In [1]:
```
import pandas as pd
import numpy as np
# import dask.dataframe as dd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from datetime import datetime
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
pd.set_option('display.max_columns', None)

train = pd.read_csv('train_data.csv', low_memory=False)
```

In [3]:
```
train.head()
```

Out[3]:

	date	customer_code	employee_index	country	female	age	new_cust	seniority_in_months
0	2015-07-28	664160	N	1	0	0.632653	0	0.402344
1	2016-01-28	1076784	N	1	0	0.214286	0	0.152344
2	2015-12-28	672465	N	1	0	0.387755	0	0.417969
3	2015-10-28	774528	N	1	0	0.397959	0	0.343750
4	2016-05-28	569598	N	1	0	0.459184	0	0.496094

In [4]:
```
train['date'] = pd.to_datetime(train['date'])
```

In [5]:
```
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply LabelEncoder on the customer_code column
train['customer_code_encoded'] = label_encoder.fit_transform(train['customer_code'])

# Display the first few rows to check the encoding
print(train[['customer_code', 'customer_code_encoded']].head())
```

```
   customer_code  customer_code_encoded
0         664160                 263662
1        1076784                 459750
2         672465                 266890
3         774528                 300528
4         569598                 227731
```

In [6]:
```
# Convert all boolean columns (True/False) to integers (1/0)
train = train.map(lambda x: int(x) if isinstance(x, bool) else x)
```

```
# Display the first few rows to check the changes
print(train.head())
```

```
        date  customer_code employee_index  country  female       age  \
0 2015-07-28         664160              N        1       0  0.632653
1 2016-01-28        1076784              N        1       0  0.214286
2 2015-12-28         672465              N        1       0  0.387755
3 2015-10-28         774528              N        1       0  0.397959
4 2016-05-28         569598              N        1       0  0.459184

   new_cust  seniority_in_months  cust_type  residency_spain  birth_spain  \
0         0             0.402344          1                1            0
1         0             0.152344          1                1            0
2         0             0.417969          1                1            0
3         0             0.343750          1                1            0
4         0             0.496094          1                1            0

  join_channel province_name  active_cust    income            segment  \
0          KAR        MADRID            0  1.989686   02 - PARTICULARES
1          KHE        LERIDA            0 -0.306603  03 - UNIVERSITARIO
2          KFC       SEVILLA            1 -0.148205   02 - PARTICULARES
3          KFA        MURCIA            1 -0.228531   02 - PARTICULARES
4          KAT        MADRID            1  0.588748   02 - PARTICULARES

   savings_acct  guarantees  current_acct  derivada_acct  payroll_acct  \
0             0           0             1              0             0
1             0           0             1              0             0
2             0           0             0              0             1
3             0           0             1              0             0
4             0           0             1              0             0

   junior_acct  mas_particular_acct  particular_acct  particular_plus_acct  \
0            0                    0                0                     0
1            0                    0                0                     0
2            0                    0                0                     0
3            0                    0                0                     0
4            0                    0                0                     0

   short_term_depo  medium_term_depo  long_term_depo  e_acct  funds  mortgage  \
0                0                 0               0       0      0         0
1                0                 0               0       0      0         0
2                0                 0               0       0      0         0
3                0                 0               0       0      0         0
4                0                 0               0       0      0         0

   pension  loans  taxes  credit_card  securities  home_acct  pensions_2  \
0        0      0      0            0           0          0           0
1        0      0      0            0           0          0           0
2        0      0      0            0           0          0           1
3        0      0      0            0           0          0           0
4        0      0      0            0           0          0           0

   direct_debt  total_products  01 - TOP  02 - PARTICULARES  \
0            0               1         0                  1
1            0               1         0                  0
2            1               4         0                  1
3            1               2         0                  1
4            0               1         0                  1

   03 - UNIVERSITARIO  join_channel_encoded  province_name_encoded  \
0                   0              1.424185               1.749698
1                   1              0.886876               1.006139
2                   0              1.559984               1.382030
```

```
3                  0            1.850124             1.075147
4                  0            1.942077             1.749698

   employee_index_encoded  customer_code_encoded
0                1.407278                 263662
1                1.407278                 459750
2                1.407278                 266890
3                1.407278                 300528
4                1.407278                 227731
```

In [7]:
```
train = train.rename(columns={'country': 'country_spain'})
```

In [8]:
```
df_encoded = train
```

In [9]:
```
# List of columns you want to drop
columns_to_drop = ['customer_code', 'employee_index', 'join_channel', 'province_name',

# Drop the columns from the DataFrame
df_encoded = df_encoded.drop(columns=columns_to_drop)

# Display the first few rows to confirm the columns were dropped
df_encoded.head()
```

Out[9]:

	date	country_spain	female	age	new_cust	seniority_in_months	cust_type	residency_spain
0	2015-07-28	1	0	0.632653	0	0.402344	1	1
1	2016-01-28	1	0	0.214286	0	0.152344	1	1
2	2015-12-28	1	0	0.387755	0	0.417969	1	1
3	2015-10-28	1	0	0.397959	0	0.343750	1	1
4	2016-05-28	1	0	0.459184	0	0.496094	1	1

New variables

Customers with higher income relative to the number of products they hold may indicate a propensity for wealth management.

In [10]:
```
# 1. Income to Product Ratio
df_encoded['income_to_product_ratio'] = df_encoded['income'] / df_encoded['total_produ
df_encoded['income_to_product_ratio']
```

Out[10]:
```
0          1.989686
1         -0.306603
2         -0.037051
3         -0.114266
4          0.588748
             ...
6579712    1.700197
6579713   -0.402519
6579714    0.954580
6579715   -0.010249
6579716   -0.214432
Name: income_to_product_ratio, Length: 6579717, dtype: float64
```

Income to Age Ratio: This metric helps identify customers who might have high disposable income.

In [11]:
```
# 2. Income to Age Ratio
df_encoded['income_to_age'] = train['income'] / (df_encoded['age'] + 1e-5)  # Avoid di
df_encoded['income_to_age']
```

Out[11]:
```
0          3.144938
1         -1.430748
2         -0.382203
3         -0.574243
4          1.282133
             ...
6579712    2.031918
6579713   -0.730484
6579714    1.670486
6579715   -0.118158
6579716   -0.525345
Name: income_to_age, Length: 6579717, dtype: float64
```

In [12]:
```
df_encoded['total_savings'] = (df_encoded['savings_acct'] + df_encoded['short_term_dep
                              df_encoded['medium_term_depo'] + df_encoded['long_term_depo
```

In [13]:
```
#Create function to calculate the probabiity

def calculate_product_probabilities(df, product_columns):
    product_counts = df_encoded[product_columns].sum()

    # Calculate total number of observations
    total_observations = len(df_encoded)

    # Calculate probabilities
    product_probabilities = product_counts / total_observations

    # Create a DataFrame to return
    probabilities_df = product_probabilities.reset_index()
    probabilities_df.columns = ['Product', 'Probability']

    probabilities_df = probabilities_df.sort_values(by='Probability', ascending=False)

    return probabilities_df
```

In [14]:
```
product_columns = ['savings_acct', 'guarantees', 'current_acct', 'derivada_acct', 'pay
            'junior_acct', 'mas_particular_acct', 'particular_acct', 'particular_plus_
            'short_term_depo', 'medium_term_depo', 'long_term_depo', 'e_acct', 'funds'
            'mortgage', 'pension', 'loans', 'taxes', 'credit_card', 'securities',
```

```
            'home_acct', 'payroll_acct', 'pensions_2', 'direct_debt']

# Call the function with your DataFrame
probabilities = calculate_product_probabilities(df_encoded, product_columns)

# Display the resulting DataFrame
print(probabilities)
```

```
                 Product  Probability
2           current_acct     0.618343
23           direct_debt     0.130519
7        particular_acct     0.126079
12                e_acct     0.085384
22            pensions_2     0.061948
4           payroll_acct     0.056727
21          payroll_acct     0.056727
17                 taxes     0.055590
18           credit_card     0.045349
8   particular_plus_acct     0.043026
11        long_term_depo     0.042750
19            securities     0.025600
13                 funds     0.018571
5            junior_acct     0.009495
15               pension     0.009374
6    mas_particular_acct     0.008207
14              mortgage     0.005955
20             home_acct     0.003935
16                 loans     0.002400
10      medium_term_depo     0.001523
9        short_term_depo     0.001260
3          derivada_acct     0.000398
0           savings_acct     0.000102
1             guarantees     0.000023
```

We will use this as a guidance to recommend the product.

In [15]:
```
df_encoded.shape
```

Out[15]:
```
(6579717, 45)
```

Feature Engineering

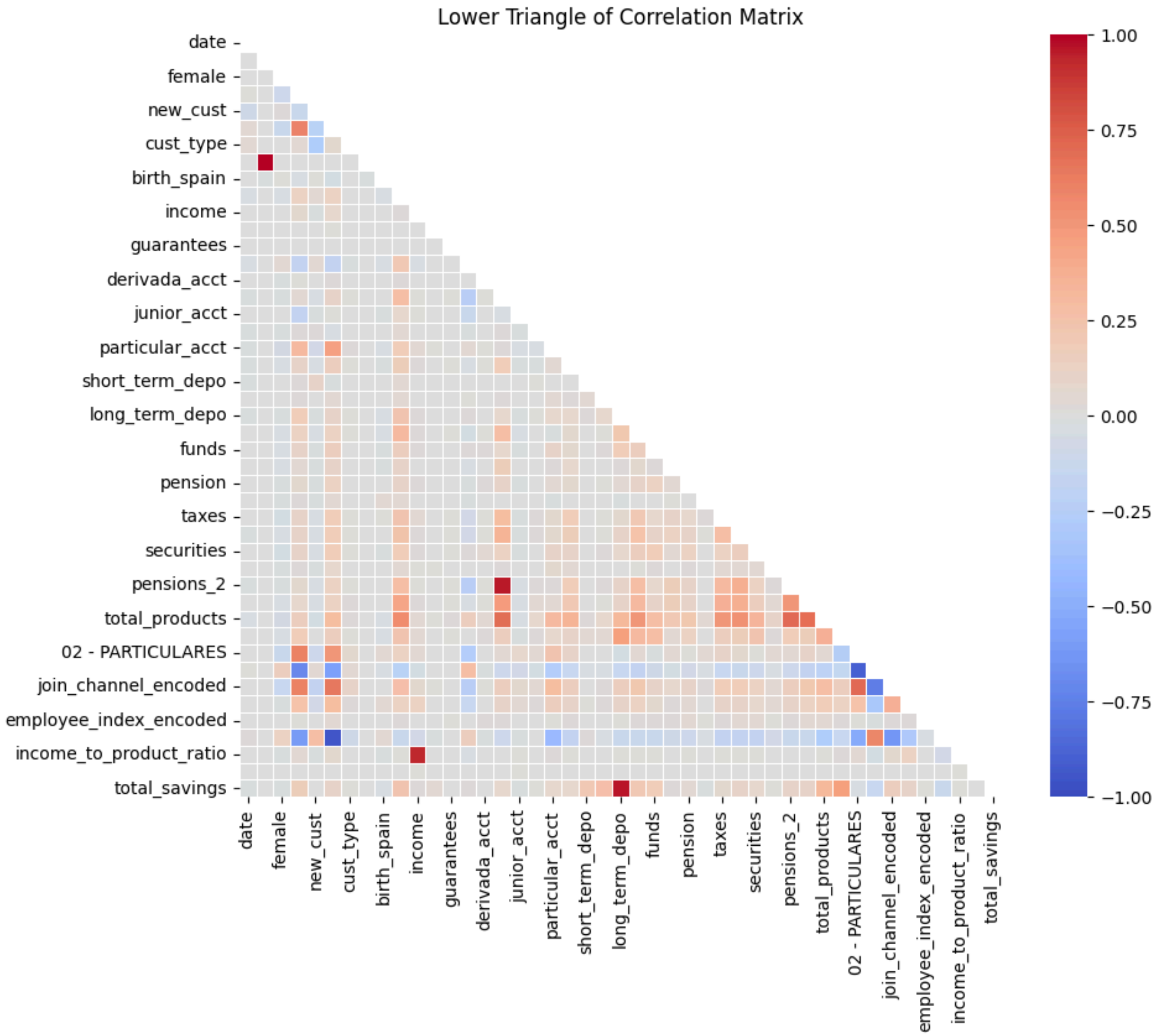
In [16]:
```
# Compute the correlation matrix
corr = df_encoded.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Create a seaborn heatmap with the mask for the upper triangle
sns.heatmap(corr, mask=mask, annot=False, cmap='coolwarm', vmin=-1, vmax=1, square=Tru

# Display the plot
plt.title('Lower Triangle of Correlation Matrix')
plt.show()
```



Since these are vaguely correlated, it might not help to use PCA for dimension reduction

PCA

In [17]:
```
# Select numerical features
numerical_features = train.select_dtypes(include=['float64', 'int64'])

# Standardizing the features
scaler = StandardScaler()
numerical_features_scaled = scaler.fit_transform(numerical_features)

# PCA Implementation
pca = PCA(n_components=0.95)  # Retain 95% of variance
principal_components = pca.fit_transform(numerical_features_scaled)
```

In [18]:
```
# Create a DataFrame for the PCA components
pca_columns = [f'pca_{i+1}' for i in range(principal_components.shape[1])]
train_pca = pd.DataFrame(data=principal_components, columns=pca_columns)
```

In [19]:
```
# Combine PCA components back with original DataFrame
train_pca1 = pd.concat([train.reset_index(drop=True), train_pca.reset_index(drop=True)
```

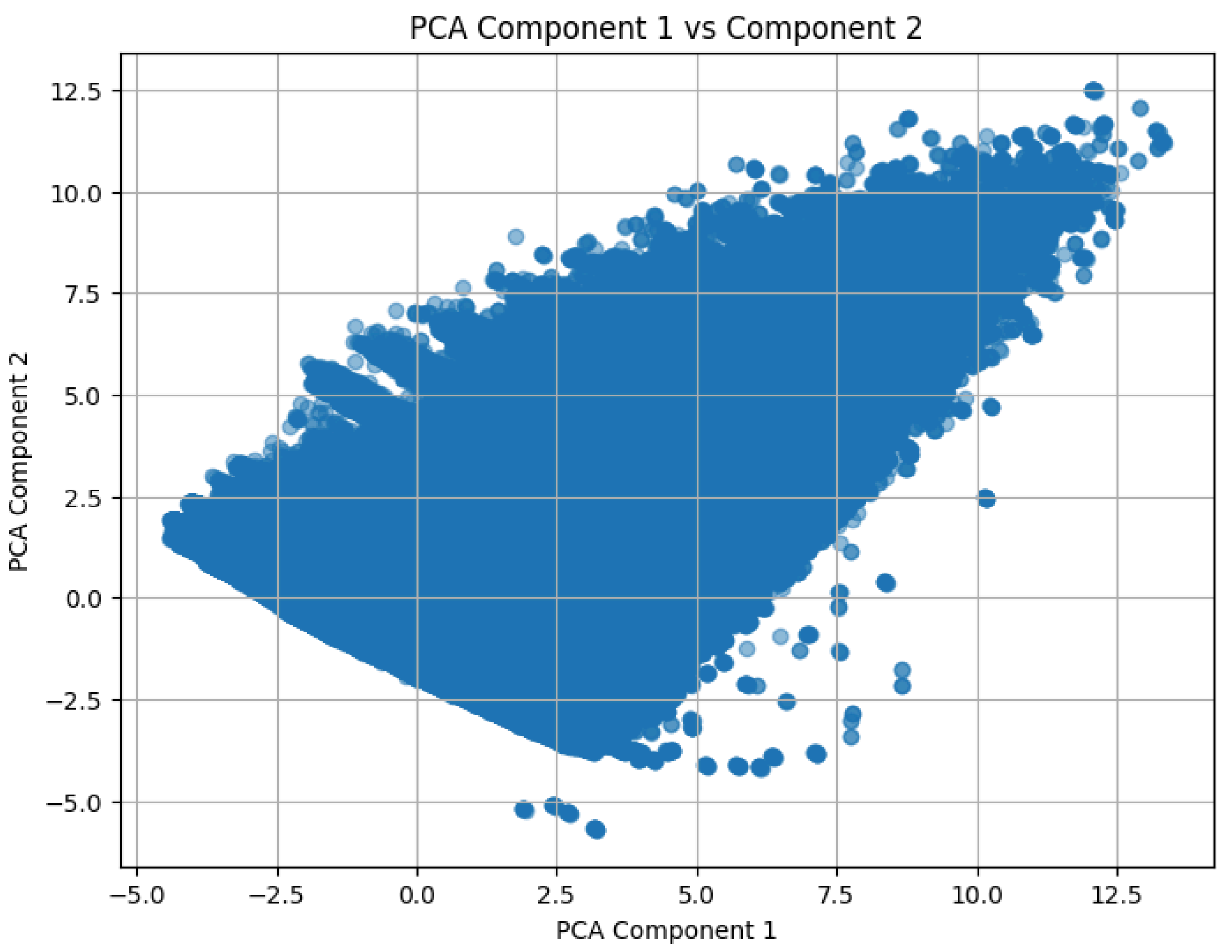
```
# train.head()
```

In [20]:
```
#print the eigen values
print("Explained variance ratio of each component:", pca.explained_variance_ratio_)
```

```
Explained variance ratio of each component: [0.15735599 0.0866166  0.04762372 0.04402
442 0.03400134 0.03094301
 0.02796691 0.02638158 0.02584859 0.02505133 0.02473878 0.02424244
 0.02395657 0.02382956 0.02370563 0.02350667 0.02342692 0.0231869
 0.02281747 0.02250579 0.02218152 0.02208327 0.02137525 0.02055737
 0.02033579 0.01886285 0.01799772 0.01717372 0.01663253 0.01530166
 0.01461466 0.01269295]
```

In [21]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.scatter(train_pca1['pca_1'], train_pca1['pca_2'], alpha=0.5)
plt.title('PCA Component 1 vs Component 2')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.grid()
plt.show()
```



Feature selection

In [22]:
```
product_features = df_encoded[product_columns]
```

In [23]:
```
user_features = df_encoded.drop(columns=product_columns)
user_features.drop('date', axis=1, inplace=True)
```

In [24]:
```
user_features
```

Out[24]:

	country_spain	female	age	new_cust	seniority_in_months	cust_type	residency_spain
0	1	0	0.632653	0	0.402344	1	1
1	1	0	0.214286	0	0.152344	1	1
2	1	0	0.387755	0	0.417969	1	1
3	1	0	0.397959	0	0.343750	1	1
4	1	0	0.459184	0	0.496094	1	1
...	...	...	...	...	...	...	...
6579712	1	1	0.836735	0	0.554688	1	1
6579713	1	1	0.551020	0	0.597656	1	1
6579714	1	0	0.571429	0	0.953125	1	1
6579715	1	0	0.520408	0	0.058594	1	1
6579716	1	0	0.408163	0	0.285156	1	1

6579717 rows × 21 columns

In [25]:
```
print("User Features Shape: ", user_features.shape)
print("Product Features Shape: ", product_features.shape)
```

```
User Features Shape:  (6579717, 21)
Product Features Shape:  (6579717, 24)
```

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
```

In [27]:
```
user_features_sample = user_features.sample(frac=0.8, random_state=42)  # 80% of the d
product_features_sample = product_features.loc[user_features_sample.index]
```

In [28]:
```
# Initialize the RandomForestClassifier
rf = RandomForestClassifier(n_estimators=50, random_state=42)
```

In [ ]:
```
# Fit the model to user features and all product interactions (multi-label)
rf.fit(user_features_sample, product_features_sample)
```

In [ ]:
```
# Select important features based on feature importance
selector = SelectFromModel(rf, threshold="mean", prefit=True)

# Get the selected feature names
selected_features = product_features_sample.columns[selector.get_support()]

# Reduced dataset with selected features
# reduced_train = user_features[selected_features]
```

In [ ]:
```
print("Selected Features: ", selected_features)
```