so each product becomes a binary column indicating presence/absence.
  Clean the data: remove duplicates, filter out extremely rare items, handle missing or noisy entries.
  Optionally enrich data with item categories, prices, or user demographics for more targeted rules.

- **Frequent Itemset Mining**

  Run an algorithm (e.g., Apriori or FP-Growth) to find item combinations that occur together above a minimum support threshold.
  Support measures how often a given itemset appears across all transactions and is used to prune infrequent sets.
  The output is a list of frequent itemsets (pairs, triples, ...) that are candidates for association rules.
  Choose support carefully: too high misses interesting patterns, too low yields many spurious itemsets.

- **Generate Association Rules**

  From each frequent itemset, generate rules of the form $A \rightarrow B$ and compute metrics like confidence and lift.
  Confidence estimates the probability of B being purchased when A is purchased (strength of implication).
  Lift compares the rule's co-occurrence to what would be expected by chance (lift > 1 indicates positive association).
  Select predictive, actionable rules by filtering on confidence, lift, and business constraints (e.g., profitability, seasonality).

- **Recommendation Generation**
  Turn selected association rules into recommendations (e.g., "customers who bought A also bought B") at the point of sale or online.

Rank and expand rules using business logic: combine multiple rules, personalize by user profile or past behavior, and account for inventory.
Apply filtering (exclude out-of-stock, avoid obvious duplicates) and present recommendations in UI slots (upsell, cross-sell, bundle).
Continuously monitor performance (conversion, lift in sales) and retrain rules periodically to capture changing patterns.

Tech Stack Suggestions

Layer / Component	**Tools / Libraries Used**
Frontend / UI	Streamlit (dropdowns, buttons, product cards, images)
Backend / App Logic	Python + Streamlit (handles user interaction & recommendation functions)
Data Source	product dataset (taken from kaggle)
ML / Recommendation	Python ML libraries (Apriori algorithm, scikit-learn for similarity), Pandas (data manipulation)
Data Storage	Pickle files (products.pkl, association_rules.pkl)
Visualization	Streamlit components (columns, tables, product images, text)

SCOPE OF MIS FOR PRODUCT RECOMMENDATION SYSTEM

1. **Personalized Decision Support**

- Helps users make better purchasing decisions by recommending products related to their selected items or past purchases.
- Provides insights based on historical transaction data and product association patterns.

2. **Efficient Data Management**

- Organizes product metadata (names, categories, images, prices) using Pandas and Pickle files.
- Stores and retrieves precomputed association rules efficiently to enhance recommendation performance.

3. **Enhanced User Interaction**

- Provides a friendly interface with Streamlit for selecting products and viewing recommended items.
- Displays product images fetched from internal datasets or e-commerce APIs for visual guidance.

4. **Automated Recommendation System**

- Automatically generates top 5–10 product recommendations using precomputed association rules or similarity metrics.
- Uses ThreadPoolExecutor to fetch multiple product images or details concurrently, improving response time.

5. **Performance Monitoring and Optimization**

- Preprocessed association rules ensure fast recommendation retrieval.
- Handles multiple user requests efficiently, ensuring smooth performance.

6. **Data-driven Insights for Users**

- Users can explore product relationships (e.g., frequently bought together, similar categories) without manual searching.
- Enables informed shopping decisions based on patterns in historical data rather than guesswork.

7. **Scalability and Integration**

- Can be expanded to include collaborative filtering, user ratings, or hybrid recommendation models in the future.

- Potential integration with additional e-commerce APIs or databases for advanced features.

Scope 1: Personalized Product Recommendation
Handles the core recommendation logic using precomputed association rule

```
# recommend products
def recommend_products(selected_product):
    # Retrieve associated products from precomputed rules
    recommended_products = association_rules.get(selected_product, [])[:5]

    with ThreadPoolExecutor(max_workers=5) as executor:
        recommended_images = list(executor.map(fetch_product_image, recommended_products))

    return recommended_products, recommended_images
```

Scope 2: Data Management and Storage
Handles loading and storing product and transaction data.

```
# load product and association data
products_dict = pickle.load(open('products.pkl', 'rb'))
products = pd.DataFrame(products_dict)
association_rules = pickle.load(open('association_rules.pkl', 'rb'))
```

Scope 3: External Data Integration (Product Images)
Fetches product images from internal datasets or e-commerce APIs.

```
# fetch product images
def fetch_product_image(product_name):
    try:
        # Example: fetch from internal dataset or API
        image_url = products.loc[products['name'] == product_name, 'image'].values[0]
        return image_url if image_url else None
    except Exception as e:
        print(f"Error fetching image for product '{product_name}': {e}")
        return None
```

Scope 4: Frontend & User Interaction
Handles the user interface and display of recommendations.

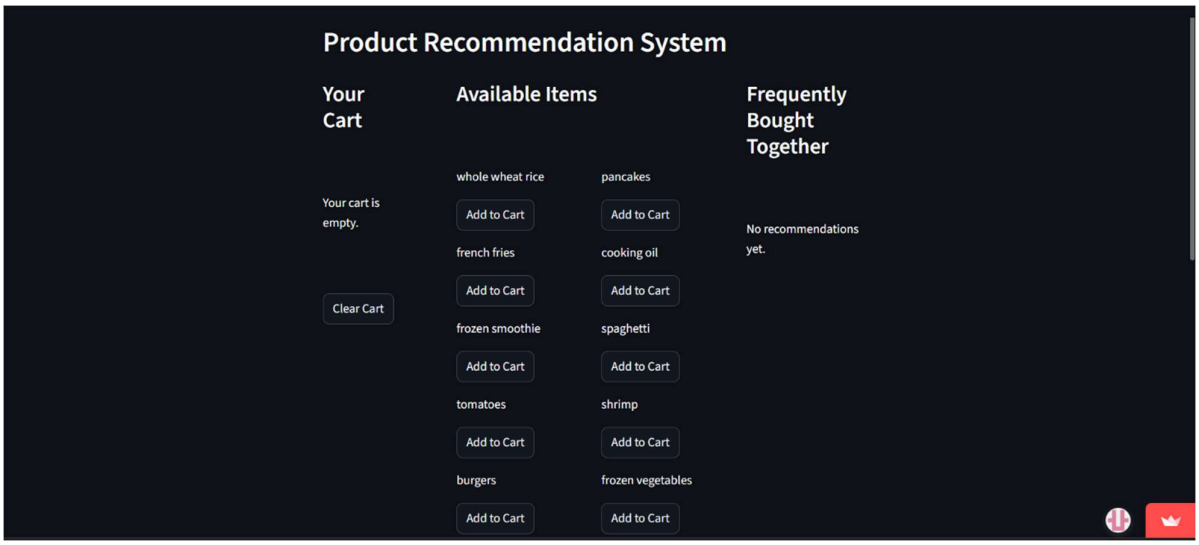
```
import streamlit as st

st.title('Product Recommendation System')

selected_product = st.selectbox(
    "Select a product:",
    products['name'].values
)

if st.button("Recommend"):
    names, images = recommend_products(selected_product)

    cols = st.columns(5)

    for i in range(5):
        if i < len(names):
            name = names[i]
            image = images[i]
        else:
            name = "N/A"
            image = None

        with cols[i]:
            if image:
                st.image(image, use_container_width=True)
            else:
                st.markdown(
                    "<div style='width:100%; height:300px; border:1px solid #ccc; display:flex; align-
items:center; justify-content:center;'>No Image</div>",
                    unsafe_allow_html=True
                )
            st.markdown(
                f"<div style='text-align: center; font-weight: bold;'>{name}</div>",
                unsafe_allow_html=True
            )
```

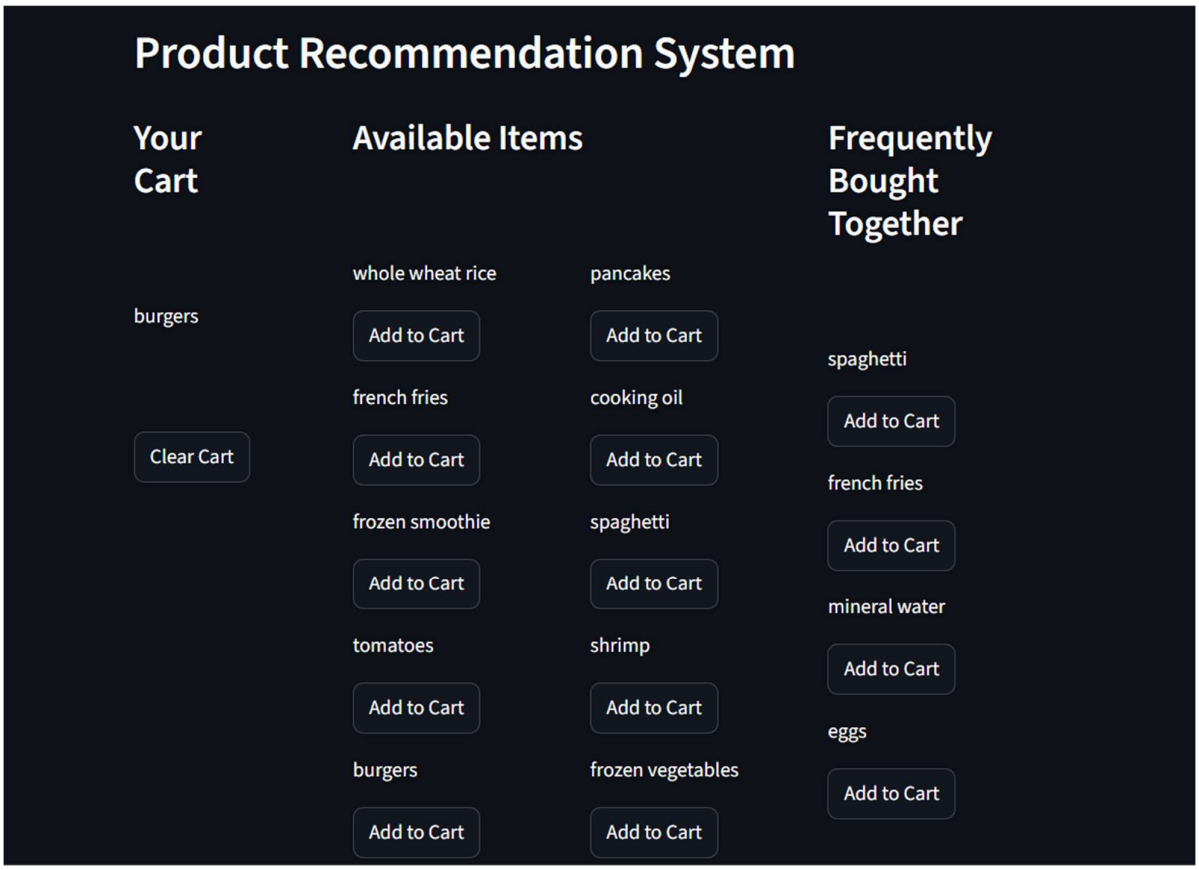
Implementation Code:

```
import streamlit as st
import json

# Load the apriori results from the JSON file
with open("apriori_results.json") as file:
    apriori_results = json.load(file)

# Initialize session state for cart and frequently bought together list
if "cart" not in st.session_state:
    st.session_state.cart = []
if "frequently_bought" not in st.session_state:
    st.session_state.frequently_bought = []

# Function to add items to the cart
def add_to_cart(item):
    if item not in st.session_state.cart:
        st.session_state.cart.append(item)
        # Update the frequently bought together list
        for fb_item in apriori_results.get(item, []):
            if fb_item not in st.session_state.frequently_bought:
                st.session_state.frequently_bought.append(fb_item)

# Function to clear the cart and frequently bought together list
def clear_cart():
    st.session_state.cart = []
    st.session_state.frequently_bought = []

st.header("Product Recommendation System \n")
# Layout with even more space for each section
col_left, col_center, col_right = st.columns([3, 7, 4], gap="large")

# Left Column: Cart Section
with col_left:
    st.subheader("Your Cart")
    st.write("###")  # Add space between the header and content
    if st.session_state.cart:
        for item in st.session_state.cart:
            st.write(item)
    else:
        st.write("Your cart is empty.")
    st.write("###")  # Add space before the clear button
    if st.button("Clear Cart"):
        clear_cart()

# Center Column: Available Items
with col_center:
    st.subheader("Available Items")
```

```
    st.write("###")  # Add space below the header
    item_cols = st.columns(2, gap="large")  # Only 2 items per row for maximum space
    for idx, item in enumerate(apriori_results.keys()):
        with item_cols[idx % 2]:  # Only 2 items per row
            st.write(item)
            st.button("Add to Cart", key=f"button_{item}", on_click=add_to_cart, args=(item,))

# Right Column: Frequently Bought Together Section
with col_right:
    st.subheader("Frequently Bought Together")
    st.write("###")  # Add space below the header
    if st.session_state.frequently_bought:
        fb_cols = st.columns(1, gap="large")  # 1 item per row for more space
        for fb_item in st.session_state.frequently_bought:
            with fb_cols[0]:
                st.write(fb_item)
                st.button("Add to Cart", key=f"fb_button_{fb_item}", on_click=add_to_cart,
args=(fb_item,))
    else:
        st.write("No recommendations yet.")
```

RESULTS :


Product Recommendation System
Your Cart
Your cart is empty.
Clear Cart
Available Items
whole wheat rice
Add to Cart
pancakes
Add to Cart
french fries
Add to Cart
cooking oil
Add to Cart
frozen smoothie
Add to Cart
spaghetti
Add to Cart
tomatoes
Add to Cart
shrimp
Add to Cart
burgers
Add to Cart
frozen vegetables
Add to Cart
Frequently Bought Together
No recommendations yet.


Product Recommendation System
Your Cart
burgers
Clear Cart
Available Items
whole wheat rice
Add to Cart
pancakes
Add to Cart
french fries
Add to Cart
cooking oil
Add to Cart
frozen smoothie
Add to Cart
spaghetti
Add to Cart
tomatoes
Add to Cart
shrimp
Add to Cart
burgers
Add to Cart
frozen vegetables
Add to Cart
Frequently Bought Together
spaghetti
Add to Cart
french fries
Add to Cart
mineral water
Add to Cart
eggs
Add to Cart

Conclusion

The Product Recommendation System provides an intelligent, data-driven solution for helping users discover products that match their preferences or past purchase behavior. Using precomputed association rules and similarity metrics, the system delivers fast and relevant recommendations. Integration with a user-friendly interface built on Streamlit allows for an interactive experience, making it accessible even to non-technical users. Overall, this system demonstrates the practical application of machine learning, data processing, and API integration to enhance user experience and improve decision-making in e-commerce.

Future Scope

1. **Personalized User Profiles**
    - Integrate user accounts to track purchase history, ratings, and preferences for more personalized recommendations.
2. **Hybrid Recommendation Systems**
    - Combine association rule mining (Apriori) with collaborative filtering or content-based filtering to improve recommendation accuracy.
3. **Real-time Data Updates**
    - Fetch live product details, prices, and stock availability from e-commerce APIs to keep recommendations up-to-date.
4. **Mobile and Web App Integration**
    - Expand beyond Streamlit to scalable web or mobile applications using frameworks like ReactJS for frontend and FastAPI or Flask for backend.
5. **Advanced Machine Learning Models**
    - Incorporate deep learning models such as autoencoders or sequence models for purchase pattern prediction and sequential recommendations.
6. **Caching and Performance Optimization**
    - Store frequently accessed product images or details locally, or use caching solutions like Redis to reduce API calls and improve system speed.
7. **User Interaction Features**
    - Add features like wishlist, product category filters, trending items, or social sharing options to enhance user engagement.