RevoU Bank Credit Card Products

By Ligasyah Arnanda Primadana
Batch FEB25

List of Content

1. The Business
2. The Code

Business Context

The importance of excluding customers under the age of 21 from eligibility for the RevoU credit card product. This decision is rooted in both legal compliance and the objective of promoting responsible lending practices. By setting this age limit, the company aims to ensure that applicants demonstrate a level of financial maturity and responsibility, which in turn helps to mitigate the risk of defaults. This strategic approach not only enhances the stability of the credit portfolio but also aligns with RevoU's goal of fostering a reliable customer base and supporting sustainable growth in its financial services.

Disclaimer: The data and context presented herein are fictional and intended solely for educational purposes.

1. Business Objective

1. Increase Credit Card Usage: Encourage existing customers to use RevoBank's credit card products more frequently.
2. Summarize Sales Performance: Analyze and report on sales data from the past
3. Develop User Personas: Create customer profiles to better understand and target client segments.
4. Enable Data-Driven Decisions: Use data analytics to guide strategic business decisions.

Here the google collab :

The Code

Importing Libraries and Data

```
import pandas as pd
import gdown
import numpy as np
from datetime import datetime

!pip install gdown
```

Installing gdown for using database from drive. To prepare for data cleaning in Python, you need to install Pandas for data manipulation, NumPy for numerical operations, and utilize the built-in datetime module for handling date and time data, as it is included in Python's standard library.

Importing Libraries and Data

```
# load sales_data.csv as df1
sales_url = 'https://drive.google.com/file/d/1dXE7wrsL37LhWl962gqluv6ZOuEnn_zG/view'
customers = 'https://drive.google.com/uc?id='+ sales_url.split('/')[-2]
df1 = pd.read_csv(customers)
df1.head()
```

The code snippet demonstrates how to load a CSV file from Google Drive into a Pandas DataFrame by constructing a direct access URL from the file's ID and using the pd.read_csv() function to read the data.

Reviewing Dataset

```
# Users dataset Overview for Understanding
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12559 entries, 0 to 12558
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   account_id               12559 non-null  int64
 1   account_activity_level   12559 non-null  object
 2   customer_value_level     12559 non-null  object
 3   MOB                      12559 non-null  float64
 4   flag_female              12559 non-null  int64
 5   avg_sales_L36M           11820 non-null  float64
 6   cnt_sales_L36M           12559 non-null  int64
 7   last_sales               12559 non-null  float64
 8   month_since_last_sales   12559 non-null  int64
 9   count_direct_promo_L12M  12559 non-null  int64
 10  birth_date               12559 non-null  object
dtypes: float64(3), int64(5), object(3)
memory usage: 1.1+ MB
```

The output from df1.info() provides a comprehensive overview of the dataset, indicating that it contains 12,559 entries across 11 columns, with details on the non-null counts and data types for each column, which include integers, floats, and objects.

Copying dataset before data cleaning

```
# Make a copy of the original data for before data cleaning
df_customers_dc = df1.copy()
```

The code snippet creates a copy of the original DataFrame df1 and assigns it to df_customers_dc, ensuring that the original data remains unchanged for reference before any data cleaning processes are applied.

Change Dataset

```
# Change account_id data type into string
df_customers_dc['account_id'] = df_customers_dc['account_id'].astype(str)

# Change MOB data type into int
df_customers_dc[['MOB']] = df_customers_dc[['MOB']].astype(int)

# Change date data type into date
df_customers_dc['birth_date'] = pd.to_datetime(df_customers_dc['birth_date'])
```

The code snippet demonstrates the process of changing the data types of specific columns in the DataFrame df_customers_dc, converting the account_id to a string, the MOB column to an integer, and the birth_date column to a datetime format for accurate data representation and analysis.

Check data type

```
#Check the data type of id after converted
df_customers_dc.dtypes
```

	0
account_id	object
account_activity_level	object
customer_value_level	object
MOB	int64
flag_female	int64
avg_sales_L36M	float64
cnt_sales_L36M	int64
last_sales	float64
month_since_last_sales	int64
count_direct_promo_L12M	int64
birth_date	datetime64[ns]

dtype: object

The output displays the data types of each column in the DataFrame df_customers_dc after conversion, confirming that account_id is now an object, MOB is an integer, and birth_date has been successfully converted to a datetime format, along with other columns retaining their respective types.

Checking Unique Value

```
# Check unique data in categorical columns
for column in df_customers_dc.select_dtypes(include=['object']).columns:
  num_unique = df_customers_dc[column].nunique()
  print(f"Number of unique '{column}' is {num_unique}")
```

```
Number of unique 'account_id' is 12487
Number of unique 'account_activity_level' is 4
Number of unique 'customer_value_level' is 6
```

The code snippet iterates through the categorical columns of the DataFrame df_customers_dc to count and display the number of unique values in each column, revealing that there are 12,487 unique account_ids, 4 unique account_activity_levels, and 6 unique customer_value_levels.

Checking Unique Value

```
# Count the values
list_obj = ['account_activity_level','customer_value_level']

for i in list_obj:
    unique_data = df_customers_dc[i].value_counts()
    print(unique_data.to_markdown(), '\n\n')
```

account_activity_level	count
X	8238
Z	3442
Y	878
XYZ	1

customer_value_level	count
E	4480
B	2390
A	2204
C	2003
D	1481
F	1

The code snippet counts and displays the occurrences of each unique value in the account_activity_level and customer_value_level columns of the DataFrame df_customers_dc, revealing the distribution of activity levels and customer value categories among the clients.

Excluding XYZ

```
#Excluding XYZ
df_customers_dc = df_customers_dc[df_customers_dc['account_activity_level']!='XYZ']
```

```
# To confirm the changes after fixing the typos
df_customers_dc['account_activity_level'].value_counts().reset_index()
```

	account_activity_level	count
0	X	8238
1	Z	3442
2	Y	878

The code snippet filters the DataFrame df_customers_dc to exclude entries with the account_activity_level value of 'XYZ' and then confirms the changes by displaying the updated counts of the remaining unique values in that column.

Copying dataset before data cleaning

```
#Excluding F
df_customers_dc = df_customers_dc[df_customers_dc['customer_value_level']!='F']
```

```
# To confirm the changes after fixing the typos
df_customers_dc['customer_value_level'].value_counts().reset_index()
```

	customer_value_level	count
0	E	4480
1	B	2390
2	A	2204
3	C	2003
4	D	1480

The code snippet filters the DataFrame df_customers_dc to exclude entries with the customer_value_level value of 'F' and then confirms the changes by displaying the updated counts of the remaining unique values in that column.

Handling Missing Value

```
# Number of missing values in each column
df_customers_dc.isnull().sum()
```

	0
account_id	0
account_activity_level	0
customer_value_level	0
MOB	0
flag_female	0
avg_sales_L36M	738
cnt_sales_L36M	0
last_sales	0
month_since_last_sales	0
count_direct_promo_L12M	0
birth_date	0

dtype: int64

The output displays the number of missing values in each column of the DataFrame df_customers_dc, indicating that the avg_sales_L36M column has 738 missing entries, while all other columns are complete with no missing values.

The missing value will replace with 0 based on count sales

Replacing Missing Value

account_id	0
account_activity_level	0
customer_value_level	0
MOB	0
flag_female	0
avg_sales_L36M	0
cnt_sales_L36M	0
last_sales	0
month_since_last_sales	0
count_direct_promo_L12M	0
birth_date	0

```
# Replace missing values
df_customers_dc['avg_sales_L36M'].fillna(0, inplace=True)

# Confirm whether null data still exist
missing_values_count = df_customers_dc['avg_sales_L36M'].isnull().sum()
print(f'Missing values in avg_sales_L36M after replacement: {missing_values_count}')
```

```
Missing values in avg_sales_L36M after replacement: 0
```

The Pandas library to manage missing values in the avg_sales_L36M column of the DataFrame df_customers_dc. It replaces any NaN entries with zero using the .fillna(0, inplace=True) method. Subsequently, it checks for any remaining null values by employing .isnull().sum() and prints the count of missing values after replacement. This process ensures that all missing data is addressed effectively.

Handling Duplicated

```
# Check the duplicated
df_customers_dc[df_customers_dc['account_id'].duplicated()]
```

The code snippet checks for duplicate entries in the account_id column of the DataFrame df_customers_dc by using the duplicated() method, which identifies any repeated account IDs within the dataset.

Excluding Duplicated

```
# Exclude the duplicate
df_customers_dc.drop_duplicates('account_id', keep = 'first', inplace = True)

# Check the data we have
df_customers_dc.shape
print('Number of rows is: ', df_customers_dc.shape[0])
```

```
Number of rows is:  11750
```

The code snippet removes duplicate entries based on the account_id column in the DataFrame df_customers_dc, retaining the first occurrence of each duplicate and updating the DataFrame in place, after which it checks and prints the total number of remaining rows, confirming that there are 11,750 entries.

Column of Ages

```
#Formulas of Age
cutoff_date = pd.Timestamp('2023-05-31')
df_customers_dc['Age'] = round((cutoff_date - df_customers_dc['birth_date']).dt.days / 365.25
)
df_customers_dc
```

The code snippet calculates the age of each customer in the DataFrame df_customers_dc by determining the difference in days between a specified cutoff date of May 31, 2023, and each customer's birth_date, then converting this difference into years by dividing by 365.25 and rounding the result, ultimately storing the calculated ages in a new column named Age.

Handling Underage

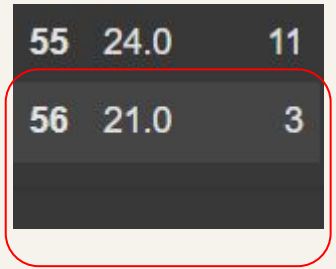
```
#Excluding the Underage
df_under_21 = df_customers_dc[df_customers_dc['Age'] < 21]
df_under_21
```

The code snippet creates a new DataFrame df_under_21 by filtering the original DataFrame df_customers_dc to include only those entries where the Age is less than 21, effectively excluding underage customers from the dataset.

The Legal Age

```
#The Legal Customers
df_customers_dc = df_customers_dc[df_customers_dc['Age']>= 21]
print(df_customers_dc)
```

```
df_customers_dc['Age'].value_counts().reset_index()
```



55	24.0	11
56	21.0	3

The code snippet filters the DataFrame df_customers_dc to retain only those entries where the Age is 21 or older, effectively excluding underage customers, and then prints the updated DataFrame containing only legal customers.

The code snippet counts the occurrences of each age in the Age column of the DataFrame df_customers_dc, and the resulting output shows that there are 11 customers aged 24 and 3 customers aged 21, reflecting the distribution of ages among the legal customers after excluding those under 21.

Summary Table

No	Data Cleaning Steps	Original Data	After Cleaning	Reason
1	Remove Irrelevant	1 XYZ and 1 F	Removed	Not necessary in analysis
2	Missing Value	738 Avg_Sales_L36M	Removed	Can’t fill
3	Duplicates	69 Rows Account_id	Excluding	Not necessary in analysis
4	Converted Data Types	Account_Id = Str, MOB = Int, Birth_date = Datetime	Change	For Necessary Analysis
5	Remove Clients Under Age < 21	119 rows Under age	Excluding	For Business Purpose of Legal Age Eligibility

The Conclusion

In conclusion, removing customers under the age of 21 from eligibility for the RevoU credit card product aligns with both legal requirements and our business objectives of promoting responsible lending practices and minimizing financial risk. This age threshold ensures that applicants possess a certain level of financial maturity and responsibility, thereby reducing the likelihood of default and enhancing the overall stability of the credit portfolio. By focusing on a demographic that is more likely to meet financial obligations, RevoU can foster a healthier customer base, support sustainable growth, and achieve its goal of providing reliable financial services to responsible borrowers.

Milestone 2

Descriptive Statistics, Insight and Recomendations

Here the google collab :

The Code

Descriptive Analysis : Total Sales Code

```
# Calculate total sales for each account
df_customers_dc['total_sales'] = df_customers_dc['avg_sales_L36M'] * df_customers_dc['cnt_sales_L36M']

# Calculate overall total sales in the last 3 years
overall_total_sales = df_customers_dc['total_sales'].sum()
print(f'Total Sales in the Last 3 Years: ${overall_total_sales:.2f}')
```

```
Total Sales in the Last 3 Years: $402603860.00
```

Total sales for each account by multiplying the average sales over 36 months (avg_sales_L36M) with the count of sales transactions (cnt_sales), storing the result in a new column called total_sales. It then computes the overall total sales by summing all values in this new column and prints the result formatted to two decimal places, thereby providing a comprehensive overview of total sales over the last three years.

Total Sales Insight

Based on the total sales figure of $402,603,860.00 over the last three years

Insights from Total Sales :

1. Strong Revenue Generation: The total sales amount indicates robust revenue generation for RevoBank, reflecting effective engagement with clients and successful marketing strategies.

2. Market Positioning: This substantial figure positions RevoBank favorably within the competitive landscape of financial institutions, suggesting a strong market presence.

3. Growth Opportunities: Analyzing this data further can reveal trends in customer behavior and product performance, identifying areas for potential growth or improvement.

4. Strategic Focus: To maintain or enhance this level of sales performance, RevoBank should consider targeted marketing initiatives aimed at increasing product adoption among existing customers and attracting new clients.

The number of clients with no sales : Code

```python
# Create a new column indicating whether clients have sales or not
df_customers_dc['has_sales'] = np.where(df_customers_dc['total_sales'] > 0, 'Yes', 'No')

# Identify clients with no sales (where total_sales is zero)
clients_no_sales = df_customers_dc.loc[df_customers_dc['total_sales'] == 0]

# Count number of clients with no sales
num_clients_no_sales = clients_no_sales.shape[0]
print(f'Number of Clients with No Sales: {num_clients_no_sales}')

# Calculate total number of clients
total_clients = df_customers_dc.shape[0]

# Calculate percentage of clients with no sales
percentage_no_sales = (num_clients_no_sales / total_clients) * 100
print(f'Percentage of Clients with No Sales: {percentage_no_sales:.2f}%')
```

```
Number of Clients with No Sales: 752
Percentage of Clients with No Sales: 6.09%
```

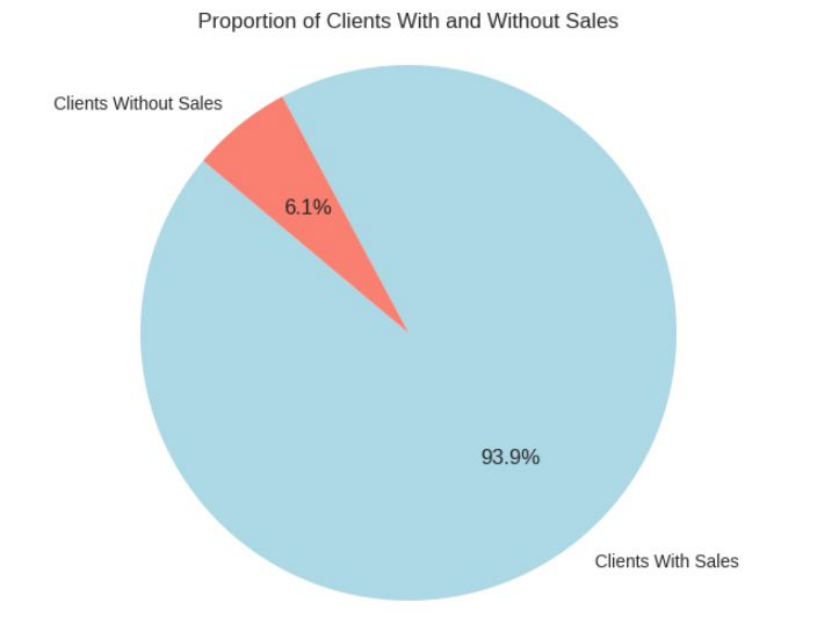
It creates a new column has_sales to indicate whether clients have made any sales, using NumPy's where function to assign 'Yes' or 'No' based on the total_sales value. It identifies clients with no sales by filtering rows where total_sales is zero, then counts these clients and calculates their percentage relative to the total number of clients. The results are printed, showing both the number and percentage of clients with no sales.

The number of clients with no sales : Code

```
# Prepare data for pie chart visualization
labels = ['Clients With Sales', 'Clients Without Sales']
sizes = [total_clients - num_clients_no_sales, num_clients_no_sales]
colors = ['lightblue', 'salmon']

plt.figure(figsize=(8,6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Proportion of Clients With and Without Sales')
plt.axis('equal') # Equal aspect ratio ensures that pie chart is circular.
plt.show()
```

The code snippet uses Matplotlib to create a pie chart visualizing the proportion of clients with and without sales. It defines labels for the two categories, calculates their sizes based on the number of clients, and assigns colors for each segment. The plt.pie function is used to generate the pie chart with specified labels, colors, and percentage display (autopct). The chart is titled "Proportion of Clients With and Without Sales," set to be circular using plt.axis('equal'), and displayed with plt.show().

The number of clients with no sales : Insight



Insights from Client Sales Data

- Low Proportion of Inactive Clients: Only 6.1% of clients have not made any purchases, indicating that a significant majority (93.9%) are actively engaging with RevoBank's products.
- Total Number of Inactive Clients: The total count of 752 clients without sales suggests there is a small segment that may require targeted marketing efforts to encourage engagement.
- Opportunity for Growth: While the majority of clients are active, focusing on the 6.1% who have not engaged in any transactions presents an opportunity for RevoBank to enhance its marketing strategies—potentially through personalized promotions or outreach initiatives aimed at re-engaging these inactive clients.

Strategic Recommendations:

1. Implement targeted campaigns specifically designed to address the needs and preferences of inactive clients.
2. Analyze reasons behind inactivity (e.g., lack of awareness about products or services) to tailor communication effectively.

The account activity level : Code

```python
# Group by account activity level and calculate median months since last transaction
activity_summary = df_customers_dc.groupby('account_activity_level').agg(
    median_months_since_last_transaction=('month_since_last_sales', 'median'),
    avg_transactions=('cnt_sales_L36M', 'mean')
).reset_index()

# Display summary statistics for verification
print(activity_summary)
```

```
  account_activity_level  median_months_since_last_transaction  \
0                      X                                  18.0
1                      Y                                  18.0
2                      Z                                  19.0

   avg_transactions
0          3.284143
1          2.339471
2          1.227781
```

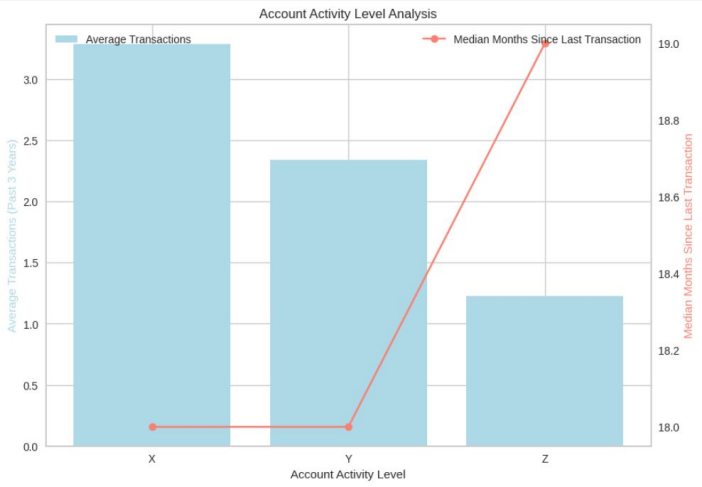
The code snippet groups data from the DataFrame df_customers_dc by the account_activity_level and calculates two key metrics: the median number of months since the last transaction and the average transactions over 36 months (cnt_sales_L36M). The .agg() function is used to specify these calculations, with median applied to month_since_last_sales and mean applied to cnt_sales_L36M. The results are then reset to a standard index for easier readability. Finally, it prints out the summary statistics, showing how account activity levels correlate with transaction recency and average sales.

The account activity level : Code

```
# Set up figure with subplots for better visualization
fig, ax1 = plt.subplots(figsize=(10, 7))

# Create bar chart for average transactions per account activity level
ax1.bar(activity_summary['account_activity_level'],
        activity_summary['avg_transactions'],
        color='lightblue', label='Average Transactions')

# Create a second y-axis to plot median months since last transaction
ax2 = ax1.twinx()
ax2.plot(activity_summary['account_activity_level'],
        activity_summary['median_months_since_last_transaction'],
        color='salmon', marker='o', label='Median Months Since Last Transaction')

# Adding titles and labels
ax1.set_title('Account Activity Level Analysis')
ax1.set_xlabel('Account Activity Level')
ax1.set_ylabel('Average Transactions (Past 3 Years)', color='lightblue')
ax2.set_ylabel('Median Months Since Last Transaction', color='salmon')

# Add legends to distinguish between two datasets plotted on different axes
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Adjust grid settings for better visibility
ax2.grid(False)  # Disable grid lines on secondary axis (line plot)

plt.show()
```

The code snippet sets up a dual-axis visualization using Matplotlib to analyze account activity levels. It first creates a bar chart on the primary axis (ax1) to display average transactions per account activity level, using light blue bars for clarity. A secondary y-axis (ax2) is then created to plot the median months since the last transaction as a line graph with salmon-colored markers. Titles and labels are added for both axes, enhancing interpretability. Legends are included to differentiate between the two datasets, and grid lines on the secondary axis are disabled for better visibility before displaying the final plot with plt.show().

The account activity level Insight



No	Account	Median	Average
1	X	18.0	3.28
2	Y	18.0	2.33
3	Z	19.0	1.22

Insights

1. High Engagement: Clients in Activity Level X have the highest average transactions at 3.28, indicating strong engagement.
2. Lower Activity: Clients in Activity Levels Y (2.34) and Z (1.23) show significantly lower transaction averages, suggesting less engagement.
3. Transaction Recency: Median months since the last transaction is similar for Levels X and Y (18 months) but slightly higher for Level Z (19 months), indicating potential disengagement among lower activity levels.

Strategic Recommendations:

1. RevoBank should consider implementing re-engagement campaigns aimed at clients in Levels Y and Z to boost their activity levels.
2. Tailored promotions or personalized communication could help increase transaction frequency among these groups.

Gender Comparison : Code

```
# Calculate profit for each client
df_customers_dc['profit'] = df_customers_dc['total_sales'] * 0.024

# Group by gender flag and calculate average profit per client
gender_profit_comparison = df_customers_dc.groupby('flag_female')['profit'].agg(['mean', 'median', 'std']).reset_index()
gender_profit_comparison.columns = ['Gender_Flag', 'Average_Profit', 'Median_Profit', 'Std_Dev']

# Display summary statistics for verification
print(gender_profit_comparison)
```

```
   Gender_Flag  Average_Profit  Median_Profit     Std_Dev
0            0      788.048600         660.00  522.482179
1            1      776.637916         695.52  494.891269
```

The code snippet calculates and analyzes the profit generated by clients based on their gender. First, it computes the profit for each client by multiplying their total sales by a fixed rate (0.024) and stores this in a new column called profit. Next, it groups the data by the flag_female column to differentiate between genders and aggregates the profit data to calculate average, median, and standard deviation values using .agg(). The results are then reset to a standard index for clarity. Finally, it renames the columns for better readability before printing out the summary statistics that compare profits across different gender flags.

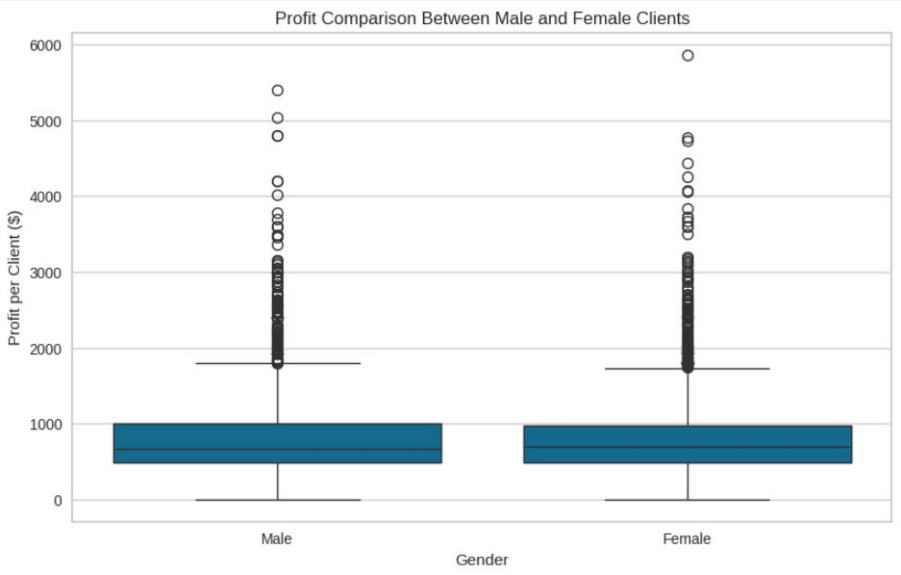
Gender Comparison : Code

```
# Create a new column indicating gender for better labeling in plots
df_customers_dc['Gender'] = np.where(df_customers_dc['flag_female'] == 1, 'Female', 'Male')

# Set up boxplot visualization
plt.figure(figsize=(10,6))
sns.boxplot(x='Gender', y='profit', data=df_customers_dc)
plt.title('Profit Comparison Between Male and Female Clients')
plt.xlabel('Gender')
plt.ylabel('Profit per Client (€)')
plt.show()
```

The code snippet creates a boxplot to visualize profit comparisons between male and female clients. It first adds a new column, Gender, to the DataFrame df_customers_dc, using NumPy's where function to label clients as 'Female' or 'Male' based on the value of the flag_female column. Next, it sets up a boxplot visualization with Seaborn, specifying gender on the x-axis and profit on the y-axis. The plot is titled "Profit Comparison Between Male and Female Clients," with appropriate labels for both axes. Finally, it displays the plot using plt.show().

Gender Comparison Insight

No	Gender	Average Profit	Median Profit	Std Dev
1	Male	788.048	660.00	522.482
2	Female	776.637	695.52	494.891



Insights

1. Average Profit: Male clients generate an average profit of $788.05, slightly higher than female clients at $776.64.
2. Median Profit: Female clients have a higher median profit of $695.52, compared to male clients at $660, indicating more consistent profitability among females.
3. Profit Distribution: Both genders show similar variability in profits, but males have more extreme high-profit outliers.
4. Conclusion: While males exhibit slightly higher average profits, females demonstrate stronger typical profitability per client, suggesting targeted marketing strategies could enhance engagement for both groups.

Strategic Recommendations:

Understanding these dynamics can help RevoBank tailor its marketing strategies effectively—potentially focusing on enhancing engagement with both genders based on their unique profitability patterns.

The proportion of total sales in the past 3 years based on generations : The Code

```python
# Extract year of birth from 'birth_date' column
df_customers_dc['year_of_birth'] = df_customers_dc['birth_date'].dt.year

# Define age groups for generations
bins = [1927, 1946, 1965, 1980, 1996, 2012] # Boomer: 1928-1945; Gen-X: 1966-1980; Millennial: 1981-1996; Gen-Z: 1997-2012; Gen-Alpha: after 2012
labels = ['Boomer', 'Gen-X', 'Millennial', 'Gen-Z', 'Gen-Alpha']

# Categorize year_of_birth into generations
df_customers_dc['generation'] = pd.cut(df_customers_dc['year_of_birth'], bins=bins, labels=labels)

# Calculate total sales per generation
sales_by_generation = df_customers_dc.groupby('generation').agg(
    total_sales=('total_sales', 'sum'),
    avg_sales=('avg_sales_L36M', 'mean'),
    count_sales=('cnt_sales_L36M', 'sum')
).reset_index()

# Calculate proportion of total sales for each generation
total_overall_sales = sales_by_generation['total_sales'].sum()
sales_by_generation['proportion'] = (sales_by_generation['total_sales'] / total_overall_sales) * 100

print(sales_by_generation)
```

The code snippet analyzes sales data by generational groups based on clients' birth years. It first extracts the year of birth from the birth_date column and defines age group bins corresponding to different generations (Boomer, Gen-X, Millennial, Gen-Z, and Gen-Alpha). The pd.cut() function categorizes each client's year of birth into these defined generational labels. Next, it calculates total sales per generation by grouping the DataFrame and aggregating total sales, average sales over 36 months (avg_sales_L36M), and count of transactions (cnt_sales_L36M). Finally, it computes the proportion of total sales for each generation relative to overall sales and stores this in a new column called proportion, which is then printed for review.

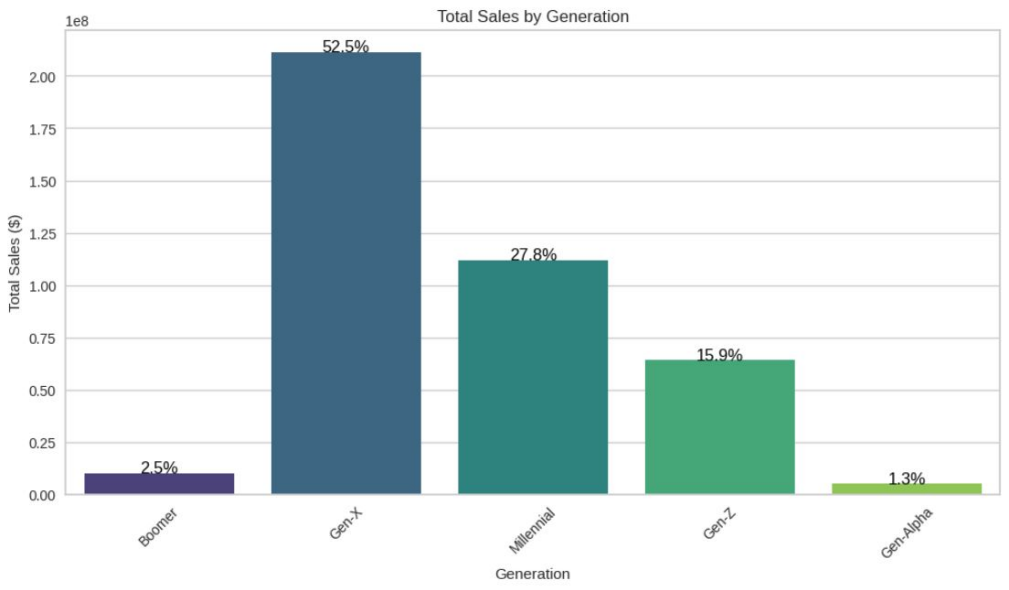
The proportion of total sales in the past 3 years based on generations : The Code

```
plt.figure(figsize=(12,6))

# Create bar plot for Total Sales by Generation
sns.barplot(x='generation', y='total_sales', data=sales_by_generation,
            palette='viridis')

plt.title('Total Sales by Generation')
plt.xlabel('Generation')
plt.ylabel('Total Sales ($)')
plt.xticks(rotation=45)

for index in range(len(sales_by_generation)):
    plt.text(index,
             sales_by_generation.total_sales[index],
             f"{sales_by_generation.proportion[index]:.1f}%",
             color='black',
             ha="center")

plt.show()
```

The code snippet creates a bar plot to visualize total sales by generation using Seaborn. It sets the figure size and generates a bar plot with generations on the x-axis and total sales on the y-axis, applying a 'viridis' color palette for aesthetic appeal. The plot is titled "Total Sales by Generation," with appropriate labels for both axes, and x-tick labels are rotated for better readability. Additionally, it annotates each bar with the corresponding proportion of total sales as a percentage, centering this text above each bar in black font. Finally, it displays the completed plot using plt.show().

The proportion of total sales in the past 3 years based on generations : Insight

	generation	total_sales	avg_sales	count_sales	proportion
0	Boomer	10144070.0	13627.580645	888	2.519616
1	Gen-X	211415060.0	13547.182944	18207	52.511931
2	Millennial	111935250.0	15345.049824	8332	27.802826
3	Gen-Z	64030320.0	14783.380884	4933	15.904050
4	Gen-Alpha	5079160.0	12649.090909	417	1.261578



Insights

1. Dominant Generation: Gen-X accounts for 52.5% of total sales, making them the most engaged demographic.
2. Significant Contribution: Millennials contribute 27.8%, indicating they are also an important market segment.
3. Lower Engagement: Both Gen-Z (15.9%) and Gen-Alpha (1.3%) show significantly lower sales contributions, suggesting potential areas for growth.

Strategic Recommendations:

1. To enhance overall sales performance, RevoBank should consider developing targeted marketing strategies aimed at underrepresented generations like Gen-Z and Boomers.
2. For Gen-Z: Leverage digital platforms and social media to promote products that align with their values (e.g., sustainability).
3. For Boomers: Focus on traditional marketing channels while emphasizing products that cater to retirement planning or wealth management.

Relationship the number of email and SMS messages and total sales performance : Code

```
# Group by count of direct promotions received in the last year
promo_sales_analysis = df_customers_dc.groupby('count_direct_promo_L12M').agg(
    total_sales=('total_sales', 'sum'),
    num_clients=('account_id', 'count')
).reset_index()

# Calculate average sales per client for each promo communication count
promo_sales_analysis['avg_sales_per_client'] = promo_sales_analysis['total_sales'] / promo_sales_analysis['num_clients']

print(promo_sales_analysis)
```

The code snippet analyzes the impact of direct promotions received by clients over the last year on sales performance. It groups the data in df_customers_dc by the count of direct promotions (count_direct_promo_L12M) and aggregates total sales and client counts using .agg(). The results are stored in a new DataFrame called promo_sales_analysis, which is reset to a standard index for clarity. Subsequently, it calculates average sales per client for each promotion communication count by dividing total sales by the number of clients, storing this value in a new column named avg_sales_per_client. Finally, it prints out the analysis results for review.

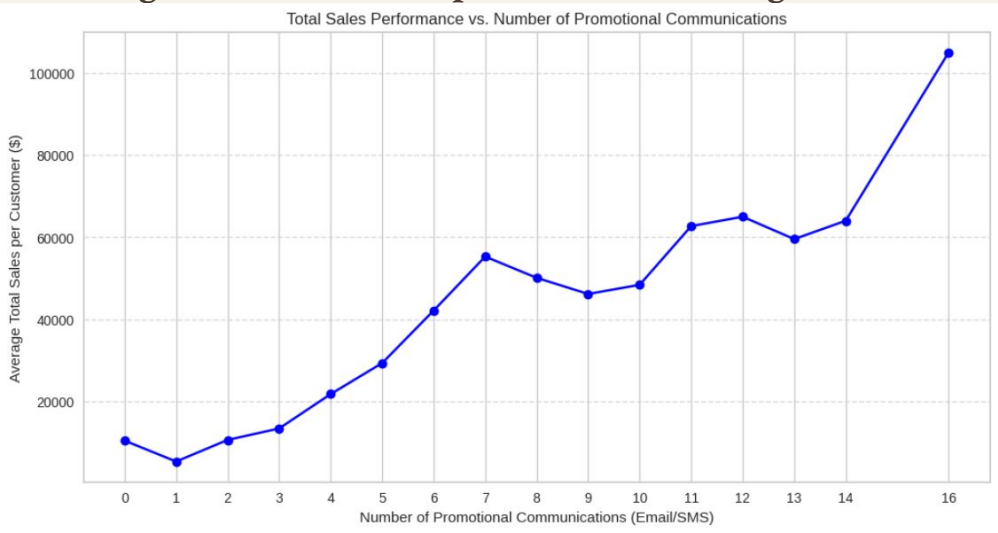
Relationship the number of email and SMS messages and total sales performance : Code

```
# Group by promotional communication count and calculate total sales per customer
promo_sales_analysis = df_customers_dc.groupby('count_direct_promo_L12M').agg(
    total_sales_per_customer=('total_sales', 'mean')
).reset_index()

# Set up the line chart visualization
plt.figure(figsize=(12, 6))

# Create line plot for total sales per customer based on promo communication count
plt.plot(promo_sales_analysis['count_direct_promo_L12M'],
         promo_sales_analysis['total_sales_per_customer'],
         marker='o', color='blue')

# Adding titles and labels
plt.title('Total Sales Performance vs. Number of Promotional Communications')
plt.xlabel('Number of Promotional Communications (Email/SMS)')
plt.ylabel('Average Total Sales per Customer ($)')

# Adjust grid settings for better visibility
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Simple horizontal grid lines only

plt.xticks(promo_sales_analysis['count_direct_promo_L12M'])  # Ensure all x-ticks are shown

plt.show()
```

The code snippet visualizes the relationship between promotional communications and average sales per customer using a line plot. It first groups the data in df_customers_dc by the count of direct promotions received (count_direct_promo_L12M) and calculates the mean total sales per customer, storing this in a DataFrame called promo_sales_analysis. A line plot is then created to display average sales against the number of promotional communications, with blue markers for clarity. The plot is titled "Total Sales Performance vs. Number of Promotional Communications," and appropriate labels are added for both axes. Grid lines are adjusted for better visibility, ensuring that all x-ticks are displayed before finally rendering the plot with plt.show().

Relationship the number of email and SMS messages and total sales performance : Insight

	count_direct_promo_L12M	total_sales	num_clients	avg_sales_per_client
0	0	52000.0	5	10400.000000
1	1	207160.0	39	5311.794872
2	2	2265330.0	214	10585.654206
3	3	16744920.0	1252	13374.536741
4	4	52462820.0	2411	21759.776027
5	5	89816130.0	3064	29313.358355
6	6	176918780.0	4200	42123.519048
7	7	52729900.0	954	55272.431866
8	8	5110020.0	102	50098.235294
9	9	2077100.0	45	46157.777778
10	10	1161540.0	24	48397.500000
11	11	2132000.0	34	62705.882353
12	12	454940.0	7	64991.428571
13	13	238170.0	4	59542.500000
14	14	128000.0	2	64000.000000
15	16	105050.0	1	105050.000000



Insights

1. Positive Correlation: There is a clear upward trend in average total sales per customer as the number of promotional communications (emails/SMS) increases.
2. Peak Performance: The highest average sales per client occur at 6 communications, reaching approximately $42,123.
3. Diminishing Returns: After 6 communications, while sales remain high, the increase in average sales per client starts to plateau.

Strategic Recommendations:

To enhance overall sales performance, RevoBank should focus on optimizing their communication strategy by finding an effective balance in messaging frequency.

1. Consider targeting clients with around 5-6 messages as this range appears most effective based on observed data.
2. Implement A/B testing to refine content and timing of promotions to maximize engagement without overwhelming customers.

Overall insight and recommendation

Insights

1. Strong Revenue: RevoBank generated $402,603,860.00 in total sales over the past three years, reflecting effective client engagement.
2. High Client Engagement: 93.9% of clients are active users, with only 6.1% not making any purchases.
3. Generational Focus: Sales are primarily driven by Gen-X (52.5%) and Millennials (27.8%), indicating a need to engage younger generations more effectively.

Recommendations

1. Targeted Marketing: Develop tailored campaigns for underrepresented groups like Gen-Z and Boomers to enhance engagement.
2. Promotional Strategies: Increase personalized email and SMS communications while ensuring content remains relevant to avoid overwhelming clients.
3. Product Development: Create financial products that cater specifically to the needs of younger generations to drive further sales growth.

Milestone 3

Customer Segmentation

Here the google collab :

The Code

RFM as the clustering methods

Why Prefer RFM Over K-Means?

1. Business Interpretability:
- RFM uses meaningful business metrics — Recency (how recently a customer purchased), Frequency (how often they purchase), and Monetary value (how much they spend).
- These dimensions are intuitive for marketers and decision-makers to understand and act upon.
- K-means clusters based on raw or transformed features that may not have direct business meaning unless carefully engineered.

2. Focus on Customer Behavior:
- RFM explicitly captures key behavioral aspects of customers that drive engagement and profitability.
- It helps identify valuable segments like loyal customers, at-risk customers, or new prospects.

The Steps on the Code

1. Define DataFrame
2. Creating 3D
3. Labeling the RFM
4. Checking Percentile
5. Checking Distributions
6. Get Categories
7. Labeling
8. Assign RFM
9. Combining RFM
10. Interpreting RFM Segments
11. Identify Segments

The Segment

1. Higher Average Sales per Client (avg_monetary)
- Segment: Champions
- Explanation: Champions have the highest average monetary value (~57,417), indicating they spend the most per transaction on average.
2. Higher Average Transaction Frequency (avg_frequency)
- Segment: Champions
- Explanation: Champions also show the highest purchase frequency (~5.99 times), meaning they buy more often than other segments.
3. Higher Total Revenue Generated
- Segment: Champions
- Explanation: With total revenue over 86 million, Champions contribute the largest share of sales revenue to RevoBank.
4. Larger Proportion of Clients with No Sales (Dormancy Rate)
- Segment: Lost
- Explanation: The Lost segment has a dormancy rate of ~52.7%, indicating over half their customers have no recent sales activity and are inactive.

Business Background Recap

Goal

Increase usage of credit card products among existing customers.

Profit Margin

2.4% profit margin on sales revenue.

Focus

Target segments that maximize revenue growth and customer engagement.

Business Opportunites based on data

1. Top Priority is Champions with this details

➔ Champions have the highest average sales per client (~57,417), highest transaction frequency (~5.99), and generate the largest total revenue (~86.35 million). Their dormancy rate is zero, meaning they are highly active and loyal.

➔ With persona : Highly engaged, frequent buyers who value your brand—ideal for upselling premium financial products.

➔ Business Opportunities of Champion : These customers are your best promoters and biggest spenders. Rewarding them with exclusive offers or early access to new credit card products can boost usage further and increase profitability.

2. Second Priority is At Risk

➔ At Risk customers have high average sales (~49,014) and decent transaction frequency (~3.42), generating substantial total revenue (~76.27 million). However, they are at risk of becoming inactive soon, though their dormancy rate is currently zero.

➔ With Persona : These customers have been valuable contributors but show signs of reduced engagement.

➔ Business Opportunities of At Risk : Focus on re-engaging these valuable customers with personalized offers and incentives to prevent churn and recover lost revenue before they become dormant.

Business Recommendations for Performance Management Lead

Focus on Champions & At Risk Segments

Why:

- ➔ Champions generate the highest revenue and transaction frequency with zero dormancy — they are your most valuable customers.
- ➔ At Risk customers have high sales but show early signs of disengagement; reactivating them can prevent revenue loss.

Actions:

- ➔ Implement loyalty rewards, exclusive offers, and early access to new credit products for Champions to maintain engagement and encourage advocacy.
- ➔ Launch personalized retention campaigns targeting At Risk customers with tailored promotions or incentives to reactivate usage.

Additional Recommendations Leverage Data Insights Continuously

- Regularly monitor RFM metrics and dormancy rates per segment to dynamically adjust marketing strategies.
- Use predictive analytics models to identify emerging at-risk customers earlier.
- Align marketing spend towards segments demonstrating highest ROI potential based on updated data insights.

Thank you