```
from google.colab import files
uploaded = files.upload()
```

Choose Files Recomme…dataset.csv

Recommendation_dataset.csv(text/csv) - 64089 bytes, last modified: 1/30/2026 - 100% done

```
Saving Recommendation_dataset.csv to Recommendation_dataset.csv
```

```
# Import libraries*

import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Load data*

df = pd.read_csv("/content/Recommendation_dataset.csv")
df.head()
```

	User ID	Product ID	Product Name	Brand	Category	Price	Rating	Color	Size
0	19	1	Dress	Adidas	Men's Fashion	40	1.043159	Black	XL
1	97	2	Shoes	H&M	Women's Fashion	82	4.026416	Black	L
2	25	3	Dress	Adidas	Women's Fashion	44	3.337938	Yellow	XL
3	57	4	Shoes	Zara	Men's Fashion	23	1.049523	White	S
4	79	5	T-shirt	Adidas	Men's Fashion	79	4.302773	Black	M

Next steps: Generate code with df | New interactive sheet

```
df = df[['User ID', 'Product ID', 'Rating']]
```

```
df.info
```

```
pandas.core.frame.DataFrame.info
def info(verbose: bool | None=None, buf: WriteBuffer[str] | None=None,
max_cols: int | None=None, memory_usage: bool | str | None=None,
show_counts: bool | None=None) -> None
```

```
    made based in column dtype and number of rows assuming values
    consume the same memory amount for corresponding dtypes. With deep
    memory introspection, a real memory usage calculation is performed
    at the cost of computational resources. See the
    :ref:`Frequently Asked Questions <df-memory-usage>` for more
    details.
show_counts : bool, optional
```

```
df.isnull().sum()
```

	0
User ID	0
Product ID	0
Rating	0

dtype: int64

```
print(f"Shape of DataFrame before removing duplicates: {df.shape}")
df.drop_duplicates(inplace=True)
print(f"Shape of DataFrame after removing duplicates: {df.shape}")
```

```
Shape of DataFrame before removing duplicates: (1000, 3)
Shape of DataFrame after removing duplicates: (1000, 3)
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
print("First 5 'Product ID' entries:")
display(df['Product ID'].head())

print("\nUnique 'Product ID' values:")
display(df['Product ID'].unique())
```

Show hidden output

```
print(df.columns)
```

```
#*Step 3. Create user item matrix*

user_item_matrix = df.pivot_table(
```

```
    index='User ID',
    columns='Product ID',
    values='Rating'
)
```

```
#Step 4. Handle missing values*
user_item_matrix.fillna(0, inplace=True)
df.head()
```

1 to 5 of 5 entries Filter

index	User ID	Product ID	Rating
0	19	1	1.0431592108361825
1	97	2	4.026416271141911
2	25	3	3.337937559377053
3	57	4	1.0495229563128543
4	79	5	4.302773408398684

Show 25 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

Next steps: Generate code with df | New interactive sheet

```
user_similarity[0].shape
```

```
(100,)
```

```
#*Step 5. Compute user similarity*
user_similarity = cosine_similarity(user_item_matrix)
user_similarity_df = pd.DataFrame(
    user_similarity,
    index=user_item_matrix.index,
    columns=user_item_matrix.index
)
```

```
user_similarity[0].shape
```

```
(100,)
```

```
user_similarity.shape
```

```
(100, 100)
```

```
#*Step 6. Find similar users*

User_ID = 1
```

```
similar_users = user_similarity_df[user_id].sort_values(ascending=False)
similar_users.head()

#Top result User itself score 1. Ignore it.
```

User ID	1
1	1.0
2	0.0
3	0.0
4	0.0
5	0.0

dtype: float64

```
User_ID = 19   # example user

similar_users = user_similarity_df[User_ID].sort_values(ascending
similar_users = similar_users[similar_users.index != User_ID]
df.head
```

pandas.core.generic.NDFrame.head
def head(n: int=5) -> Self

/usr/local/lib/python3.12/dist-packages/pandas/core/generic.py
Return the first `n` rows.

This function returns the first `n` rows for the object based
on position. It is useful for quickly testing if your object
has the right type of data in it.

```
similar_users = similar_users[similar_users > 0.3]
```

```
#Step 7. Recommend items*

#Get items rated by similar users

similar_users = similar_users[similar_users.index != user_id]
weighted_ratings = user_item_matrix.loc[similar_users.index].T.do
recommendations = weighted_ratings.sort_values(ascending=False)
#Remove already rated items.
```

```
already_rated = user_item_matrix.loc[user_id]
already_rated = already_rated[already_rated > 0].index
recommendations = recommendations.drop(already_rated)
recommendations.head(5)
#Output Top 3 recommended item IDs.
```

Product ID	0
1000	0.0
1	0.0
2	0.0
3	0.0
4	0.0

dtype: float64

```
item_similarity = cosine_similarity(user_item_matrix.T)
```

```
# User Profile = Σ (Item Features * User Rating)
# Final Similarity = 0.5 * TextSim + 0.3 * CategorySim + 0.2 * Pr
```

```
Start coding or generate with AI.
```

Double-click (or enter) to edit

Key Insights (Bullet Points)

Recommendations are based on product similarity, not user similarity

Product similarity is calculated using cosine similarity on rating patterns

Item-based systems are more scalable and stable than user-based systems

Similarity threshold improves recommendation quality

Works well for large user bases and stable product catalogs