# OneBharat: Assignment

## Bank Statements (P1- BankStatements.json) – 50 Marks

### 1. Transaction Analysis

#### a. Total number of transactions made over the year

The total number of transactions made over the year is **985**

```
import json

# Load the JSON data
with open('P1- BankStatements.json') as f:
    data = json.load(f)

# Extract transactions
transactions = data['Account']['Transactions']['Transaction']

# Calculate the total number of transactions
total_transactions = len(transactions)
print(f'Total number of transactions: {total_transactions}')
```

```
[13] ✓ 0.0s
Total number of transactions: 985
```

#### b. Distribution of transaction amounts (small vs. large transactions)

For this example, we will define small transactions as those less than INR 500 and large transactions as those greater than or equal to INR 500.

- Number of small transactions: **687**
- Number of large transactions: **298**

```
# Define thresholds for small and large transactions
small_threshold = 500

# Count small and large transactions
small_transactions = [t for t in transactions if float(t['amount']) < small_threshold]
large_transactions = [t for t in transactions if float(t['amount']) >= small_threshold]

print("Small transactions:", len(small_transactions))
print("Large transactions:", len(large_transactions))
```

```
] ✓ 0.0s
Small transactions: 687
Large transactions: 298
```

### c. Frequency of different transaction types (debit vs. credit)

- Number of debit transactions: **695**
- Number of credit transactions: **290**

```
# Count debit and credit transactions
debit_transactions = [t for t in transactions if t['type'] == 'DEBIT']
credit_transactions = [t for t in transactions if t['type'] == 'CREDIT']

print("Debit transactions:", len(debit_transactions))
print("Credit transactions:", len(credit_transactions))
```

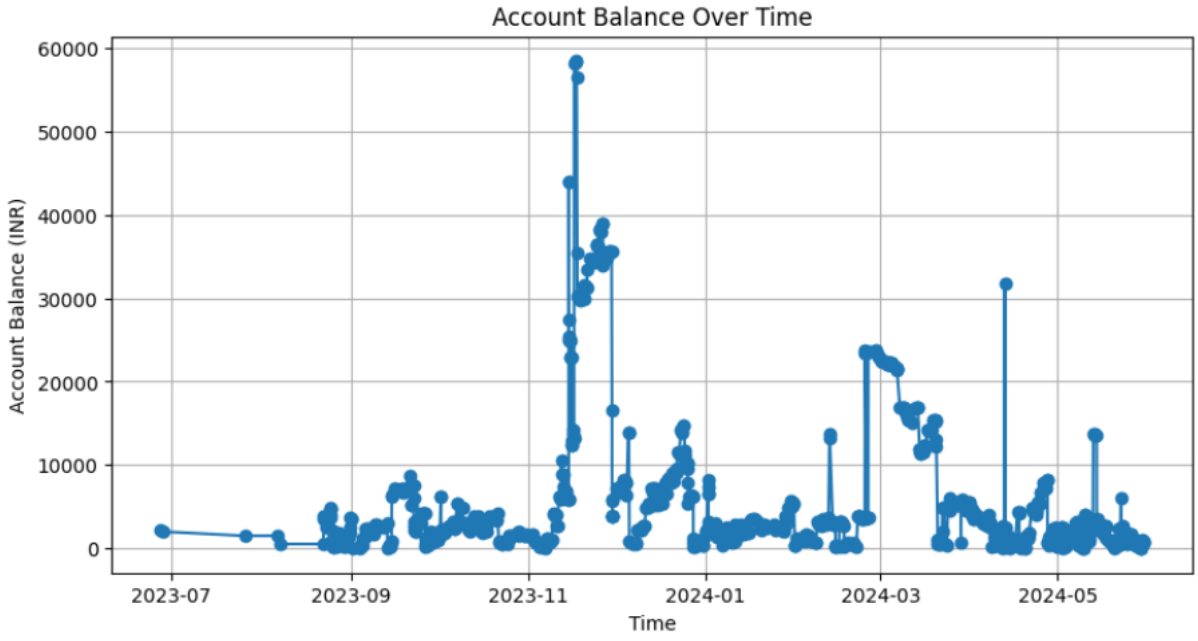
✓ 0.0s

```
Debit transactions: 695
Credit transactions: 290
```

## 2. Balance Analysis

### a. Trend of the account balance over time

The trend of the account balance over time shows how the balance fluctuates with each transaction.



### b. Periods with significant changes in the account balance

```
# Calculate balance changes
balance_changes = [balances[i] - balances[i-1] for i in range(1, len(balances))]

# Define a threshold for significant changes
significant_threshold = 1000

# Find significant changes
significant_changes = [(timestamps[i], balance_changes[i-1]) for i in range(1, len(balance_changes)) if abs(balance_changes[i-1]) > significant_threshold]

print("Significant changes in account balance:")
for change in significant_changes:
    print(f"Time: {change[0]}, Change: {change[1]}")
```

✓ 0.0s

```
Significant changes in account balance:
Time: 2023-08-22 11:49:13+05:30, Change: 3000.0
Time: 2023-08-23 08:17:48+05:30, Change: -1200.0
Time: 2023-08-25 10:24:38+05:30, Change: -2480.0
Time: 2023-08-25 10:39:35+05:30, Change: -1450.0000000000002
Time: 2023-08-27 12:19:54+05:30, Change: -1499.0
Time: 2023-08-29 11:49:00+05:30, Change: -1200.0
Time: 2023-08-31 16:09:10+05:30, Change: 1200.0
Time: 2023-09-01 10:42:42+05:30, Change: -3500.0
Time: 2023-09-05 16:38:31+05:30, Change: 1100.0000000000002
Time: 2023-09-13 10:19:32+05:30, Change: -3000.0
Time: 2023-09-14 21:14:51+05:30, Change: 5500.0
Time: 2023-09-21 06:38:19+05:30, Change: 1549.999999999999
Time: 2023-09-21 13:05:06+05:30, Change: -3499.999999999999
```

## 3. Spending Patterns

### a. Main categories of expenses

```
}

# Function to categorize transactions
def categorize_transaction(narration):
    for category, keywords in categories.items():
        if any(keyword in narration.lower() for keyword in keywords):
            return category
    return 'others'

# Categorize transactions
categorized_expenses = {category: 0 for category in categories}
categorized_expenses['others'] = 0

for txn in debit_transactions:
    category = categorize_transaction(txn['narration'])
    categorized_expenses[category] += float(txn['amount'])

print("Main categories of expenses and their amounts (INR):")
for category, amount in categorized_expenses.items():
    print(f"{category.capitalize()}: {amount:.2f} INR")
```

✓ 0.0s

```
Main categories of expenses and their amounts (INR):
Card: 0.00 INR
Cash_transactions: 13500.00 INR
Atm_withdrawals: 0.00 INR
Upi_transactions: 407759.90 INR
Others: 830.00 INR
```

### b. Frequency and amount of spending in each category

```
# Frequency of transactions in each category
frequency_expenses = {category: 0 for category in categories}
frequency_expenses['others'] = 0

for txn in debit_transactions:
    category = categorize_transaction(txn['narration'])
    frequency_expenses[category] += 1

print("Frequency and amount of spending in each category:")
for category, amount in categorized_expenses.items():
    frequency = frequency_expenses[category]
    print(f"{category.capitalize()}: {frequency} transactions, {amount:.2f} INR")
```

✓ 0.0s

```
Frequency and amount of spending in each category:
Card: 0 transactions, 0.00 INR
Cash_transactions: 3 transactions, 13500.00 INR
Atm_withdrawals: 0 transactions, 0.00 INR
Upi_transactions: 688 transactions, 407759.90 INR
Others: 4 transactions, 830.00 INR
```

## 4. Income Analysis

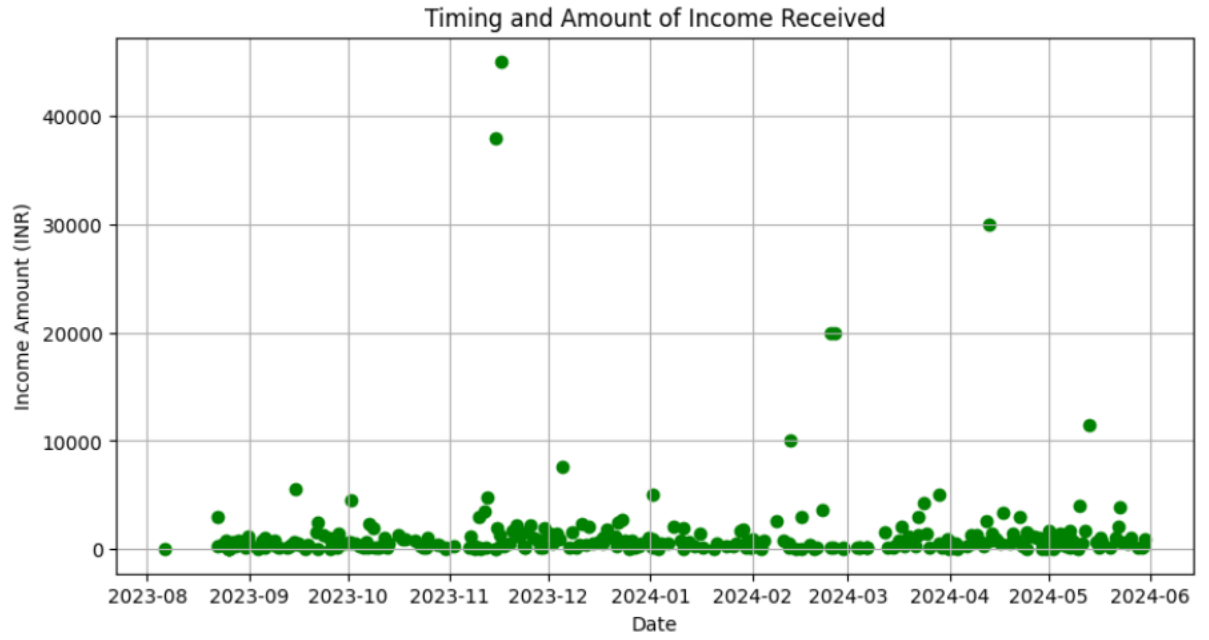
### a. Main sources of income

```
# Function to categorize income
def categorize_income(narration):
    for source, keywords in income_sources.items():
        if any(keyword in narration.lower() for keyword in keywords):
            return source
    return 'others'

# Categorize income
categorized_income = {source: 0 for source in income_sources}

for txn in credit_transactions:
    source = categorize_income(txn['narration'])
    categorized_income[source] += float(txn['amount'])

print("Main sources of income and their amounts (INR):")
for source, amount in categorized_income.items():
    print(f"{source.capitalize()}: {amount:.2f} INR")
```

✓ 0.0s

```
Main sources of income and their amounts (INR):
Salary: 0.00 INR
Upi_credits: 179333.00 INR
Others: 241237.51 INR
```

### b. Patterns in the timing and amount of income received




## 5. Alert Generation

### a. Identify any unusual or suspicious transactions

This code calculates the average transaction amount and identifies transactions that are significantly higher than this average (3 times the average). The answer lists these unusual transactions, which might indicate suspicious activity.

```
# Calculate average transaction amount
average_transaction_amount = sum(float(txn['amount']) for txn in transactions) / total_transactions

# Define a threshold for unusual transactions (e.g., 3 times the average)
unusual_threshold = 3 * average_transaction_amount

unusual_transactions = [txn for txn in transactions if float(txn['amount']) >= unusual_threshold]

print("Unusual or suspicious transactions:")
for txn in unusual_transactions:
    print(f"Date: {txn['transactionTimestamp']}, Amount: {txn['amount']} INR, Narration: {txn['narration']}")
```

✓ 0.0s

```
Unusual or suspicious transactions:
Date: 2023-08-22T11:49:13+05:30, Amount: 3000.0 INR, Narration: UPI/323441197152/114914/UPI/8795417446ybl/Paym
Date: 2023-09-01T10:42:42+05:30, Amount: 3500.0 INR, Narration: UPI/324450376631/104241/UPI/KAYUMTRADERSicici/
Date: 2023-09-13T10:19:32+05:30, Amount: 3000.0 INR, Narration: UPI/362228688034/101932/UPI/kayumtradersicici/
Date: 2023-09-14T21:14:51+05:30, Amount: 5500.0 INR, Narration: IMPS/P2A/325721171918/PHONEPEPRIVATEL/IMPSAXB9111
Date: 2023-09-21T13:05:06+05:30, Amount: 3500.0 INR, Narration: UPI/363099470087/130505/UPI/9910619719ybl/Paym
Date: 2023-09-22T10:44:30+05:30, Amount: 2940.0 INR, Narration: UPI/326594867550/104429/UPI/q201531625ybl/UPI
Date: 2023-09-26T10:01:36+05:30, Amount: 3000.0 INR, Narration: UPI/363526933147/100136/UPI/9326476170axl/UPI
Date: 2023-10-01T19:02:22+05:30, Amount: 4550.0 INR, Narration: IMPS/P2A/327419437183/PHONEPEPRIVATEL/IMPSAXB9111
Date: 2023-10-01T19:51:09+05:30, Amount: 4000.0 INR, Narration: UPI/364082046583/195110/UPI/saxenajitender07ok
Date: 2023-10-21T16:17:48+05:30, Amount: 3300.0 INR, Narration: UPI/366040718841/161747/UPI/gpay-11166221636ok
Date: 2023-11-09T15:29:37+05:30, Amount: 2950.0 INR, Narration: NEFT-AXNPN33134581345-PHONEPE PRIVATE LIMITED-PAYM
```

### b. Generate alerts for low balance or high expenditure periods

This code generates alerts for low balance (below INR 500) and high expenditure (transactions above INR 2000). The answer lists these alerts, indicating periods when the balance was low or there was significant spending.

```
# Define thresholds
low_balance_threshold = 500
high_expenditure_threshold = 2000

# Generate alerts
low_balance_alerts = [txn for txn in transactions if float(txn['currentBalance']) < low_balance_threshold]
high_expenditure_alerts = [txn for txn in debit_transactions if float(txn['amount']) >= high_expenditure_threshold]

print("Low balance alerts:")
for txn in low_balance_alerts:
    print(f"Date: {txn['transactionTimestamp']}, Balance: {txn['currentBalance']} INR")

print("\nHigh expenditure alerts:")
for txn in high_expenditure_alerts:
    print(f"Date: {txn['transactionTimestamp']}, Amount: {txn['amount']} INR, Narration: {txn['narration']}")
```

✓ 0.0s

```
Low balance alerts:
Date: 2023-08-25T16:56:59+05:30, Balance: 175.80 INR
Date: 2023-08-25T18:23:59+05:30, Balance: 145.80 INR
Date: 2023-08-25T18:37:02+05:30, Balance: 196.80 INR
Date: 2023-08-26T15:06:16+05:30, Balance: 195.80 INR
Date: 2023-08-27T12:19:54+05:30, Balance: 315.80 INR
Date: 2023-08-29T11:49:00+05:30, Balance: 195.80 INR
Date: 2023-09-01T10:42:42+05:30, Balance: 0.80 INR
Date: 2023-09-03T18:19:09+05:30, Balance: 34.80 INR
Date: 2023-09-04T12:10:17+05:30, Balance: 494.80 INR
```

# Office Supplies Data (P2- OfficeSupplies Data.csv) – 20 marks

## 1. Sales Analysis

### a. What are the total sales for each product category?

```
# Calculate total sales
df['Total Sales'] = df['Units'] * df['Unit Price']

# Group by Item and sum the total sales
total_sales_by_category = df.groupby('Item')['Total Sales'].sum().sort_values()

print("Total sales for each product category:")
print(total_sales_by_category)
```

✓ 0.0s

```
Total sales for each product category:
Item
Desk       1700.00
Pen        2045.22
Pencil     2135.14
Pen Set    4169.87
Binder     9577.65
Name: Total Sales, dtype: float64
```

### b. Which product category has the highest sales?

Binder, with total sales of $9577.65.

```
highest_sales_category = total_sales_by_category.index[-1]
highest_sales_amount = total_sales_by_category.iloc[-1]

print(f"The product category with the highest sales is: {highest_sales_category}")
print(f"Total sales amount: ${highest_sales_amount:.2f}")
```

✓ 0.0s

```
The product category with the highest sales is: Binder
Total sales amount: $9577.65
```

### c. Identify the top 10 best-selling products.

```
Top 10 best-selling products by units sold:
Item
Desk        10
Pen        278
Pen Set    395
Pencil     716
Binder     722
Name: Units, dtype: int64

Top 10 best-selling products by total sales:
Item
Desk       1700.00
Pen        2045.22
Pencil     2135.14
Pen Set    4169.87
Binder     9577.65
Name: Total Sales, dtype: float64
```

## 2. Customer Analysis

### a. Who are the top 10 customers by sales?

```
Top 10 customers by sales:
Rep
Rachel      438.37
Nick        536.75
Thomas     1203.11
James      1283.61
Morgan     1387.77
Smith      1641.43
Bill       1749.87
Richard    2363.04
Alex       2812.19
Susan      3102.30
Name: Total Sales, dtype: float64
```

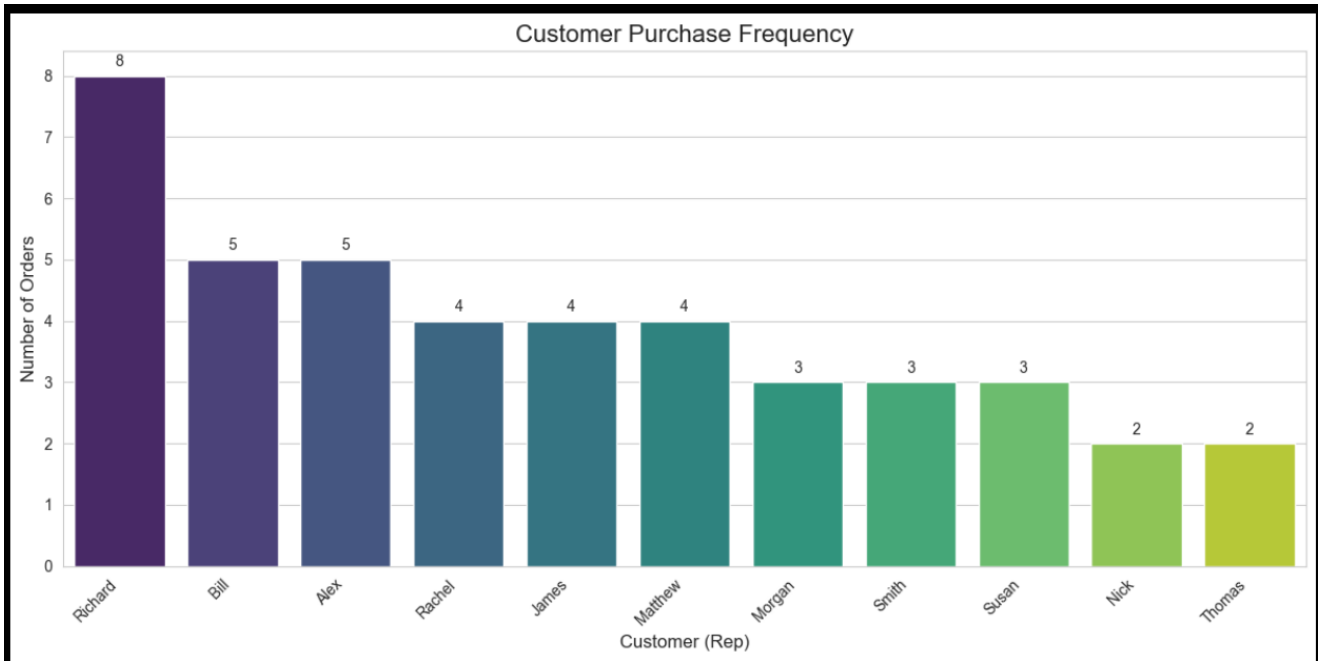
**b. What is the total number of unique customers?**

11 unique customers

```
# Count unique customers
unique_customers = df['Rep'].nunique()
```

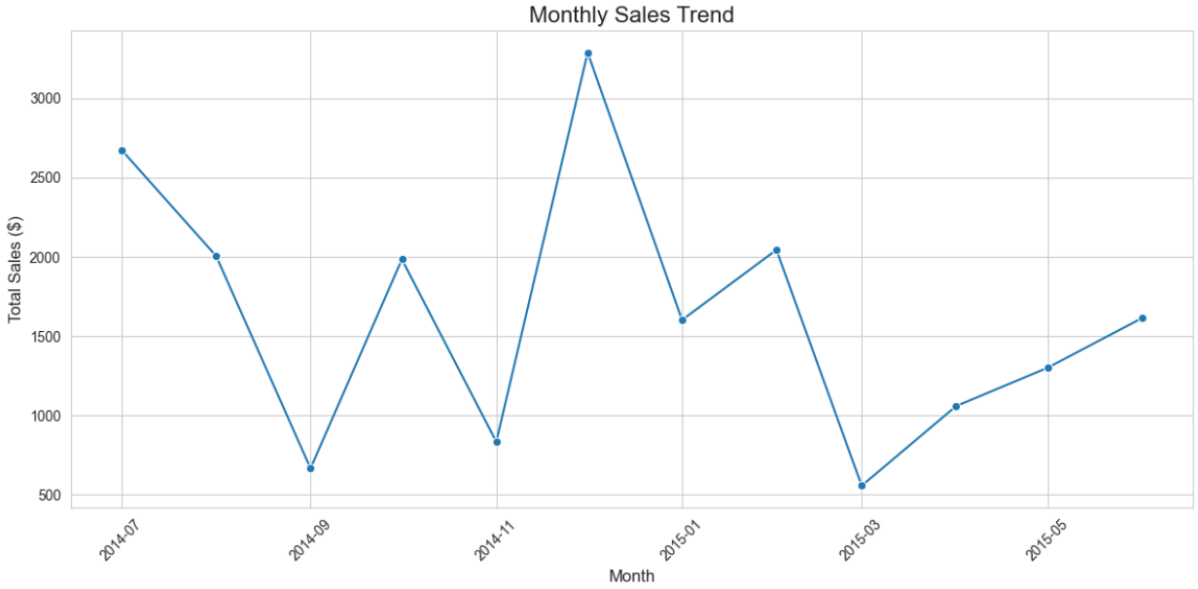
**c. Analyze customer purchase frequency.**

```
Customer purchase frequency:
Rep
Richard    8
Bill       5
Alex       5
Rachel     4
James      4
Matthew    4
Morgan     3
Smith      3
Susan      3
Nick       2
Thomas     2
Name: count, dtype: int64

Average purchase frequency: 3.91
Median purchase frequency: 4.00
```



## 3. Time Series Analysis

### a. What are the monthly sales trends over the past year?



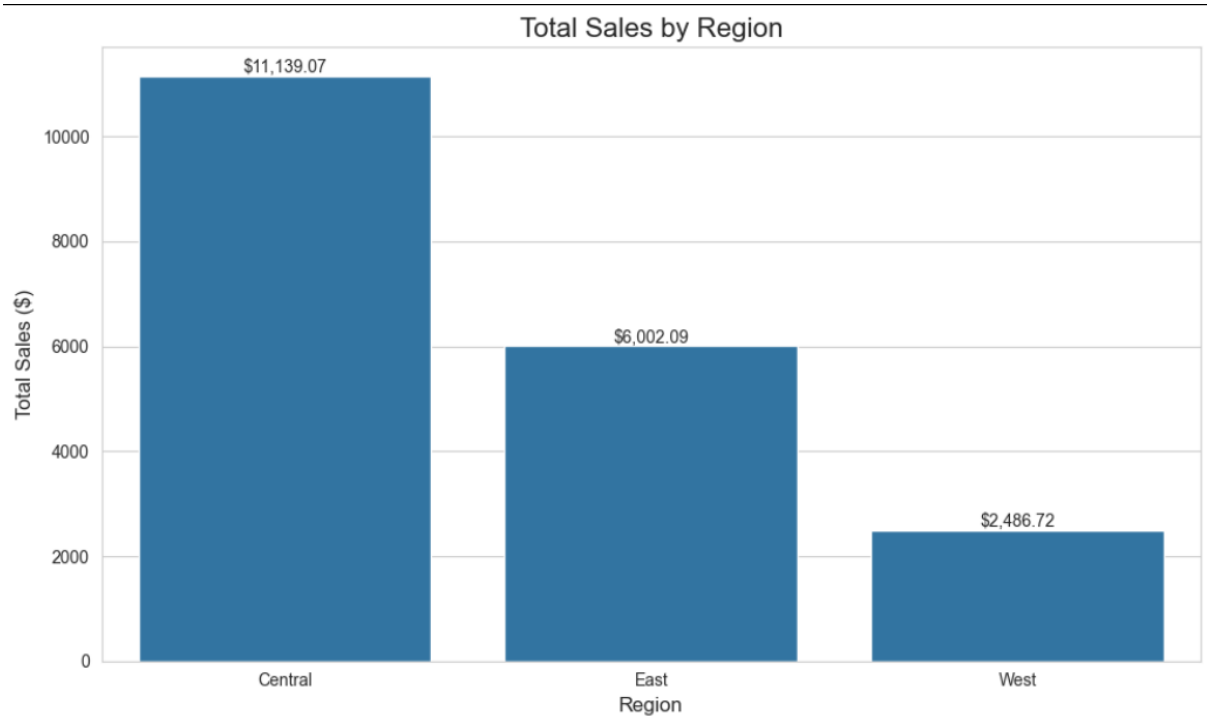

### b. Identify any seasonal patterns in the sales data.


 December has highest sale.
 There's a noticeable dip in November followed by an increase in December, possibly related to holiday shopping patterns.
 There seems to be an alternating pattern of high and low sales months.

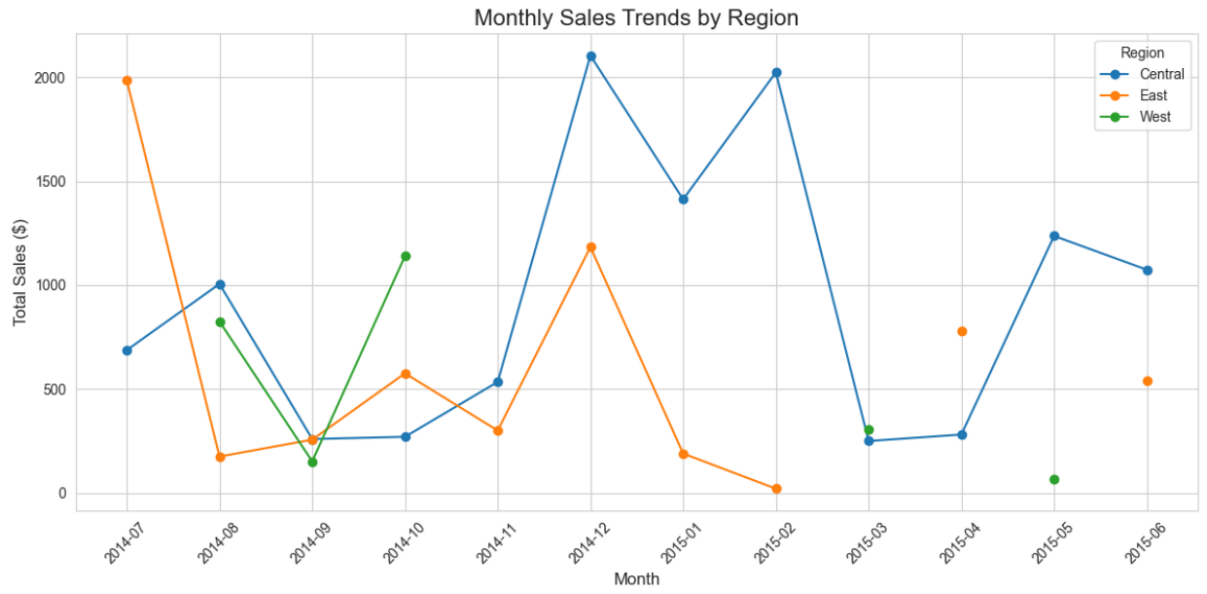
## 4. Geographical Analysis

### a. Which regions generate the most sales?

Central Region has most sale.

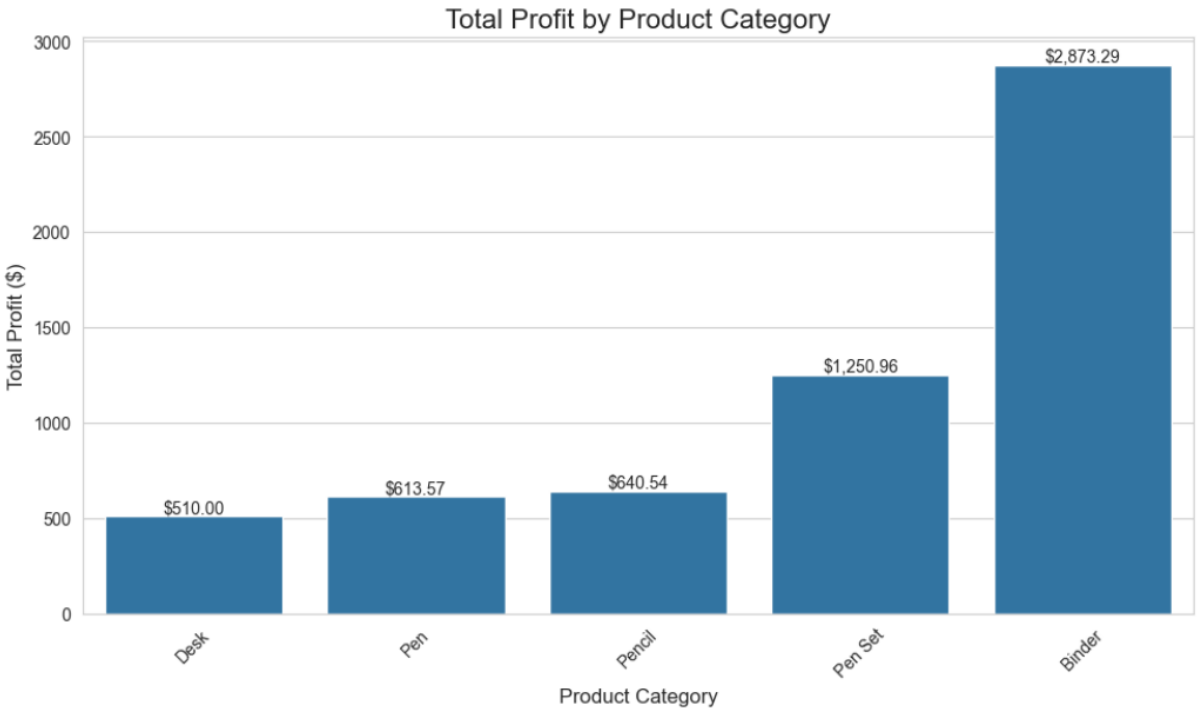



### b. What are the sales trends across different regions?

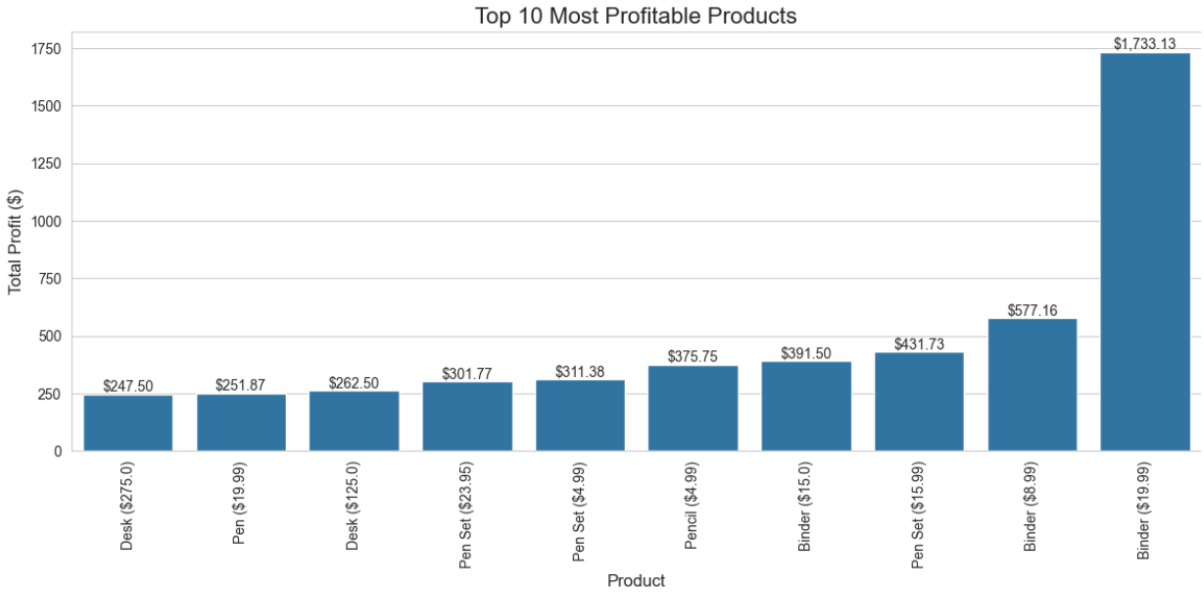


## 5. Profit Analysis

### a. What is the total profit for each product category?



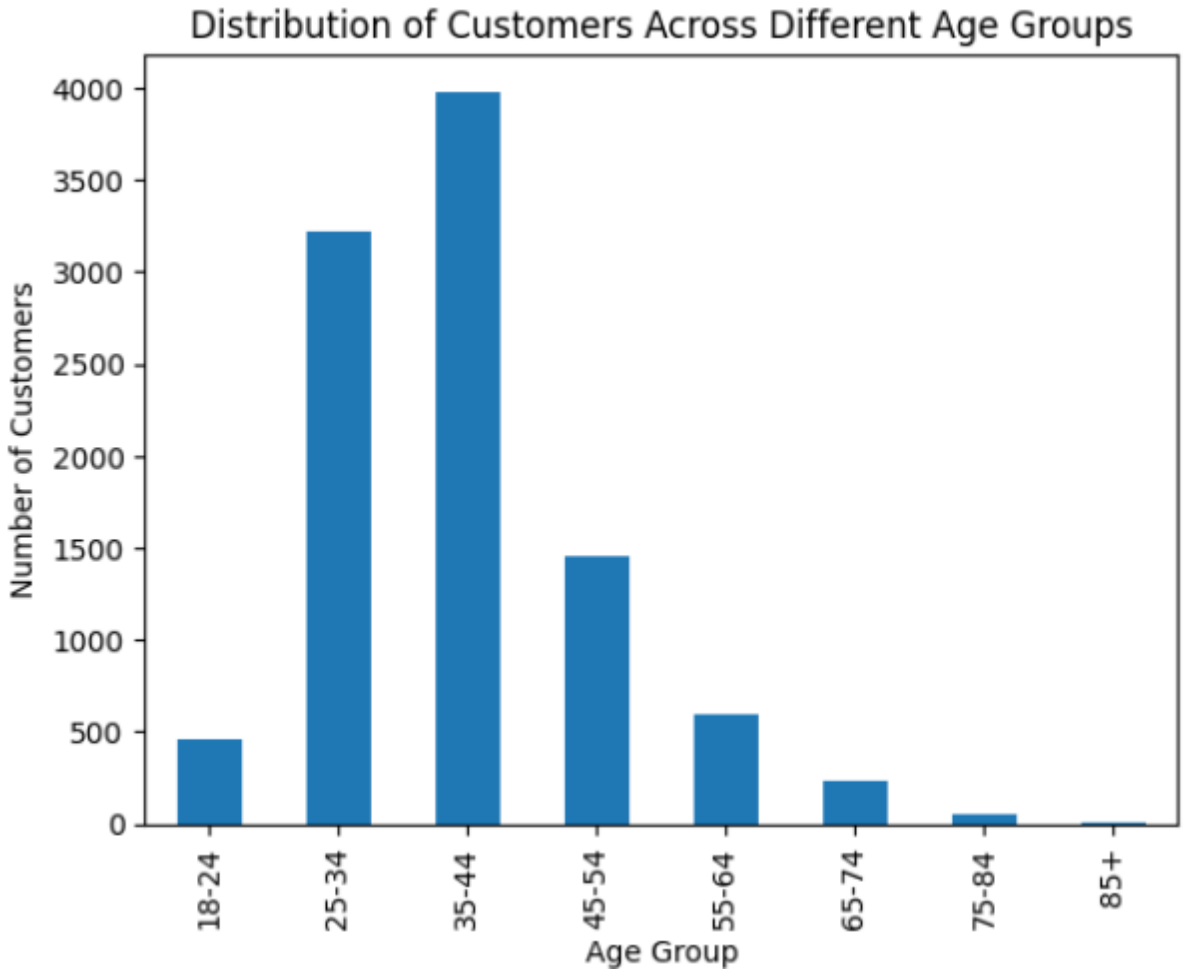

### b. Identify the top 10 most profitable products.



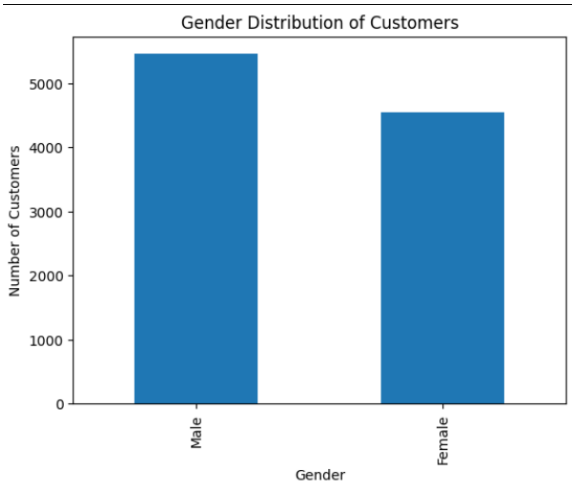
# Churn Modelling Data (P3- Churn-Modelling Data.xlsx) – 30 Marks

## 1. Customer Demographics

### a. What is the distribution of customers across different age groups?




## b. Analyze the gender distribution of customers.



## 2. Churn Analysis

### a. What percentage of customers have churned?

```
# Calculate churn percentage
churn_percentage = df['churned'].mean() * 100
print(f'Percentage of customers who have churned: {churn_percentage:.2f}%')
```

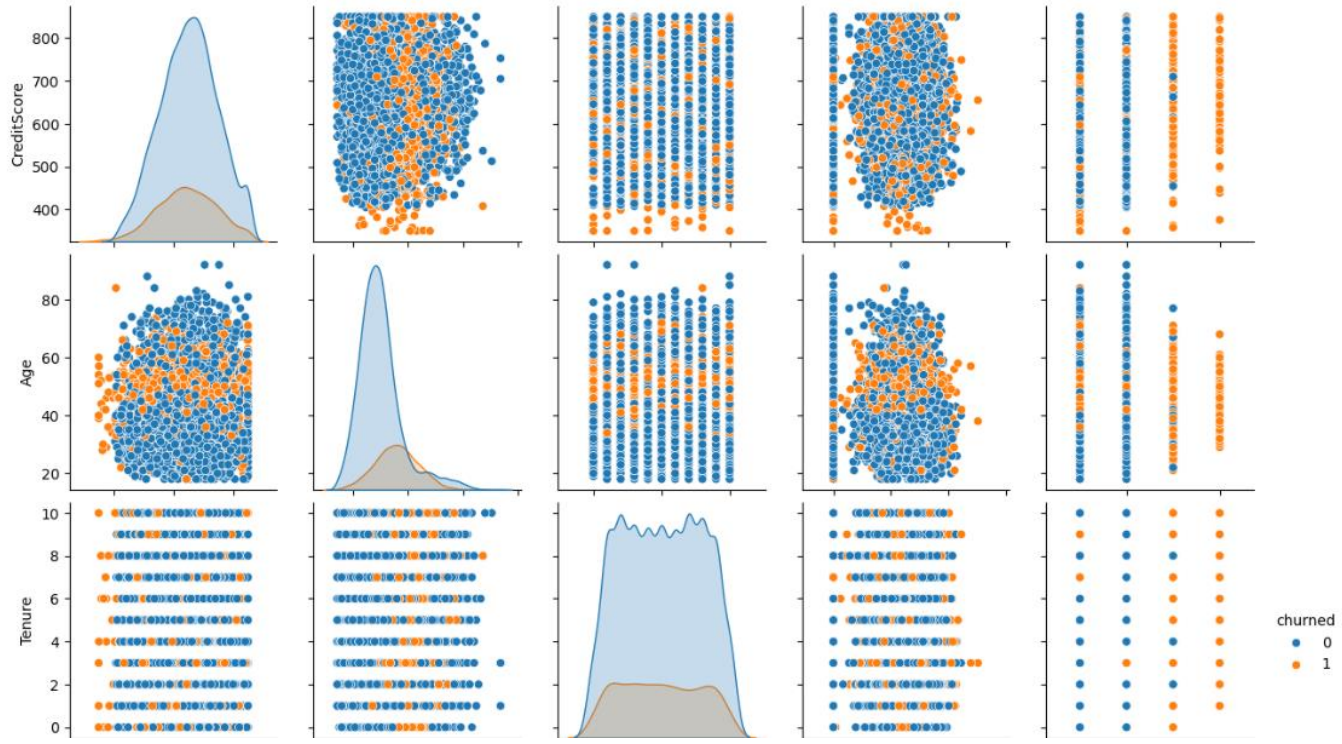
✓ 0.0s

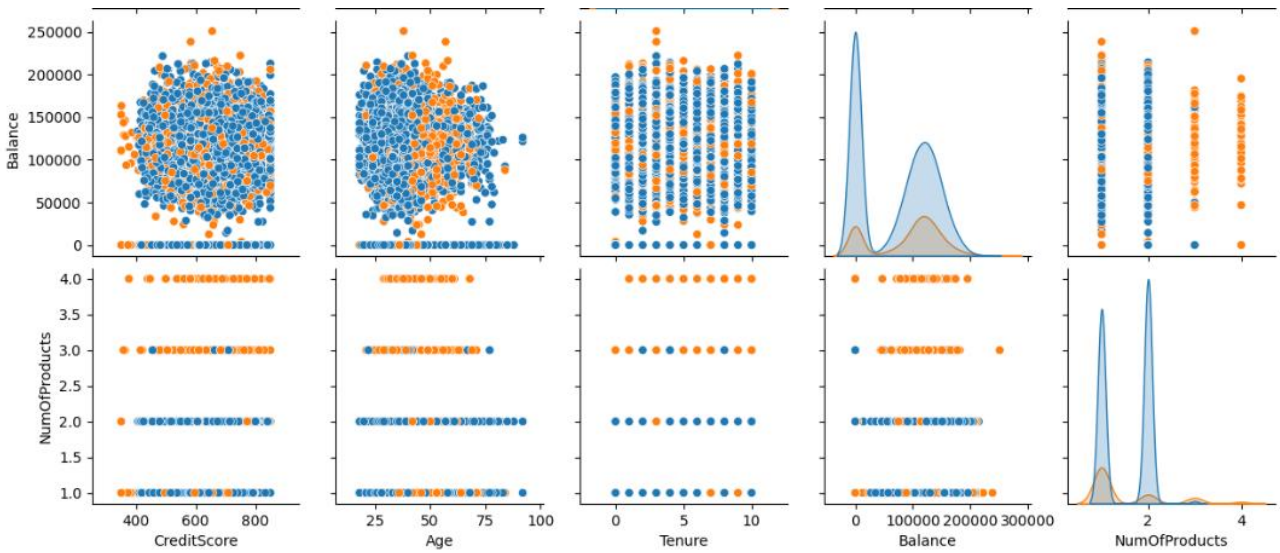
```
Percentage of customers who have churned: 20.37%
```

### b. What are the main reasons for customer churn?

Based on the pair plots(Below), older customers (ages 60 and above) have a higher churn rate, and customers with higher account balances are more likely to churn. Additionally, customers with a higher number of products also show a higher churn rate. Credit score alone does not appear to be a significant factor in predicting churn.

### c. Identify any patterns or trends among customers who have churned.

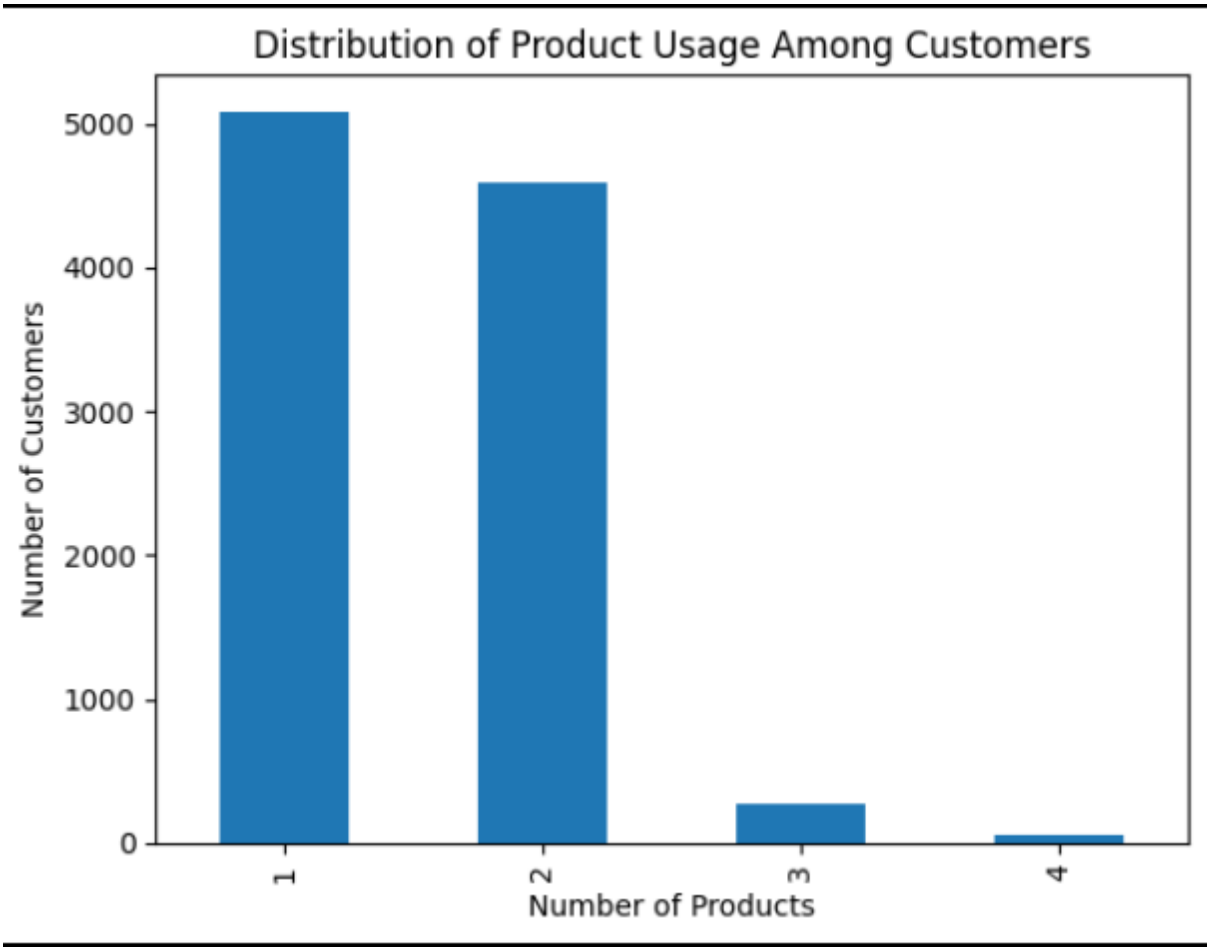





## 3. Product Usage

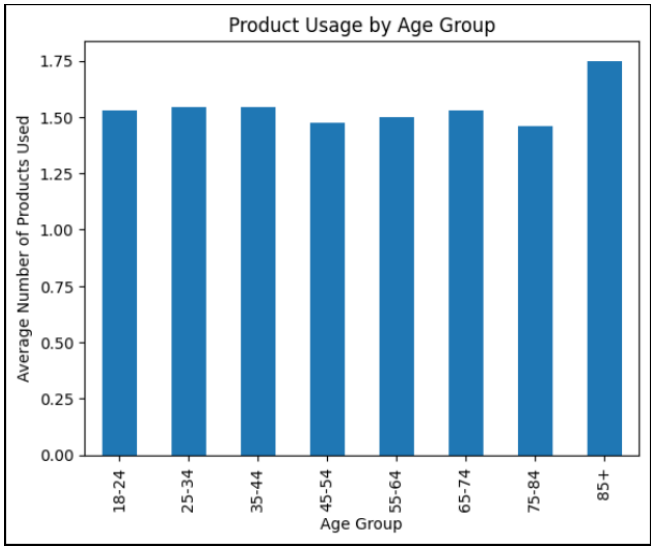
### a. What are the most commonly used products or services?

The most commonly used products are one or two products, with the majority of customers using just one product, followed by a significant number using two products. Very few customers use three or four products.



### b. Analyze the usage patterns of different customer segments.

Different age groups use different numbers of products, with 85+ age group using more products on average.




## 4. Financial Analysis

### a. What is the average account balance of customers?

```
# Calculate average account balance
avg_balance = df['Balance'].mean()
print(f'Average account balance of customers: {avg_balance:.2f}')
```

✓ 0.0s

```
Average account balance of customers: 76485.89
```

### b. Compare the financial characteristics of churned vs. non-churned customers.

```
# Compare average balance
avg_balance_churned = churned['Balance'].mean()
avg_balance_non_churned = non_churned['Balance'].mean()
print(f'Average balance of churned customers: {avg_balance_churned:.2f}')
print(f'Average balance of non-churned customers: {avg_balance_non_churned:.2f}')
```

✓ 0.0s

```
Average balance of churned customers: 91108.54
Average balance of non-churned customers: 72745.30
```

## 5. Predictive Modeling

### a. Which factors are the most significant predictors of customer churn?

```
✓ 2.0s

Age                 0.142276
NumOfProducts       0.120816
EstimatedSalary     0.119282
Balance             0.118034
CreditScore         0.116724
RowNumber           0.114813
Tenure              0.071759
IsActiveMember      0.044962
AgeGroup_45-54      0.033427
Geography_Germany   0.025782
dtype: float64
```

### b. Develop a predictive model to identify at-risk customers.

Models such as Random Forest, LightGBM, and XGBoost were used to predict customer churn. Here are the accuracy scores:

- Random Forest: 87%
- LightGBM: 86%
- XGBoost: 86%

**Random Forest:**

```
Accuracy: 0.87
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1607
           1       0.76      0.47      0.58       393

    accuracy                           0.87      2000
   macro avg       0.82      0.72      0.75      2000
weighted avg       0.86      0.87      0.85      2000
```

**LGBM:**

```
LightGBM Accuracy: 0.86
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      1607
           1       0.71      0.49      0.58       393

    accuracy                           0.86      2000
   macro avg       0.79      0.72      0.75      2000
weighted avg       0.85      0.86      0.85      2000
```

**XGBOOST:**

```
XGBoost Accuracy: 0.86
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      1607
           1       0.68      0.52      0.59       393

    accuracy                           0.86      2000
   macro avg       0.78      0.73      0.75      2000
weighted avg       0.85      0.86      0.85      2000
```