ONLINE RETAIL RECOMMENDATION SYSTEM

1. Introduction

This project focuses on developing a product recommendation system for an online retail platform. The goal is to predict and suggest products to customers based on the purchasing behaviors of other users. The system uses collaborative filtering, a popular approach for recommendation engines, where the primary technique employed is **cosine similarity**. By comparing the purchasing history of similar users, the system identifies products that a target user might be interested in. This report provides an in-depth explanation of the system's design, methodologies, implementation, results, and future improvements.

---

2. Problem Statement

In the context of an online retail environment, customers interact with a wide variety of products, generating large amounts of transactional data. The problem is to leverage this data to predict what products a customer is likely to buy next, based on the behavior of other customers who have made similar purchases. The system should return a list of the top n recommended products for a given user.

---

3. Dataset Overview

The dataset used for this project is a real-world e-commerce dataset that contains transactional data from an online retail store. The key features include:

- **InvoiceNo**: A unique identifier for each transaction.
- **StockCode**: A unique identifier for each product.
- **Description**: The name or description of the product.
- **Quantity**: The number of units of each product purchased in a transaction.

- **InvoiceDate**: The date and time when the transaction occurred.
- **CustomerID**: A unique identifier for each customer.
- **Country**: The country where the customer resides.

The recommendation system uses the CustomerID, StockCode, and Quantity columns to build a **user-item matrix** (pivot table). The matrix represents the interaction between users and products, forming the core input for the collaborative filtering algorithm.

---

4. Methodology

The project follows a structured approach, beginning with data preprocessing, followed by collaborative filtering to compute user similarities, and finally generating product recommendations. The core steps include:

4.1 Data Preprocessing

The first step is to prepare the dataset for analysis by transforming it into a form suitable for building a recommendation system.

Key tasks in data preprocessing:

- **Reading the CSV File**: The dataset is loaded into a pandas DataFrame.
- **Date Parsing**: The InvoiceDate column is parsed into a datetime format to ensure it is ready for future time-based analyses if needed. However, the recommendation system currently doesn’t leverage time, focusing instead on purchase quantities.
- **Creating the User-Item Matrix**: A pivot table is created where:
    - Rows represent customers (identified by CustomerID).
    - Columns represent products (identified by StockCode).
    - Values represent the total quantities of products purchased by each customer. This matrix allows us to compare customer preferences based on the products they bought.

The user-item matrix fills missing values with 0, indicating that the customer has not purchased that particular item.

Code Implementation:

```
def preprocess_data(file_path):
    df = pd.read_csv(file_path)

    # Convert InvoiceDate to datetime format
    df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

    # Create pivot table (user-item matrix)
    user_item_matrix = df.pivot_table(
        index='CustomerID',
        columns='StockCode',
        values='Quantity',
        aggfunc='sum',
        fill_value=0
    )

    return user_item_matrix
```

4.2 Collaborative Filtering via Cosine Similarity

The recommendation system uses collaborative filtering, which is based on the assumption that if two users have purchased similar items, they are likely to share similar preferences for other products as well. We use **cosine similarity** to measure the similarity between user vectors, which represent their purchasing behaviors.

Cosine Similarity:

- **Definition**: Cosine similarity measures the cosine of the angle between two vectors in a multi-dimensional space. In this case, the vectors represent the purchase histories of two users.
- **Formula**: cosine similarity(A,B)=A·B‖A‖‖B‖\text{cosine similarity}(A, B) = \frac{A \cdot B}{\|A\| \|B\|}cosine similarity(A,B)=‖A‖‖B‖A·B
- A cosine similarity value closer to 1 indicates greater similarity between two users. A value closer to 0 indicates dissimilarity.

We compute the cosine similarity between all pairs of users to form a **similarity matrix**.

Code Implementation:

```
from sklearn.metrics.pairwise import cosine_similarity


def compute_similarity(user_item_matrix):
    return cosine_similarity(user_item_matrix)
```

4.3 Recommendation Generation

To generate product recommendations for a given user, the following steps are taken:

1. **Identify Similar Users**:
    - For a given user_id, the system identifies the most similar users based on the cosine similarity matrix. These are the users whose purchasing behaviors are closest to the target user.
2. **Find Products for Recommendation**:
    - For each similar user, the system checks the products they have purchased but the target user has not.

   - These products are weighted by the similarity score of the similar user and the quantity they purchased, allowing the system to prioritize items bought by highly similar users.

3. **Rank and Return Recommendations**:
   - Products are ranked by their weighted scores, and the top n items are selected as recommendations.

Code Implementation:

```
def get_recommendations(user_id, user_item_matrix, similarity_matrix, n=5):
    # Get the index of the target user
    user_index = user_item_matrix.index.get_loc(user_id)

    # Find the most similar users, excluding the user themselves
    similar_users = similarity_matrix[user_index].argsort()[::-1][1:n+1]

    recommendations = {}

    for similar_user in similar_users:
        similar_user_id = user_item_matrix.index[similar_user]

        # Loop over all items and check if the target user hasn't purchased the item
        for item in user_item_matrix.columns:
            if user_item_matrix.loc[user_id, item] == 0 and user_item_matrix.loc[similar_user_id, item] > 0:
                # Calculate the weighted score for the item
                if item not in recommendations:
                    recommendations[item] = 0
                recommendations[item] += similarity_matrix[user_index][similar_user] * user_item_matrix.loc[similar_user_id, item]
    # Sort the items by their weighted score
    sorted_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)
    return [item for item, score in sorted_recommendations[:n]]
```

---

5. System Architecture

The architecture of the system is composed of three main components:

1. **Data Preprocessing**:
    - Converts raw transaction data into a **user-item matrix**.
2. **Similarity Computation**:
    - Computes the **cosine similarity** between all users to form a **similarity matrix**.
3. **Recommendation Generation**:
    - Based on the similarity scores, the system generates product recommendations by identifying items purchased by similar users but not yet purchased by the target user.

---

6. Results

When the system is executed with a sample user (user_id = 13047), it generates the following output:

```
Top 5 recommendations for user 13047:
1. StockCode: 85099B
2. StockCode: 84406B
3. StockCode: 22197
4. StockCode: 47566
5. StockCode: 22423
```

This output represents the top five products that user 13047 is most likely to purchase, based on their similarity to other customers.

---

7. Conclusion

This collaborative filtering recommendation system successfully predicts and recommends products to users by analyzing the purchasing behaviors of similar customers. By using cosine similarity, the system effectively identifies patterns of similarity between users and leverages these patterns to make predictions. The system is scalable and can be further improved to accommodate additional features and complexities.

---

8. Future Improvements

1. **Time-Based Recommendations**:
    - Currently, the system does not consider the recency of purchases. Integrating temporal factors could provide more relevant recommendations.
2. **Incorporate Demographic Data**:
    - Including customer demographic information (e.g., age, location) could enhance the model by enabling more personalized recommendations.
3. **Hybrid Recommendation System**:
    - Combine collaborative filtering with content-based filtering to increase the diversity of recommendations and overcome the limitations of relying solely on user behavior.
4. **Model Optimization**:
    - Optimize the computation of cosine similarity to improve the scalability of the system for large datasets.

9. Appendix:

Code

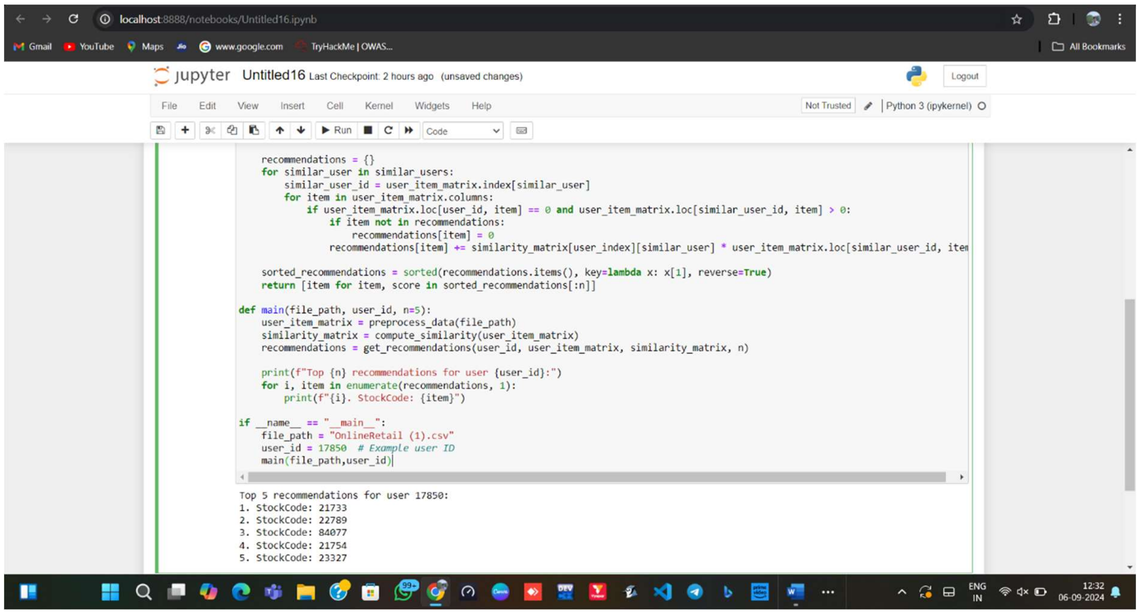
The full code for the system, including functions for data preprocessing, similarity computation, and recommendation generation, is available in the attached script.

```
import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity


def preprocess_data(file_path):

   df = pd.read_csv(file_path)


   df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])


   user_item_matrix = df.pivot_table(

      index='CustomerID',

      columns='StockCode',

      values='Quantity',

      aggfunc='sum',

      fill_value=0

   )


   return user_item_matrix

def compute_similarity(user_item_matrix):

   return cosine_similarity(user_item_matrix)
```

```
def get_recommendations(user_id, user_item_matrix, similarity_matrix, n=5):
    user_index = user_item_matrix.index.get_loc(user_id)
    similar_users = similarity_matrix[user_index].argsort()[::-1][1:n+1]

    recommendations = {}
    for similar_user in similar_users:
        similar_user_id = user_item_matrix.index[similar_user]
        for item in user_item_matrix.columns:
            if user_item_matrix.loc[user_id, item] == 0 and
user_item_matrix.loc[similar_user_id, item] > 0:
                if item not in recommendations:
                    recommendations[item] = 0
                recommendations[item] += similarity_matrix[user_index][similar_user]
* user_item_matrix.loc[similar_user_id, item]

    sorted_recommendations = sorted(recommendations.items(), key=lambda x:
x[1], reverse=True)
    return [item for item, score in sorted_recommendations[:n]]

def main(file_path, user_id, n=5):
    user_item_matrix = preprocess_data(file_path)
    similarity_matrix = compute_similarity(user_item_matrix)
    recommendations = get_recommendations(user_id, user_item_matrix,
similarity_matrix, n)

    print(f"Top {n} recommendations for user {user_id}:")
    for i, item in enumerate(recommendations, 1):
        print(f"{i}. StockCode: {item}")


if __name__ == "__main__":
    file_path = "OnlineRetail (1).csv"
    user_id = 17850  # Example user ID
    main(file_path,user_id)
```

---

10. Screenshot :



localhost:8888/notebooks/Untitled16.ipynb
jupyter Untitled16 Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

```
    recommendations = {}
    for similar_user in similar_users:
        similar_user_id = user_item_matrix.index[similar_user]
        for item in user_item_matrix.columns:
            if user_item_matrix.loc[user_id, item] == 0 and user_item_matrix.loc[similar_user_id, item] > 0:
                if item not in recommendations:
                    recommendations[item] = 0
                recommendations[item] += similarity_matrix[user_index][similar_user] * user_item_matrix.loc[similar_user_id, iten

    sorted_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)
    return [item for item, score in sorted_recommendations[:n]]

def main(file_path, user_id, n=5):
    user_item_matrix = preprocess_data(file_path)
    similarity_matrix = compute_similarity(user_item_matrix)
    recommendations = get_recommendations(user_id, user_item_matrix, similarity_matrix, n)

    print(f"Top {n} recommendations for user {user_id}:")
    for i, item in enumerate(recommendations, 1):
        print(f"{i}. StockCode: {item}")

if __name__ == "__main__":
    file_path = "OnlineRetail (1).csv"
    user_id = 17850  # Example user ID
    main(file_path,user_id)
```

```
Top 5 recommendations for user 17850:
1. StockCode: 21733
2. StockCode: 22789
3. StockCode: 84077
4. StockCode: 21754
5. StockCode: 23327
```

---

11. Project Link:

https://drive.google.com/drive/folders/1c1nu-OdUhDUIbgNR7bVclPhnuiWfT8_w?usp=drive_link

---