Task 2: Lookalike Model

Build a Lookalike Model that takes a user's information as input and recommends 3 similar customers based on their profile and transaction history

```
# Importing common libraries :
import pandas as pd                # Data manipulation
import numpy as np                 # Numerical operations
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity

# Read the CSV files
products = pd.read_csv('C:\\Users\\kampl\\Downloads\\Products.csv')
products
```

```
   ProductID              ProductName     Category   Price
0       P001     ActiveWear Biography        Books  169.30
1       P002     ActiveWear Smartwatch  Electronics  346.30
2       P003   ComfortLiving Biography        Books   44.12
3       P004            BookWorld Rug   Home Decor   95.69
4       P005         TechPro T-Shirt     Clothing  429.31
..       ...                      ...          ...     ...
95      P096     SoundWave Headphones  Electronics  307.47
96      P097       BookWorld Cookbook        Books  319.34
97      P098         SoundWave Laptop  Electronics  299.93
98      P099     SoundWave Mystery Book        Books  354.29
99      P100        HomeSense Sweater     Clothing  126.34

[100 rows x 4 columns]
```

```
customers = pd.read_csv('C:\\Users\\kampl\\Downloads\\Customers.csv')
customers
```

```
    CustomerID        CustomerName         Region  SignupDate
0        C0001     Lawrence Carroll  South America  2022-07-10
1        C0002       Elizabeth Lutz           Asia  2022-02-13
2        C0003       Michael Rivera  South America  2024-03-07
3        C0004   Kathleen Rodriguez  South America  2022-10-09
4        C0005          Laura Weber           Asia  2022-08-15
..         ...                 ...            ...         ...
195      C0196          Laura Watts         Europe  2022-06-07
196      C0197     Christina Harvey         Europe  2023-03-21
197      C0198         Rebecca Ray         Europe  2022-02-27
198      C0199      Andrea Jenkins         Europe  2022-12-03
199      C0200          Kelly Cross           Asia  2023-06-11

[200 rows x 4 columns]
```

```
transactions = pd.read_csv('C:\\Users\\kampl\\Downloads\\
Transactions.csv')
transactions

    TransactionID CustomerID ProductID      TransactionDate  Quantity  \
0          T00001      C0199      P067  2024-08-25 12:38:23         1

1          T00112      C0146      P067  2024-05-27 22:23:54         1

2          T00166      C0127      P067  2024-04-25 07:38:55         1

3          T00272      C0087      P067  2024-03-26 22:55:37         2

4          T00363      C0070      P067  2024-03-21 15:10:10         3

..            ...        ...       ...                  ...       ...

995        T00496      C0118      P037  2024-10-24 08:30:27         1

996        T00759      C0059      P037  2024-06-04 02:15:24         3

997        T00922      C0018      P037  2024-04-05 13:05:32         4

998        T00959      C0115      P037  2024-09-29 10:16:02         2

999        T00992      C0024      P037  2024-04-21 10:52:24         1


     TotalValue   Price
0        300.68  300.68
1        300.68  300.68
2        300.68  300.68
3        601.36  300.68
4        902.04  300.68
..          ...     ...
995      459.86  459.86
996     1379.58  459.86
997     1839.44  459.86
998      919.72  459.86
999      459.86  459.86

[1000 rows x 7 columns]

# Save the recommendations to a CSV fileimport pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

# Merge datasets
transactions_with_products = transactions.merge(products,
on="ProductID", suffixes=('_transaction', '_product'))
```

```python
customer_data = transactions_with_products.merge(customers,
on="CustomerID")

# Feature Engineering: Create aggregated features for each customer
customer_features = customer_data.groupby('CustomerID').agg(
    TotalSpent=('TotalValue', 'sum'),
    NumTransactions=('TransactionID', 'count'),
    Region=('Region', 'first'),  # Use the first region occurrence
    Books=('Category', lambda x: (x == 'Books').sum()),
    Electronics=('Category', lambda x: (x == 'Electronics').sum()),
    HomeDecor=('Category', lambda x: (x == 'Home Decor').sum()),
    Clothing=('Category', lambda x: (x == 'Clothing').sum())
).reset_index()

# Encode categorical data and standardize numerical features
customer_features['Region'] =
customer_features['Region'].astype('category').cat.codes
scaler = StandardScaler()
features = scaler.fit_transform(customer_features.iloc[:, 1:])

# Calculate cosine similarity
similarity_matrix = cosine_similarity(features)

# Generate recommendations for the first 20 customers
recommendations = {}
customer_ids = customer_features['CustomerID'].tolist()

for idx in range(20):  # First 20 customers
    customer_id = customer_ids[idx]
    similarities = list(enumerate(similarity_matrix[idx]))
    # Sort by similarity score (excluding self-similarity)
    similar_customers = sorted(similarities, key=lambda x: x[1],
reverse=True)[1:4]
    recommendations[customer_id] = [(customer_ids[sim[0]],
round(sim[1], 2)) for sim in similar_customers]

# Save recommendations to CSV
recommendations_df = pd.DataFrame([
    {"CustomerID": cust, "Recommendations": str(recs)}
    for cust, recs in recommendations.items()
])

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

# Merge datasets
transactions_with_products = transactions.merge(products,
on="ProductID", suffixes=('_transaction', '_product'))
customer_data = transactions_with_products.merge(customers,
on="CustomerID")
```

```
# Feature Engineering: Create aggregated features for each customer
customer_features = customer_data.groupby('CustomerID').agg(
    TotalSpent=('TotalValue', 'sum'),
    NumTransactions=('TransactionID', 'count'),
    Region=('Region', 'first'),  # Use the first region occurrence
    Books=('Category', lambda x: (x == 'Books').sum()),
    Electronics=('Category', lambda x: (x == 'Electronics').sum()),
    HomeDecor=('Category', lambda x: (x == 'Home Decor').sum()),
    Clothing=('Category', lambda x: (x == 'Clothing').sum())
).reset_index()

# Encode categorical data and standardize numerical features
customer_features['Region'] =
customer_features['Region'].astype('category').cat.codes
scaler = StandardScaler()
features = scaler.fit_transform(customer_features.iloc[:, 1:])

# Calculate cosine similarity
similarity_matrix = cosine_similarity(features)

# Generate recommendations for the first 20 customers
recommendations = {}
customer_ids = customer_features['CustomerID'].tolist()

for idx in range(20):  # First 20 customers
    customer_id = customer_ids[idx]
    similarities = list(enumerate(similarity_matrix[idx]))
    # Sort by similarity score (excluding self-similarity)
    similar_customers = sorted(similarities, key=lambda x: x[1],
reverse=True)[1:4]
    recommendations[customer_id] = [(customer_ids[sim[0]],
round(sim[1], 2)) for sim in similar_customers]

# Function to calculate average similarity score for top
recommendations
def calculate_average_similarity(recommendations):
    all_scores = []
    for recs in recommendations.values():
        all_scores.extend([score for _, score in recs])
    avg_score = np.mean(all_scores)
    return avg_score

# Function to evaluate recommendation diversity
def calculate_diversity(recommendations):
    all_recommended_customers = [
        recommended[0] for recs in recommendations.values() for
recommended in recs
    ]
    unique_customers = len(set(all_recommended_customers))
```

```python
    total_recommendations = len(all_recommended_customers)
    diversity = unique_customers / total_recommendations
    return diversity

# Inspect top recommendations for the first 5 customers
print("Top Recommendations for First 5 Customers:")
for customer, recs in list(recommendations.items())[:5]:
    print(f"Customer {customer}'s recommendations: {recs}")

# Calculate average similarity score
avg_similarity_score = calculate_average_similarity(recommendations)
print(f"\nAverage Similarity Score of Top Recommendations:
{avg_similarity_score:.2f}")

# Calculate diversity of recommendations
diversity_score = calculate_diversity(recommendations)
print(f"Recommendation Diversity Score: {diversity_score:.2f} (Higher
is better)")

# Analyze recommendations for a specific customer
specific_customer = "C0001"
if specific_customer in recommendations:
    print(f"\nDetailed Recommendations for {specific_customer}:")
    for rec_customer, score in recommendations[specific_customer]:
        print(f" - Customer: {rec_customer}, Similarity Score:
{score:.2f}")
else:
    print(f"Customer {specific_customer} not found in
recommendations.")
```

```
Top Recommendations for First 5 Customers:
Customer C0001's recommendations: [('C0091', 0.85), ('C0055', 0.85),
('C0190', 0.84)]
Customer C0002's recommendations: [('C0159', 0.92), ('C0134', 0.92),
('C0106', 0.9)]
Customer C0003's recommendations: [('C0031', 0.98), ('C0158', 0.96),
('C0129', 0.87)]
Customer C0004's recommendations: [('C0113', 0.92), ('C0122', 0.9),
('C0194', 0.9)]
Customer C0005's recommendations: [('C0007', 0.99), ('C0197', 0.94),
('C0140', 0.93)]

Average Similarity Score of Top Recommendations: 0.89
Recommendation Diversity Score: 0.93 (Higher is better)

Detailed Recommendations for C0001:
 - Customer: C0091, Similarity Score: 0.85
 - Customer: C0055, Similarity Score: 0.85
 - Customer: C0190, Similarity Score: 0.84
```

Top Recommendations for First 5 Customers

The recommendations for customers C0001 to C0005 demonstrate that the Lookalike Model successfully identifies similar customers with meaningful similarity scores. Key observations:

1. Customer C0001:
   - Top matches include customers C0091, C0055, and C0190, with scores of 0.85, 0.85, and 0.84 respectively. These scores indicate a high degree of similarity based on shared transactional or profile features.
2. Customer C0002:
   - Recommendations are highly similar, with top scores reaching 0.92 (C0159, C0134). This could suggest customers in similar demographics or purchasing patterns.
3. Customer C0003:
   - The top match (C0031) has a similarity score of 0.98, indicating a near-identical behavior/profile.

Overall Metrics

1. Average Similarity Score:
   - The model achieves an average similarity score of 0.89, reflecting its ability to generate highly relevant recommendations. Scores above 0.80 typically indicate a strong level of alignment.
2. Recommendation Diversity:
   - A diversity score of 0.93 shows that the recommendations are not overly repetitive across customers. This suggests the model is effective at exploring the dataset and identifying unique matches.

Business Insights

1. High Relevance:
   - Recommendations are highly relevant and can be used to drive personalized marketing. For example:
     - Targeting C0001 with promotions similar to what worked for C0091 or C0055.
   - Customer segmentation for creating loyalty programs.
2. Actionable Patterns:
   - Scores near 1.0 (C0003's match with C0031) identify near-identical profiles. These pairs can be directly leveraged for predicting future purchasing behaviors.
3. Diversity in Recommendations:
   - The high diversity score ensures that recommendations do not cluster around a few customers, allowing a broader marketing strategy.

Model Accuracy and Recommendation Quality

1. Accuracy:
   - The high similarity scores and consistent patterns confirm that the model successfully identifies meaningful relationships.
   - The inclusion of customer and product features ensures the model considers diverse aspects like purchasing behavior and demographics.
2. Recommendation Quality:
   - Recommendations are highly interpretable, with each similarity score providing a clear measure of closeness.
   - The quality of matches (scores > 0.80) indicates strong alignment between customer profiles.