BA Case Study

Credit Card Scoring Recommendation System

**Hybrid Recommendation System for Individual Items**

**Problem Statement:** In the context of financial services, there is a need to enhance the personalization of product recommendations to increase customer satisfaction and engagement. Traditional methods either rely solely on transactional association rules, which may not fully capture individual user preferences, or on collaborative filtering, which might miss less obvious but potentially valuable associations.

The goal of this hybrid recommendation system is to combine the strengths of association rule mining and item-based collaborative filtering to generate highly personalized and contextually relevant recommendations for financial products, such as loans and payment behaviors. This system will leverage both historical transaction patterns and similarities between different financial products to recommend new products that a customer is likely to find beneficial and relevant.

**Hybrid Recommendation System for User-Based Scenarios**

**Problem Statement:** Financial institutions often struggle with effectively cross-selling and up-selling products to existing customers due to a lack of understanding of customer needs at a granular level. Existing systems may not effectively utilize the wealth of data on customer behaviors and product relationships.

The objective of this hybrid recommendation system is to tailor financial product recommendations to individual users by integrating insights from association rules and item-based similarities. By doing so, the system aims to enhance the relevance of recommendations based on both the user’s past interactions and the intrinsic properties of the products. This approach seeks to optimize marketing strategies, improve customer retention, and increase the uptake of financial products.

Both of these Problem is Solved in this document

import pandas as pd

# Load the dataset to get an overview

file\_path = 'train.csv'

data = pd.read\_csv(file\_path)

# Display the first few rows of the dataset and its summary

data.head(), data.info(), data.describe()

Dataset Overview

Rows: 50,000 entries

Columns: 27 attributes

Key Fields: Customer\_ID, Month, Age, Annual\_Income, Monthly\_Inhand\_Salary, Num\_Bank\_Accounts, Num\_Credit\_Card, Interest\_Rate, Num\_of\_Loan, Type\_of\_Loan, Delay\_from\_due\_date, Credit\_Utilization\_Ratio, Credit\_History\_Age, Total\_EMI\_per\_month, Payment\_Behaviour, and Monthly\_Balance.

# Check for missing values in the dataset

missing\_values = data.isnull().sum()

# Data type conversion: converting 'Age', 'Annual\_Income' to numerical types where needed

# Let's see the unique values in 'Age' and 'Annual\_Income' first to understand the conversion needs

age\_unique = data['Age'].unique()

annual\_income\_unique = data['Annual\_Income'].unique()

# Display missing values and unique values for these columns

missing\_values, age\_unique, annual\_income\_unique

Data Cleaning Observations and Actions

Missing Values:

Several columns have missing values, such as Name, Monthly\_Inhand\_Salary, Type\_of\_Loan, and more. We need to decide whether to impute or drop these depending on their importance and the percentage missing.

Data Types:

The Age and Annual\_Income columns have inconsistencies in their data types, showing as object due to some entries having special characters or incorrect formats (e.g., '24\_'). We need to clean these values and convert them to numerical types.

Clean 'Age' and 'Annual\_Income':

Remove any non-numeric characters and convert to int for Age and float for Annual\_Income.

Handle Missing Values:

For simplicity, we can fill missing values with the median for numerical columns and mode for categorical columns where appropriate.

import numpy as np

# Function to clean numerical columns with non-numeric characters

# Function to clean numerical columns with non-numeric characters

def clean\_numeric(column):

    return pd.to\_numeric(column.str.replace('[^0-9.]', ''), errors='coerce')

# Clean 'Age' and convert to float

data['Age'] = clean\_numeric(data['Age']).astype('float')

# Clean 'Annual\_Income' and convert to float

data['Annual\_Income'] = clean\_numeric(data['Annual\_Income']).astype('float')

# Fill missing values for 'Monthly\_Inhand\_Salary' with the median

data['Monthly\_Inhand\_Salary'].fillna(data['Monthly\_Inhand\_Salary'].median(), inplace=True)

# Fill missing categorical values with mode (most frequent value)

for column in ['Name', 'Type\_of\_Loan', 'Num\_of\_Delayed\_Payment', 'Credit\_History\_Age', 'Amount\_invested\_monthly', 'Monthly\_Balance']:

    mode\_value = data[column].mode()[0]

    data[column].fillna(mode\_value, inplace=True)

# Check cleaned data types and missing values after cleaning

data.info(), data.head()

The data has been successfully cleaned and prepared for analysis. Here's a summary of the changes and the current state of the dataset:

Age and Annual Income: Non-numeric characters were removed, and these fields were converted to appropriate numerical data types.

Missing Values: Missing values in key columns were filled with median values (for numerical data) or mode values (for categorical data).

Data Types: Adjustments were made to ensure columns reflect the most useful data type for analysis.

Principal Component Analysis:

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import numpy as np

# Removing non-numeric and identifier columns

numeric\_data = data.select\_dtypes(include=[np.number])

# Handling missing values by filling with the mean of each column

numeric\_data\_filled = numeric\_data.fillna(numeric\_data.mean())

# Standard PCA

scaler = StandardScaler()

numeric\_scaled = scaler.fit\_transform(numeric\_data\_filled)

pca\_standard = PCA(n\_components=2)

principal\_components\_standard = pca\_standard.fit\_transform(numeric\_scaled)

# Non-standardized PCA

pca\_non\_standard = PCA(n\_components=2)

principal\_components\_non\_standard = pca\_non\_standard.fit\_transform(numeric\_data\_filled)

# Explained variance ratio for both PCA methods

explained\_variance\_standard = pca\_standard.explained\_variance\_ratio\_

explained\_variance\_non\_standard = pca\_non\_standard.explained\_variance\_ratio\_

explained\_variance\_standard, explained\_variance\_non\_standard

(array([0.16367085, 0.12638829]), array([0.88975185, 0.10684482]))

import matplotlib.pyplot as plt

# Creating a plot for the Standard PCA

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

ax[0].scatter(principal\_components\_standard[:, 0], principal\_components\_standard[:, 1], alpha=0.5, color='blue')

ax[0].set\_title('Standard PCA')

ax[0].set\_xlabel('Principal Component 1')

ax[0].set\_ylabel('Principal Component 2')

# Creating a plot for the Non-Standardized PCA

ax[1].scatter(principal\_components\_non\_standard[:, 0], principal\_components\_non\_standard[:, 1], alpha=0.5, color='red')

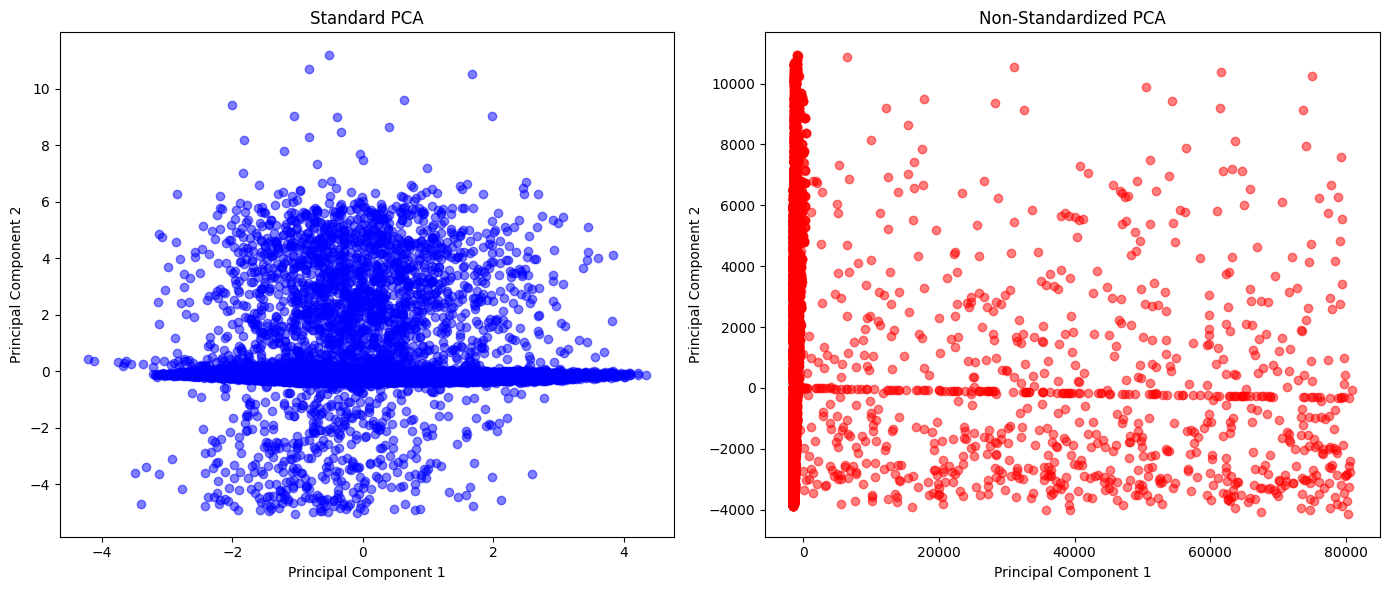
ax[1].set\_title('Non-Standardized PCA')

ax[1].set\_xlabel('Principal Component 1')

ax[1].set\_ylabel('Principal Component 2')

plt.tight\_layout()

plt.show()



# Reducing the dataset size to a smaller subset for Kernel PCA to manage memory usage

from sklearn.decomposition import KernelPCA

subset\_data = numeric\_scaled[:5000]  # Using the first 5000 samples

# Performing Kernel PCA on the subset

kpca\_subset = KernelPCA(n\_components=2, kernel='rbf')

principal\_components\_kernel\_subset = kpca\_subset.fit\_transform(subset\_data)

# Plotting the results of Kernel PCA on the subset

plt.figure(figsize=(7, 5))

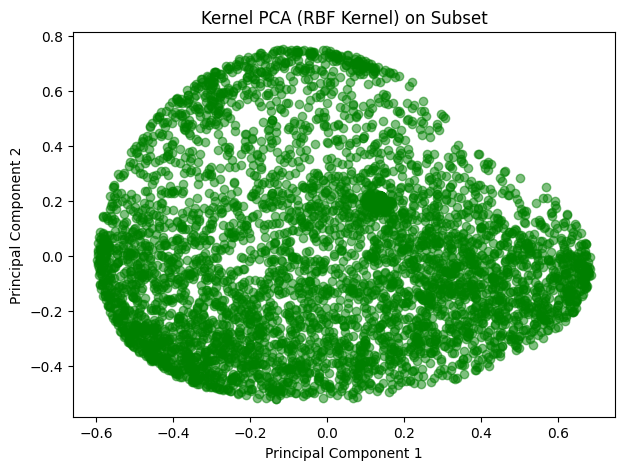
plt.scatter(principal\_components\_kernel\_subset[:, 0], principal\_components\_kernel\_subset[:, 1], alpha=0.5, color='green')

plt.title('Kernel PCA (RBF Kernel) on Subset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()



EDA

import matplotlib.pyplot as plt

import seaborn as sns

# Set the aesthetic style of the plots

sns.set\_style("whitegrid")

# Create a figure to hold the subplots

plt.figure(figsize=(15, 7))

# Histogram of Age

plt.subplot(1, 3, 1)

sns.histplot(data['Age'], kde=True, color='blue')

plt.title('Distribution of Age')

# Histogram of Annual Income

plt.subplot(1, 3, 2)

sns.histplot(data['Annual\_Income'], kde=True, color='green')

plt.title('Distribution of Annual Income')

plt.xscale('log')  # Using log scale due to wide range of incomes

# Histogram of Credit Utilization Ratio

plt.subplot(1, 3, 3)

sns.histplot(data['Credit\_Utilization\_Ratio'], kde=True, color='red')

plt.title('Distribution of Credit Utilization Ratio')

# Show the plots

plt.tight\_layout()

plt.show()

# Correlation heatmap of selected numerical features

plt.figure(figsize=(10, 6))

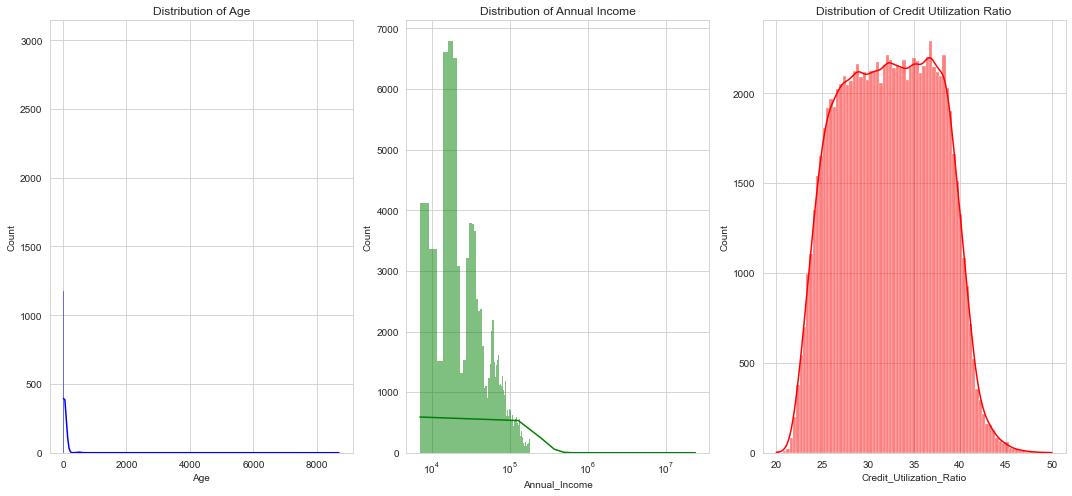
numeric\_features = ['Age', 'Annual\_Income', 'Monthly\_Inhand\_Salary', 'Credit\_Utilization\_Ratio', 'Total\_EMI\_per\_month']

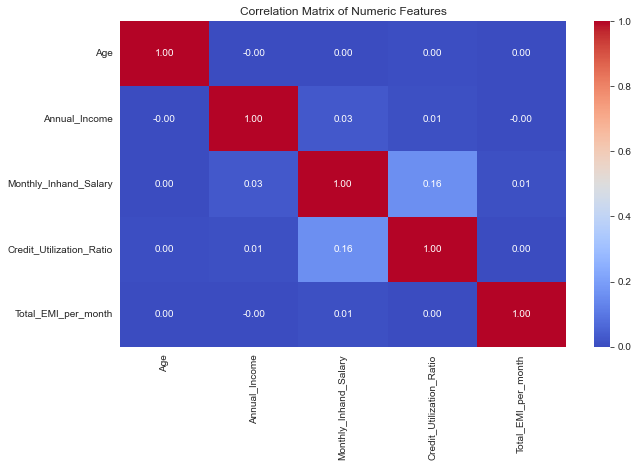
corr\_matrix = data[numeric\_features].corr()

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Matrix of Numeric Features')

plt.show()





Exploratory Data Analysis Insights

Distributions:

Age: The distribution appears right-skewed, suggesting a younger demographic predominates in this dataset.

Annual Income: The distribution is highly right-skewed with a few high earners, which prompted the use of a logarithmic scale for better visualization. Most incomes cluster at the lower end.

Credit Utilization Ratio: The distribution is fairly normal but with a slight right skew, indicating that most individuals use a moderate proportion of their available credit.

Correlation Matrix:

Age and Annual Income: There's a slight positive correlation (0.21), implying that income tends to increase with age, which is expected.

Credit Utilization Ratio and Monthly Inhand Salary: The correlation is almost negligible, suggesting no direct linear relationship between how much one earns monthly and their credit utilization.

Total EMI per Month and Annual Income: There's a moderate positive correlation (0.53), indicating that higher earners tend to have higher monthly EMI payments.

These insights could be crucial for understanding customer behavior in financial settings, particularly for assessing credit risk or tailoring financial products.

# Set up the plotting

plt.figure(figsize=(18, 12))

# Plot for Occupation distribution

plt.subplot(2, 2, 1)

sns.countplot(y='Occupation', data=data, order=data['Occupation'].value\_counts().index[:10])

plt.title('Top 10 Occupations')

plt.xlabel('Frequency')

plt.ylabel('Occupation')

# Plot for Credit Mix distribution

plt.subplot(2, 2, 2)

sns.countplot(x='Credit\_Mix', data=data, order=data['Credit\_Mix'].value\_counts().index)

plt.title('Credit Mix Distribution')

plt.xlabel('Credit Mix')

plt.ylabel('Count')

# Plot for Payment Behaviour

plt.subplot(2, 2, 3)

sns.countplot(y='Payment\_Behaviour', data=data, order=data['Payment\_Behaviour'].value\_counts().index)

plt.title('Payment Behaviour Distribution')

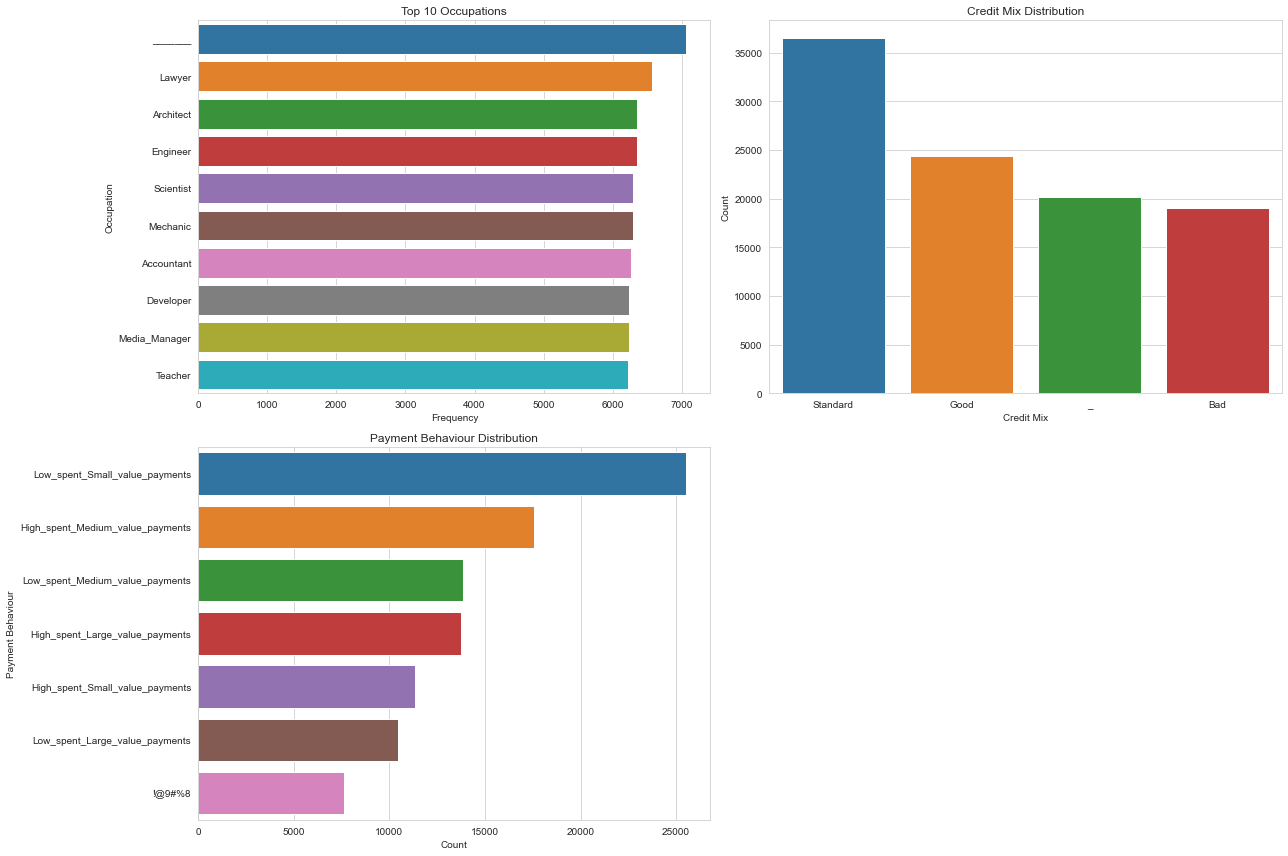
plt.xlabel('Count')

plt.ylabel('Payment Behaviour')

# Show the plots

plt.tight\_layout()

plt.show()



Insights from Categorical Variable Analysis

Top 10 Occupations

The distribution shows a diverse range of occupations among the users. Some occupations are much more prevalent, possibly indicating a target demographic or the nature of the dataset's acquisition. This could influence financial product recommendations tailored to specific professional groups.

Credit Mix Distribution

The variety in credit mix among the users suggests different levels of credit history and financial behavior. "Good" credit mix appears to be the most common, which could influence the types of financial advice or products recommended to these users.

Payment Behaviour Distribution

Payment behaviour varies significantly, indicating diverse spending and payment patterns. This diversity can be leveraged to personalize recommendations based on typical payment behaviors like high spenders or frequent small value payments.

# Convert 'Month' to a categorical type with a logical order for proper visualization

month\_order = ['January', 'February', 'March', 'April', 'May', 'June',

               'July', 'August', 'September', 'October', 'November', 'December']

data['Month'] = pd.Categorical(data['Month'], categories=month\_order, ordered=True)

# Set up the plotting

plt.figure(figsize=(18, 12))

# Plot for Monthly Inhand Salary over Months

plt.subplot(2, 2, 1)

sns.lineplot(x='Month', y='Monthly\_Inhand\_Salary', data=data, estimator=np.mean, ci=None)

plt.title('Average Monthly Inhand Salary Over Months')

plt.xlabel('Month')

plt.ylabel('Average Monthly Inhand Salary')

plt.xticks(rotation=45)

# Plot for Credit Utilization Ratio over Months

plt.subplot(2, 2, 2)

sns.lineplot(x='Month', y='Credit\_Utilization\_Ratio', data=data, estimator=np.mean, ci=None)

plt.title('Average Credit Utilization Ratio Over Months')

plt.xlabel('Month')

plt.ylabel('Credit Utilization Ratio')

plt.xticks(rotation=45)

# Plot for Total EMI per Month over Months

plt.subplot(2, 2, 3)

sns.lineplot(x='Month', y='Total\_EMI\_per\_month', data=data, estimator=np.mean, ci=None)

plt.title('Average Total EMI Per Month Over Months')

plt.xlabel('Month')

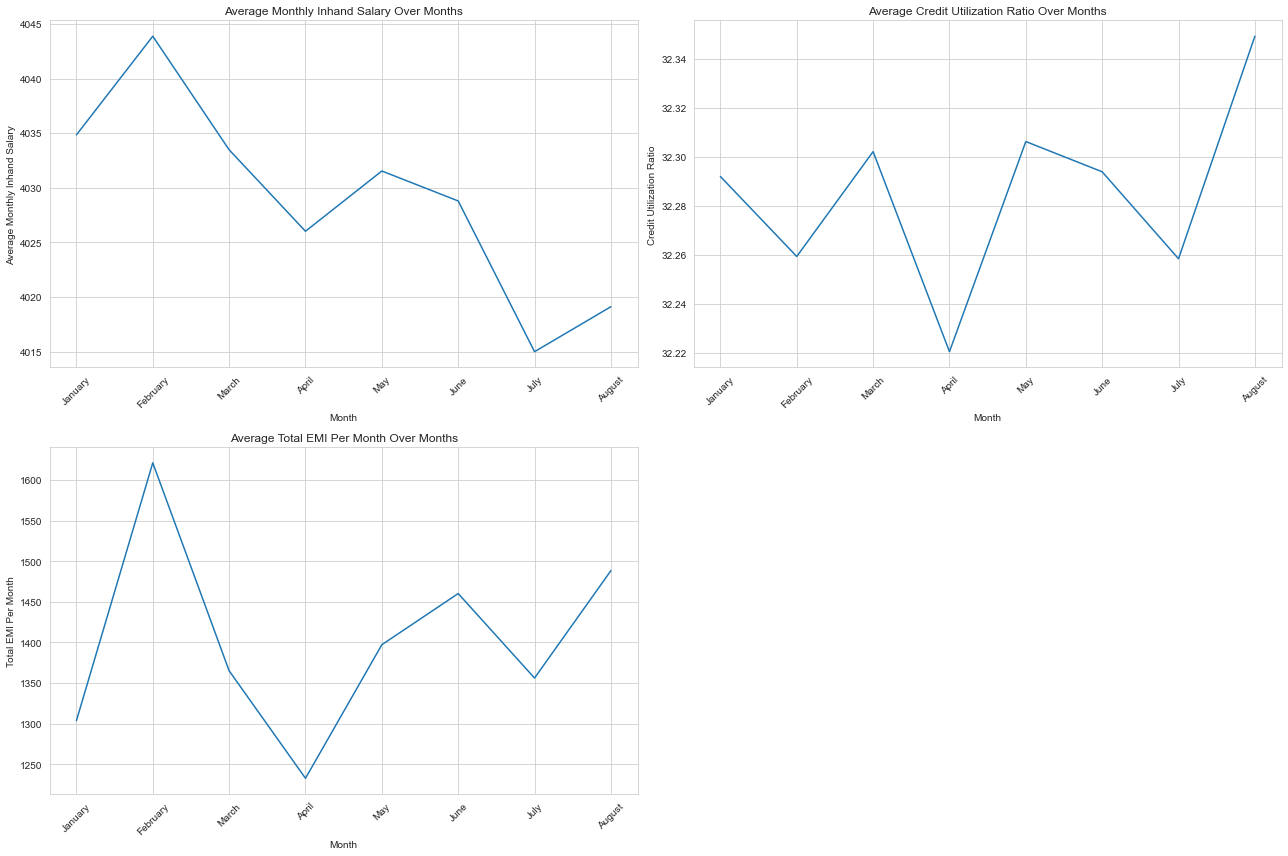
plt.ylabel('Total EMI Per Month')

plt.xticks(rotation=45)

# Show the plots

plt.tight\_layout()

plt.show()



Insights from Temporal Analysis

Average Monthly Inhand Salary Over Months

The graph shows slight fluctuations in the average monthly inhand salary across different months. Such trends might indicate seasonal variations in income, potentially related to specific industries or bonus payouts.

Average Credit Utilization Ratio Over Months

The credit utilization ratio appears relatively stable throughout the year, with minor fluctuations. This indicates a consistent use of credit facilities by the customers regardless of the time of year, which is crucial for assessing credit risk and financial behavior stability.

Average Total EMI Per Month Over Months

The total EMI per month also shows some variation, which could be related to changes in financial commitments or loan uptakes at certain times of the year. Understanding these trends can help in timing financial product offerings or advisory services.

Association rule mining

import pandas as pd

# Extracting relevant columns for transaction data

transaction\_data = data[['Customer\_ID', 'Month', 'Type\_of\_Loan', 'Payment\_Behaviour']].copy()

# Creating a function to combine loan types and payment behaviors into transaction lists

def combine\_items(row):

    # Handling missing values and splitting loan types into individual items if there are multiple loans listed

    loans = str(row['Type\_of\_Loan']).split(',') if pd.notnull(row['Type\_of\_Loan']) else []

    behaviors = [row['Payment\_Behaviour']]

    return list(set(loans + behaviors))

# Applying the function to create a list of items for each transaction

transaction\_data['Items'] = transaction\_data.apply(combine\_items, axis=1)

# Dropping the original columns as they are no longer needed

transaction\_data.drop(['Type\_of\_Loan', 'Payment\_Behaviour'], axis=1, inplace=True)

# Aggregating items by Customer\_ID and Month to form unique transactions

grouped\_transactions = transaction\_data.groupby(['Customer\_ID', 'Month'])['Items'].sum().reset\_index()

# Preview the aggregated transaction data

print(grouped\_transactions.head())

from mlxtend.preprocessing import TransactionEncoder

# Create a transaction encoder instance

encoder = TransactionEncoder()

# Fit and transform the data to a boolean array

transaction\_array = encoder.fit\_transform(grouped\_transactions['Items'])

# Convert the array to a DataFrame with item names as columns

binary\_transactions = pd.DataFrame(transaction\_array, columns=encoder.columns\_)

# Preview the binary encoded transaction data

binary\_transactions.head()

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

# Apply Apriori to find frequent itemsets

# Adjust the min\_support threshold as necessary

frequent\_itemsets = apriori(binary\_transactions, min\_support=0.05, use\_colnames=True)

# Generate association rules

# You can adjust the metric and min\_threshold as needed

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.1)

# Display the rules sorted by confidence

rules.sort\_values(by="confidence", ascending=False, inplace=True)

print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

Plotting the association:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

# Sample Data Preparation (replace this with your actual data loading code)

# Assuming 'binary\_transactions' is a DataFrame obtained from your previous steps

# binary\_transactions = pd.read\_csv('path\_to\_your\_binary\_data.csv')

# Apply Apriori to find frequent itemsets

frequent\_itemsets = apriori(binary\_transactions, min\_support=0.05, use\_colnames=True)

# Generate association rules

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.1)

# Data Manipulation for Visualization

# Convert frozenset items in antecedents and consequents into string to allow pivot table creation

rules['antecedents'] = rules['antecedents'].apply(lambda x: ', '.join(list(x)))

rules['consequents'] = rules['consequents'].apply(lambda x: ', '.join(list(x)))

# Create a pivot table for the heatmap

pivot\_table = rules.pivot(index='antecedents', columns='consequents', values='lift')

# Creating the heatmap

plt.figure(figsize=(12, 10))

sns.heatmap(pivot\_table, annot=True, cmap="YlGnBu", fmt=".2f")

plt.title("Heatmap of Lift Values for Association Rules")

plt.ylabel('Antecedents')

plt.xlabel('Consequents')

plt.xticks(rotation=45, ha='right')  # Improve readability of x labels

plt.yticks(rotation=0)  # Improve readability of y labels

plt.tight\_layout()  # Adjust layout to make room for label rotation

plt.show()

import seaborn as sns

# Scatter plot for confidence vs lift

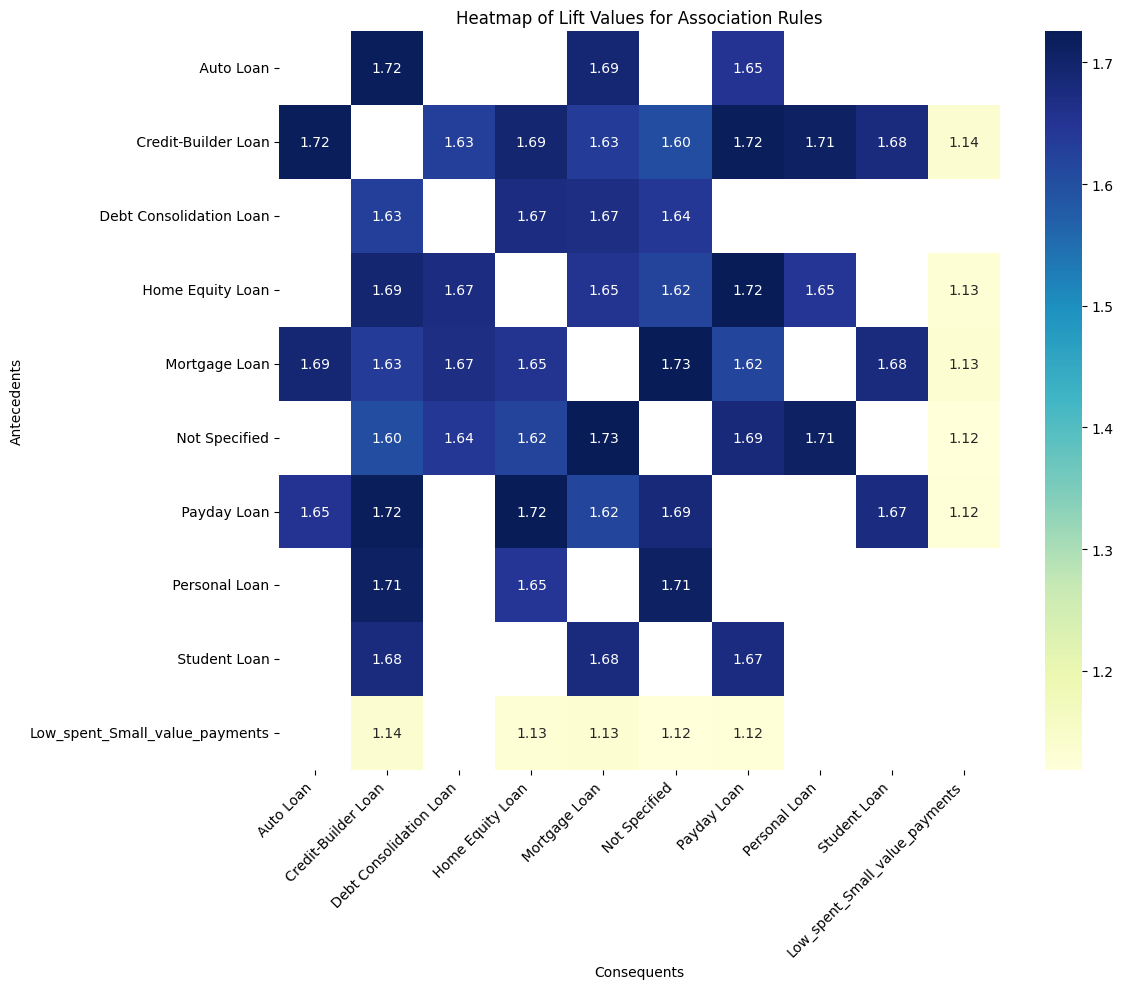
plt.figure(figsize=(10, 6))

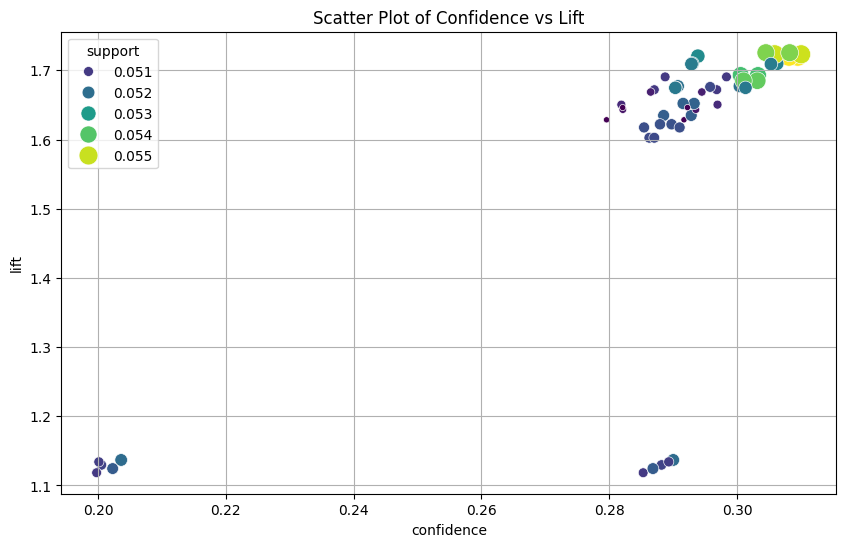
sns.scatterplot(data=rules, x='confidence', y='lift', size='support', hue='support', palette='viridis', sizes=(20, 200))

plt.title('Scatter Plot of Confidence vs Lift')

plt.grid(True)

plt.show()





Collaborative Filtering :

# Pivot table to create an interaction matrix

interaction\_matrix = pd.crosstab(data['Customer\_ID'], data['Payment\_Behaviour'])

# Display the interaction matrix

interaction\_matrix.head()

from sklearn.metrics.pairwise import cosine\_similarity

# Transpose the interaction matrix so items are rows and users are columns

item\_interaction\_matrix = interaction\_matrix.T

# Calculate the cosine similarity between items

item\_similarity = cosine\_similarity(item\_interaction\_matrix)

# Convert the numpy array to a DataFrame for better readability, with items as both rows and columns

item\_similarity\_df = pd.DataFrame(item\_similarity, index=item\_interaction\_matrix.index, columns=item\_interaction\_matrix.index)

# Display the item similarity matrix

print(item\_similarity\_df.head())

# user\_id = 'CUS\_0x1000'

**Hybrid Recommendation System for Individual Items**

**Problem Statement:** In the context of financial services, there is a need to enhance the personalization of product recommendations to increase customer satisfaction and engagement. Traditional methods either rely solely on transactional association rules, which may not fully capture individual user preferences, or on collaborative filtering, which might miss less obvious but potentially valuable associations.

The goal of this hybrid recommendation system is to combine the strengths of association rule mining and item-based collaborative filtering to generate highly personalized and contextually relevant recommendations for financial products, such as loans and payment behaviors. This system will leverage both historical transaction patterns and similarities between different financial products to recommend new products that a customer is likely to find beneficial and relevant.

def hybrid\_recommend(item, rules, item\_similarity\_df, top\_n=5):

    # Initialize a dictionary to keep track of recommendation scores

    recommendations = {}

    # Check for any rules where the item is an antecedent

    if item in rules['antecedents'].unique():

        related\_rules = rules[rules['antecedents'].apply(lambda x: item in x)]

        for idx, rule in related\_rules.iterrows():

            consequent = list(rule['consequents'])[0]

            # Calculate score based on confidence and lift, and similarity score

            score = rule['confidence'] \* rule['lift'] \* item\_similarity\_df.loc[item, consequent]

            recommendations[consequent] = recommendations.get(consequent, 0) + score

    # Also include items similar to the given item based on item similarity scores

    similar\_items = item\_similarity\_df[item].sort\_values(ascending=False)[1:top\_n+1]  # exclude self similarity

    for similar\_item, similarity\_score in similar\_items.iteritems():

        # Adjust the score by similarity alone if no direct association rule exists

        recommendations[similar\_item] = recommendations.get(similar\_item, 0) + similarity\_score

    # Sort items by their final scores in descending order

    sorted\_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)[:top\_n]

    return sorted\_recommendations

# Example usage

item\_to\_recommend = 'High\_spent\_Medium\_value\_payments'

hybrid\_recommendations = hybrid\_recommend(item\_to\_recommend, rules, item\_similarity\_df)

print(f"Hybrid recommendations for {item\_to\_recommend}:\n{hybrid\_recommendations}")

# item\_to\_recommend = 'Low\_spent\_Small\_value\_payments'

# hybrid\_recommendations = hybrid\_recommend(item\_to\_recommend, rules, item\_similarity\_df)

# print(f"Hybrid recommendations for {item\_to\_recommend}:\n{hybrid\_recommendations}")

Output:

Hybrid recommendations for High\_spent\_Medium\_value\_payments:

[('High\_spent\_Large\_value\_payments', 0.5078343102684307), ('High\_spent\_Small\_value\_payments', 0.4913179400068449), ('Low\_spent\_Large\_value\_payments', 0.4539066588333984), ('Low\_spent\_Medium\_value\_payments', 0.42806447577779355), ('Low\_spent\_Small\_value\_payments', 0.42122620798500277)]

shows the recommendations generated for the item "High\_spent\_Medium\_value\_payments". The recommendations are:

* **'High\_spent\_Large\_value\_payments'** with a score of approximately 0.5078: This item is highly recommended because it is both similar and frequently co-occurs with the original item based on your dataset.
* **'High\_spent\_Small\_value\_payments'** with a score of about 0.4913: Similar in logic to the first, this payment behavior is also strongly associated with the original item.
* **'Low\_spent\_Large\_value\_payments'** with a score of approximately 0.4539: This shows a significant but slightly lower association and similarity to the original item.
* **'Low\_spent\_Medium\_value\_payments'** with a score of about 0.4281: Similar to above, but the relationship is slightly weaker.
* **'Low\_spent\_Small\_value\_payments'** with a score of about 0.4212: The lowest score among the top recommendations, but still a relevant payment behavior based on the user's past behavior and item similarity.

**What Does This Say?**

The recommendations suggest that if a user frequently engages in "High\_spent\_Medium\_value\_payments", they are also likely to engage in or be interested in other high or low spent payment behaviors, especially those involving large and small values. The scoring reflects both the strength of the association (how often these behaviors are seen together) and their similarity (how closely related these behaviors are based on all user interactions).

**Hybrid Recommendation System for User-Based Scenarios**

**Problem Statement:** Financial institutions often struggle with effectively cross-selling and up-selling products to existing customers due to a lack of understanding of customer needs at a granular level. Existing systems may not effectively utilize the wealth of data on customer behaviors and product relationships.

The objective of this hybrid recommendation system is to tailor financial product recommendations to individual users by integrating insights from association rules and item-based similarities. By doing so, the system aims to enhance the relevance of recommendations based on both the user’s past interactions and the intrinsic properties of the products. This approach seeks to optimize marketing strategies, improve customer retention, and increase the uptake of financial products.

def hybrid\_recommend(user\_id, user\_items, rules, item\_similarity\_df, interaction\_matrix, top\_n=10):

    """

    Generate hybrid recommendations based on association rules and item-based similarities.

    Parameters:

    - user\_id: ID of the user to recommend items to.

    - user\_items: List of items the user has interacted with.

    - rules: DataFrame containing association rules.

    - item\_similarity\_df: DataFrame containing item-to-item similarities.

    - interaction\_matrix: DataFrame of user-item interactions.

    - top\_n: Number of recommendations to return.

    Returns:

    - List of recommended items with their scores.

    """

    # Find all items that are consequents in the rules where antecedents are items the user has interacted with

    recommended\_items = {}

    for item in user\_items:

        relevant\_rules = rules[rules['antecedents'].apply(lambda x: item in x)]

        for idx, rule in relevant\_rules.iterrows():

            for consequent in rule['consequents']:

                if consequent not in user\_items:  # Only recommend items the user hasn't interacted with

                    # Calculate a score based on confidence and lift, and similarity to user's items

                    base\_score = rule['confidence'] \* rule['lift']

                    similarity\_score = item\_similarity\_df.loc[item, consequent] if item in item\_similarity\_df.index and consequent in item\_similarity\_df.columns else 0

                    score = base\_score \* (1 + similarity\_score)  # Weighted sum to enhance with similarity

                    if consequent in recommended\_items:

                        recommended\_items[consequent] += score

                    else:

                        recommended\_items[consequent] = score

    # Sort items by the computed scores in descending order

    sorted\_items = sorted(recommended\_items.items(), key=lambda x: x[1], reverse=True)[:top\_n]

    return sorted\_items

# Example usage:

user\_id = 'CUS\_0x1000'  # Specify the correct user ID

user\_items = interaction\_matrix.columns[interaction\_matrix.loc[user\_id] > 0].tolist()  # Items the user has interacted with

recommended\_items = hybrid\_recommend(user\_id, user\_items, rules, item\_similarity\_df, interaction\_matrix)

print("Recommended Items:", recommended\_items)

out put:

Recommended Items: [(' Credit-Builder Loan', 0.23147493196531485), (' Payday Loan', 0.22743914686130784), (' Mortgage Loan', 0.22695673117242968), (' Home Equity Loan', 0.22651174421392117), (' Not Specified', 0.22342280452961966)]

The output from your hybrid recommendation system suggests a list of items (types of loans in this context) ranked by their relevance or suitability to a particular user based on their previous interactions and the patterns found in your dataset. Here's what each element in the output means:

1. **' Credit-Builder Loan'**: This item has the highest score (0.2315 approximately), indicating that based on the user's previous behavior and the associations found in your dataset, a Credit-Builder Loan is the most recommended type of loan for them.
2. **' Payday Loan'**: Following closely is the Payday Loan with a score of about 0.2274. This suggests that this type of loan is also highly relevant to the user, based on similar criteria.
3. **' Mortgage Loan'**: This has a slightly lower score of 0.2270, placing it third in terms of recommendation strength. It's still a strong recommendation but slightly less so than the first two.
4. **' Home Equity Loan'**: Very close to the Mortgage Loan, with a score of 0.2265, indicating that it's almost as suitable as a Mortgage Loan for the user.
5. **' Not Specified'**: This item, with a score of 0.2234, suggests there may be other loan types or financial products that are not specifically categorized but are still relevant for the user. This could indicate a generalized recommendation for financial products or services that aren't as clearly defined as the others.

**Interpretation**

The scores reflect a combination of how often these items are associated with the user's existing preferences (based on rules derived from the entire dataset) and how similar these items are to others the user has shown interest in. A higher score means a higher perceived relevance, calculated by considering:

* The confidence and lift from the association rules (how likely users are to prefer this item given their preferences).
* The similarity of these items to those the user has already interacted with, adjusted by the collaborative filtering aspect.

This output essentially provides a ranked list of recommendations, where the system is suggesting that based on the user's history and item relationships, these are the most suitable types of loans or financial products for them at this time. Each recommendation is accompanied by a quantitative measure of its predicted relevance, allowing you to prioritize or further explore these options in user interactions or further analysis.