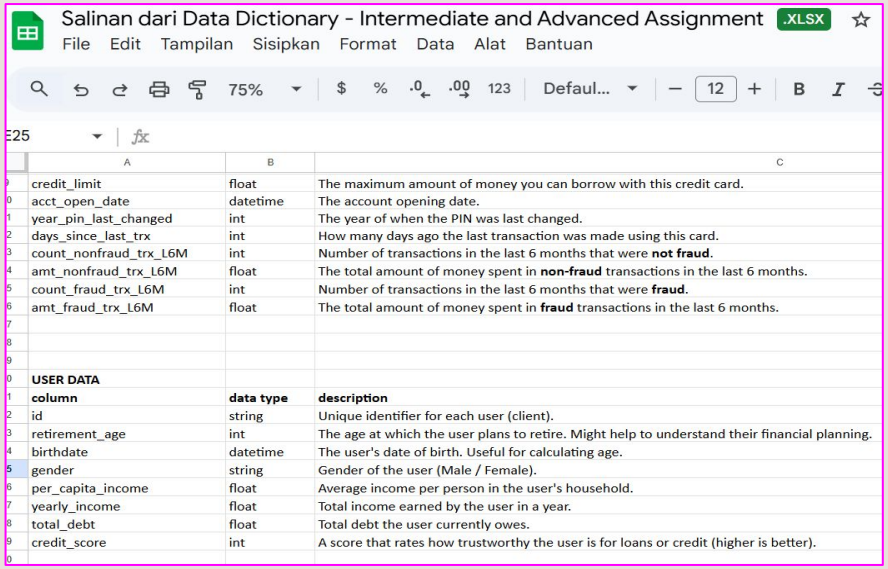
Data Driven Insights RevoBank Credit Card Trends using a Phython Tool

Santa Riyanti T
14/22 Desember 2025 and 2 Jan 2026
RevoU FSDA Batch OCT25

MILESTONE 1

COMPANY AND DATASET OVERVIEW

RevoBank is a digital banking institution based in Indonesia, offering a wide range of financial products including savings, loans, and credit cards. The bank is committed to enhancing customer experience through data driven strategies and modern technology. The Performance Management (PM) team collaborates with the Management Information System (MIS) team to optimize credit card usage among existing customers by leveraging behavioral insights and transaction data.



Salinan dari Data Dictionary - Intermediate and Advanced Assignment .XLSX

File Edit Tampilan Sisipkan Format Data Alat Bantuan

	A	B	C
9	credit_limit	float	The maximum amount of money you can borrow with this credit card.
10	acct_open_date	datetime	The account opening date.
11	year_pin_last_changed	int	The year of when the PIN was last changed.
12	days_since_last_trx	int	How many days ago the last transaction was made using this card.
13	count_nonfraud_trx_L6M	int	Number of transactions in the last 6 months that were **not fraud**.
14	amt_nonfraud_trx_L6M	float	The total amount of money spent in **non-fraud** transactions in the last 6 months.
15	count_fraud_trx_L6M	int	Number of transactions in the last 6 months that were **fraud**.
16	amt_fraud_trx_L6M	float	The total amount of money spent in **fraud** transactions in the last 6 months.
17			
18			
19			
20	**USER DATA**		
21	**column**	**data type**	**description**
22	id	string	Unique identifier for each user (client).
23	retirement_age	int	The age at which the user plans to retire. Might help to understand their financial planning.
24	birthdate	datetime	The user's date of birth. Useful for calculating age.
25	gender	string	Gender of the user (Male / Female).
26	per_capita_income	float	Average income per person in the user's household.
27	yearly_income	float	Total income earned by the user in a year.
28	total_debt	float	Total debt the user currently owes.
29	credit_score	int	A score that rates how trustworthy the user is for loans or credit (higher is better).
30			

DISCLAIMERS :

- This analysis is based on a publicly dataset and does not reflect real world financial or business insights.
- RevoBank is a fictional entity and the results presented here are purely for educational purposes.

ANALYTICAL OBJECTIVE

a. State the goal of analysis

This dataset represents a financial transaction ecosystem designed for fraud detection analysis. It combines customer demographic information, card attributes, transaction records, merchant category data, and fraud labels. This multi table relational design allows analysts to explore relationships between customer profiles, card behavior, and transaction patterns. See the next slide for the analysis goal in the Table Explanations.

b. Goal of analysis related to business problem

The goal of this analysis is to generate a clear, data driven understanding of RevoBank credit card performance and customer behavior over the past six months. By integrating user level and card level data, the analysis aims to uncover actionable insights that will help the Performance Management (PM) team increase credit card usage among existing customers. Specifically, the analysis seeks to :

- Measure overall credit card performance, including transaction volume, transaction value, and profitability.
- Identify key customer behavior patterns, such as spending frequency, average transaction size, and engagement trends.
- Segment customers into meaningful personas that reflect their usage intensity, value contribution, and potential for growth.
- Highlight opportunities for targeted interventions, enabling the PM team to design more effective marketing strategies, product offers, and engagement campaigns.

Table Explanations in the Fraud Detection Database Schema

TABLE	COLUMN	CONTAINS	PURPOSE
card	id, client_id, card_brand, card_number, expires, cvv, credit_limit, acct_open_date, credit_limit, year_pin_last_changed, day_since_last_trx	Detailed information about each credit card used by customers.	Useful for analyzing card usage behavior, credit risk, and transaction frequency.
user	id, retirement_age, birthdate, gender, per_capita_income, yearly_income, total_debt, credit_score	Stores demographic and financial information about each customer.	Supports customer profiling, risk segmentation, and financial behavior analysis.
transaction	id, date, client_id, card_id, amount, use_chip, merchant_id, merchant_city, merchant_state, mcc	Captures all financial transactions made by users.	Central to fraud detection, behavioral analysis, and transaction pattern modeling.
mcc	mcc, mcc_description	Provides reference information for merchant categories.	Helps classify transaction types and identify spending patterns by merchant category.
train_fraud_label	transaction_id, target	Contains fraud labels used for supervised machine learning.	Serves as ground truth for training fraud detection models using historical transaction data.

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on Card Table*

Colab Link

4a. Convert the correct data type in the "card" table dataset

4a.1. Convert the data types (client_id and id) from interger to object/string

```
df_card.dtypes
```

	0
id	int64
client_id	int64

```
df_card['id'] = df_card['id'].astype(str)
df_card['client_id'] = df_card['client_id'].astype(str)
```

```
df_card.dtypes
```

	0
id	object
client_id	object

4a.2. Change all count data types (count_nonfraud_trx_L6M and count_fraud_trx_L6M) from float to interger

count_nonfraud_trx_L6M	float64
amt_nonfraud_trx_L6M	object
count_fraud_trx_L6M	float64

```
df_card['count_nonfraud_trx_L6M'] = df_card['count_nonfraud_trx_L6M'].fillna(0).astype(int)
df_card['count_fraud_trx_L6M'] = df_card['count_fraud_trx_L6M'].fillna(0).astype(int)
```

count_nonfraud_trx_L6M	int64
amt_nonfraud_trx_L6M	object
count_fraud_trx_L6M	int64

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on Card Table*

4a.3. Change data related to money (credit_limit, amt_nonfraud_trx_L6M, amt_fraud_trx_L6M) from object to float

credit_limit	object
acct_open_date	object
year_pin_last_changed	int64
days_since_last_trx	int64
count_nonfraud_trx_L6M	int64
amt_nonfraud_trx_L6M	object
count_fraud_trx_L6M	int64
amt_fraud_trx_L6M	object

```
1 df_card['credit_limit'] = df_card['credit_limit'].str.replace('Rp', '').str.
  replace('.', '', regex=False).astype(float)
2 df_card['amt_nonfraud_trx_L6M'] = df_card['amt_nonfraud_trx_L6M'].str.replace
  ('Rp', '').str.replace('.', '', regex=False).astype(float)
3 df_card['amt_fraud_trx_L6M'] = df_card['amt_fraud_trx_L6M'].str.replace('Rp',
  '').str.replace('.', '', regex=False).fillna(0).astype(float)
```

credit_limit	float64
acct_open_date	object
year_pin_last_changed	int64
days_since_last_trx	int64
count_nonfraud_trx_L6M	int64
amt_nonfraud_trx_L6M	float64
count_fraud_trx_L6M	int64
amt_fraud_trx_L6M	float64

4a.4. Change all the datetime data (expires and acct_open_date) from object to timeseries

expires	object
cvv	int64
credit_limit	float64
acct_open_date	object

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on Card Table*

4b. Check the unique values in each column and see some typo data that either needs to be fixed or excluded (use value_counts())

```
df_card['id'].value_counts()
```

id	count
4972	2
1781	2
6115	2
4167	2
2352	2

```
df_card['client_id'].value_counts()
```

client_id	count
797	9
1301	9
921	9
1345	8
665	8

```
df_card['card_brand'].value_counts()
```

card_brand	count
Mastercard	2826
Visa	2093
Amex	402
JCB	206
Visa	69
Jcb	3

```
df_card['card_number'].value_counts()
```

card_number	count
4900994567069763	2
4806267788873524	2
4816027381867141	2
4557993037807765	2
4417739183409950	2

```
df_card['acct_open_date'].value_counts()
```

acct_open_date	count
2020-02-01	233
2025-02-01	82
2020-01-01	79
2025-01-01	78
2015-02-01	51

```
df_card['expires'].value_counts()
```

expires	count
2026-02-01	350
2026-01-01	127
2030-03-01	106
2026-11-01	105
2030-10-01	102

```
df_card['year_pin_last_changed'].value_counts()
```

year_pin_last_changed	count
2025	1083
2016	734
2015	657
2017	435

```
df_card['cvv'].value_counts()
```

cvv	count
877	15
740	14
939	14
269	13

4b.1. Handling typos in the *card_brand* column, particularly for entries such as Visa and JCB, to ensure consistent spelling and avoid duplicate categories caused by misspellings or typo.

```
1  df_card['card_brand'].value_counts()
```

card_brand	count
Mastercard	2826
Visa	2093
Amex	402
JCB	206
Visa	69
Jcb	3

```
1  df_card['card_brand'] = df_card['card_brand'].replace({'Visa ': 'Visa', 'Jcb': 'JCB'})
```

```
1  df_card['card_brand'].value_counts()
```

card_brand	count
Mastercard	2826
Visa	2162
Amex	402
JCB	209

4b.2. As both id serve as unique identifiers within the dataset, ensuring that they contain no duplicate values is essential. The steps focus on correcting any inconsistencies or duplicates found in these columns.

```
1  df_card.drop_duplicates(keep='first', subset=['id'])
```

	id	client_id	card_brand	card_number	expires	cvv	credit_limit
0	0	1362	Amex	393314135668401	2030-04-01	866	53189000.0
1	1	550	Mastercard	5278231764792292	2030-06-01	396	18200000.0
2	2	556	Mastercard	5889825928297675	2027-09-01	422	31298000.0
3	3	1937	Visa	4289888672554714	2026-04-01	736	25732000.0
5565	6143	253	Mastercard	5482273460029161	2030-07-01	79	38739000.0
5566	6144	737	Mastercard	5413710543564908	2030-10-01	454	10355000.0
5567	6145	682	Visa	4605950682175499	2026-08-01	517	25178000.0

5568 rows × 14 columns

4b.3. Same as id, card_number serve as unique identifiers too, ensuring again that they contain no duplicate values. The steps still focus on correcting any inconsistencies or duplicates in these columns.

```
df_card.drop_duplicates(keep='first', subset=['card_number'])
```

	id	client_id	card_brand	card_number	expires	cvv	credit_limit
0	0	1362	Amex	393314135668401	2030-04-01	866	53189000.0
1	1	550	Mastercard	5278231764792292	2030-06-01	396	18200000.0
2	2	556	Mastercard	5889825928297675	2027-09-01	422	31298000.0
3	3	1937	Visa	4289888672554714	2026-04-01	736	25732000.0
5564	6142	1236	Mastercard	5791756986936871	2030-11-01	196	19178000.0
5565	6143	253	Mastercard	5482273460029161	2030-07-01	79	38739000.0
5566	6144	737	Mastercard	5413710543564908	2030-10-01	454	10355000.0
5567	6145	682	Visa	4605950682175499	2026-08-01	517	25178000.0

5568 rows × 14 columns

4c. Check Missing Value

4c.1. Filling missing value in credit_limit column

Cause the proportion of missing values in credit_limit is relatively small (12 out of 5599 entries), we can filling missing values with the median helps maintain the natural structure of the dataset without introducing artificial inflation or deflation in the credit limit values and median imputation is easy to implement and doesn't require complex assumptions about the data distribution.

```
df_card.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5599 entries, 0 to 5598
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      5599 non-null   object
 1   client_id               5599 non-null   object
 2   card_brand              5599 non-null   object
 3   card_number             5599 non-null   int64
 4   expires                 5599 non-null   datetime64[ns]
 5   cvv                     5599 non-null   int64
 6   credit_limit            5587 non-null   float64
 7   acct_open_date          5599 non-null   datetime64[ns]
 8   year_pin_last_changed   5599 non-null   int64
 9   days_since_last_trx     5599 non-null   int64
 10  count_nonfraud_trx_L6M  5599 non-null   int64
 11  amt_nonfraud_trx_L6M    3707 non-null   float64
 12  count_fraud_trx_L6M     5599 non-null   int64
 13  amt_fraud_trx_L6M       5599 non-null   float64
dtypes: datetime64[ns](2), float64(3), int64(6), object(3)
memory usage: 612.5+ KB
```

```
median_credit_limit = df_card['credit_limit'].median()
df_card['credit_limit'] = df_card['credit_limit'].fillna(median_credit_limit)
```

```
df_card.isnull().sum()
```

	0
id	0
client_id	0
card_brand	0
card_number	0
expires	0
cvv	0
credit_limit	0

4d. Based on the card expiry date, remove all cards that have already expired and also exclude cards with a credit limit 0 (data calculated as May 31, 2025)

condition 1 → remove all card that have already expiry by May 31, 2025
condition 2 → exclude cards with a credit limit 0
Define reference date as May 31, 2025 for checking card expiry to determine which cards are still valid (create a datetime object for May 31, 2025 as referance_date)

```
reference_date = pd.to_datetime('2025-05-31')
print(f"Reference Date: {reference_date}")
```

```
Reference Date: 2025-05-31 00:00:00
```

```
df_card_filtered = df_card[(df_card['expires'] >= reference_date) & (df_card['credit_limit'] > 0)]
print("First  5 rows of the filtered DataFrame:")
print(df_card_filtered.head())
print(f"\nShape of the filtered DataFrame: {df_card_filtered.shape}")
```

```
First  5 rows of the filtered DataFrame:
   id client_id  card_brand       card_number    expires  cvv  credit_limit
0   0      1362        Amex   393314135668401 2030-04-01  866     53189000.0
1   1       550  Mastercard  5278231764792292 2030-06-01  396     18200000.0
2   2       556  Mastercard  5889825928297675 2027-09-01  422     31298000.0
3   3      1937        Visa  4289888672554714 2026-04-01  736     25732000.0
4   4      1981  Mastercard  5433366978583845 2030-03-01  530     30500000.0

  acct_open_date  year_pin_last_changed  days_since_last_trx  \
0     1996-01-01                   2019                   17
1     1999-01-01                   2018                   27
2     2000-01-01                   2016                   20
3     2000-01-01                   2020                    7
4     2002-01-01                   2012                   14

   count_nonfraud_trx_L6M  count_fraud_trx_L6M  amt_fraud_trx_L6M
0                     181                    0                0.0
1                     148                    0                0.0
2                     415                    0                0.0
3                     148                    0                0.0
4                      48                    0                0.0

Shape of the filtered DataFrame: (5559, 13)
```

Before Filtering (Original Data Condition) :

- The dataset contained all 5599 card records, including :
  - Cards that were already expired before May 31, 2025.
  - Cards with a credit limit of 0, meaning they were not usable for credit transactions.
- No filtering had been applied, so active and inactive cards were mixed together, cards with zero financial usability were still present and the dataset was not yet ready for accurate analysis of user behavior, fraud risk, or credit performance.

After Filtering (Cleaned Data Condition) :

- Only cards that expire on or after May 31, 2025 were kept.
- Cards with credit_limit > 0 were retained.
- The resulting dataset contains only valid, active, financially usable cards.
- This ensures cleaner (more reliable analysis), no noise from expired or inactive cards, and better accuracy for modeling and segmentation.

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on User Table*

Colab Link

5a.l. Convert the correct data type in the “user” table dataset

5a.l.a Change the data types "id" from int64 to object

```
df_user.dtypes
```

	0
id	int64
retirement_age	int64
birthdate	object
gender	object
per_capita_income	object
yearly_income	object
total_debt	object
credit_score	int64

```
df_user['id'] = df_user['id'].astype(str)
```

```
df_user.dtypes
```

	0
id	object
retirement_age	int64
birthdate	object
gender	object
per_capita_income	object
yearly_income	object
total_debt	object
credit_score	int64

5a.l.b. Change the data types "birthdate" from int64 to object

```
df_user.dtypes
```

	0
id	int64
retirement_age	int64
birthdate	object
gender	object
per_capita_income	object
yearly_income	object
total_debt	object
credit_score	int64

```
df_user['birthdate'] = pd.to_datetime(df_user['birthdate'], format='%Y-%m-%d')
```

```
df_user.dtypes
```

	0
id	object
retirement_age	int64
birthdate	datetime64[ns]
gender	object
per_capita_income	object
yearly_income	object
total_debt	object
credit_score	int64

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on User Table*

5a.l.c. Change the data types ("per_capita_income", "yearly_income" and "total_debt") from object to float

```
df_user['per_capita_income'] = df_user['per_capita_income'].str.replace('Rp', '').str.replace('.', '', regex=False).astype(float)
df_user['yearly_income'] = df_user['yearly_income'].str.replace('Rp', '').str.replace('.', '', regex=False).astype(float)
df_user['total_debt'] = df_user['total_debt'].str.replace('Rp', '').str.replace('.', '', regex=False).astype(float)
```

```
df_user.dtypes
```

	0
id	object
retirement_age	int64
birthdate	datetime64[ns]
gender	object
per_capita_income	float64
yearly_income	float64
total_debt	float64
credit_score	int64

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on User Table*

5a.2. Check the unique values in each column and see some typo data that either needs to be fixed or excluded (use value_counts())

```
1  df_user.columns
```

```
Index(['id', 'retirement_age', 'birthdate', 'gender', 'per_capita_income',
       'yearly_income', 'total_debt', 'credit_score'],
      dtype='object')
```

Based on the information provided in the data dictionary, we can conclude the following :

- The id column is a row identifier and serves as the Primary Key, meaning it must not contain any null values or duplicates.
- The retirement_age column is a profile attribute, not an identifier.
- The birthdate column will naturally contain many duplicate values because multiple individuals can be born on the same date.
- The gender column is a demographic attribute, so duplicates are allowed however, potential typos or inconsistent labels should be checked.
- The per_capita_income, yearly_income, and total_debt columns represent financial values and are not identifiers, so duplicates are acceptable.
- The credit_score column represents a scoring metric, and it is normal for different users to have the same score.

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on User Table*

```
1  df_user['id'].value_counts()
```

id	count
1528	1
1735	1
1284	1
1805	1
1616	1
...	...
1164	1
708	1
1718	1
1746	1
825	1

2000 rows × 1 columns

1. All IDs Are Unique
   Each value in the id column appears exactly once (count = l), confirming that all 2000 rows have distinct identifiers. This validates that the id column functions as a proper Primary Key.
2. No Duplicates or Nulls. There are no duplicate or missing values in the id column, indicating strong data integrity. This ensures reliable operations for indexing, joining tables, or tracking user level behavior.
3. Well structured dataset
   With unique IDs, we can be confident that each row represents a single user entity, eliminating risks of double counting or incorrect data merges.

```
1  df_user['gender'].value_counts()
```

gender	count
Female	1016
Male	984

dtype: int64

No Missing or Unexpected Values
Only two categories are present : "Female" and "Male". There are no null values or unexpected entries like "Unknown", "Other", or typos.. This indicates good data quality for the gender column, since the gender distribution is balanced, demographic analysis (example transaction behavior, fraud risk, or product preferences by gender) can be conducted without bias toward one group.

DATA CLEANING, PREPARATION, AND COMMUNICATION *base on User Table*

Colab Link

5a.3. Check Missing Value

```
1   df_user.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 2000 non-null   object
 1   retirement_age     2000 non-null   int64
 2   birthdate          2000 non-null   datetime64[ns]
 3   gender             2000 non-null   object
 4   per_capita_income  2000 non-null   float64
 5   yearly_income      2000 non-null   float64
 6   total_debt         2000 non-null   float64
 7   credit_score       2000 non-null   int64
dtypes: datetime64[ns](1), float64(3), int64(2), object(2)
memory usage: 125.1+ KB
```

1. No Missing Values
   All columns contain 2000 non null entries, indicating that the dataset is complete and does not require any imputation or initial cleaning.
2. Appropriate Data Types
   - birthdate is correctly stored as datetime64[ns], making it suitable for age calculations and time based segmentation.
   - Financial columns like per_capita_income, yearly_income, and total_debt are in float64 format, ready for aggregation and analysis.
   - credit_score and retirement_age are in int64, appropriate for numerical modeling and classification.

5b. Add age, retired flag and DTI (debt to income ratio) column by using logics

5b.1. Add 'age' column by logics user's age from their 'birthdate' until the analysis cut off date as 31 May 2025

```
analysis_cutoff_date = pd.to_datetime('2025-05-31')
df_user['age'] = (analysis_cutoff_date - df_user['birthdate']).dt.days / 365.25
df_user['age'] = df_user['age'].astype(int)
display(df_user.head())
```

	id	retirement_age	birthdate	gender	per_capita_income	yearly_income	total_debt	credit_score	age
0	825	66	1972-11-25	Female	45937000.0	93663000.0	38138095.0	787	52
1	1746	68	1972-12-16	Female	59451000.0	121212000.0	57186095.0	701	52
2	1718	67	1944-11-04	Female	35586000.0	52535000.0	58666.0	698	80
3	708	63	1963-01-12	Female	255975000.0	392132000.0	60467238.0	722	62
4	1164	70	1982-09-21	Male	84407000.0	172099000.0	54946285.0	675	42

5b.2. Add 'retired_flag' column by logics user's whose age is not less than the 'retirement_age'.

```
import numpy as np
df_user['retired_flag'] = np.where(df_user['age'] >= df_user['retirement_age'],
1, 0)
display(df_user.head())
```

	id	retirement_age	birthdate	gender	per_capita_income	yearly_income	total_debt	credit_score	age	retired_flag
0	825	66	1972-11-25	Female	45937000.0	93663000.0	38138095.0	787	52	0
1	1746	68	1972-12-16	Female	59451000.0	121212000.0	57186095.0	701	52	0
2	1718	67	1944-11-04	Female	35586000.0	52535000.0	58666.0	698	80	1
3	708	63	1963-01-12	Female	255975000.0	392132000.0	60467238.0	722	62	0
4	1164	70	1982-09-21	Male	84407000.0	172099000.0	54946285.0	675	42	0

5b. Add age, retired flag and DTI (debt to income ratio) column by using logics

5b.3. Add 'DTI' column by logics the ratio between 'total_debt' and total income as 'yearly_income' in the data

```
df_user['DTI'] = df_user['total_debt'] / df_user['yearly_income']
display(df_user.head())
```

	id	retirement_age	birthdate	gender	per_capita_income	yearly_income	total_debt	credit_score	age	retired_flag	DTI
0	825	66	1972-11-25	Female	45937000.0	93663000.0	38138095.0	787	52	0	0.407184
1	1746	68	1972-12-16	Female	59451000.0	121212000.0	57186095.0	701	52	0	0.471786
2	1718	67	1944-11-04	Female	35586000.0	52535000.0	58666.0	698	80	1	0.001117
3	708	63	1963-01-12	Female	255975000.0	392132000.0	60467238.0	722	62	0	0.154201
4	1164	70	1982-09-21	Male	84407000.0	172099000.0	54946285.0	675	42	0	0.319271

The use or insight of create these new columns to transform raw data into meaningful analytical features that :

- Improve segmentation
- Enhance risk assessment
- Strengthen predictive modeling
- Provide deeper insights into user behavior
- Standardize comparisons across the dataset
- Standardizes age calculation
- Enable demographic analysis
- Supports feature engineering for modeling like credit scoring, churn prediction, fraud detection, and customer segmentation

TABLE SUMMARIZE AFTER CLEANING THE DATA “Card table”

Data Cleaning Steps	Original Data	After Cleaning	Reason to do this cleaning
Convert to the correct data type	1. id and client_id → integer data type 2. count_nonfraud_trx_L6M and count_fraud_trx_L6M → float data type 3. credit_limit, amt_nonfraud_trx_L6M and amt_fraud_trx → object data type 4. expires and acct_open_date → object data type	1. id and client_id → object data type 2. count_nonfraud_trx_L6M and count_fraud_trx_L6M → integer data type 3. credit_limit, amt_nonfraud_trx_L6M and amt_fraud_trx → float data type 4. expires and acct_open_date → timeseries	1. Ensure calculate calculations 2. Enables correct data operations 3. Improves data integrity
Check the unique values for looking the duplicates and typos	1. fixing the typo on card brand (Visa/Visa separate 2093 row and 69 row also JCB/Jcb 206 row and 3 row) 2. fixing the duplicate on id and card_number	1. after fixing typo, Visa has 2162 row and JCB has 209 row 2. id and card_number column has 5599 row before and after fixing has 5568 row	1. Improves data accuracy 2. Prevent double counting 3. Ensure consistency in categorial data 4. Enhances data integrity 5. Prevents errors in joins and merges
Check missing value (Option filling or remove the data)	1. credit_limit has 5587 row Non Null from total 5598 row, so this column has l2 row that Null	1. Filling missing value in credit_limit column with median helps	1. Cause the proportion of missing values in credit_limit is relatively small (l2 out of 5599 entries), we can filling missing values with the median helps maintain the natural structure of the dataset

TABLE SUMMARIZE AFTER CLEANING THE DATA “User table”

Data Cleaning Steps	Original Data	After Cleaning	Reason to do this cleaning
Convert to the correct data type	1. the data types "id" → int64 2. the data types "birthdate" → object 3. the data types "per_capita_income","yearly_income" and "total_debt") → object	1. Change the data types "id" from int64 to object 2. Change the data types "birthdate" from object to datetime 3. Change the data types ("per_capita_income", "yearly_income" and "total_debt") from object to float	1. Ensure calculate calculations 2. Enables correct data operations 3. Improves data integrity
Check the unique values for looking the duplicates and typos	1. The id column is a row identifier and serves as the Primary Key 2. The gender column is a demographic attribute, so duplicates are allowed however, potential typos or inconsistent labels should be checked.	1. The id column (All IDs Are Unique, No Duplicates or Nulls, and Well structured dataset 2. The gender column (No Missing and No null values or unexpected entries like "Unknown", "Other", or typos)	Indicating that the dataset is complete and does not require any imputation or initial cleaning.
Check missing value (Option filling or remove the data)	1. No Missing Values All columns contain 2000 non null entries 2. Appropriate Data Types (birthdate is correctly stored as datetime64[ns], financial columns like per_capita_income, yearly_income, and total_debt are in float64 format, and credit_score and retirement_age are in int64.	Indicating that the dataset is complete and does not require any imputation or initial cleaning.	

THE OTHERS SUMMARIZE AFTER CLEANING THE DATA

CARD TABLE	USER TABLE
Remove all cards that have already expired and also exclude cards with a credit limit 0 (data calculated as May 31, 2025), based on the card expiry date. The reason use this filter is to ensure cleaner, more reliable analysis, no noise from expired or inactive cards, and better accuracy for modeling and segmentation.	1. Add 'age' column by logics user's age from their 'birthdate' until the analysis cut off date as 31 May 2025 2. Add 'retired_flag' column by logics user's whose age is not less than the 'retirement_age'. 3. Add 'DTI' column by logics the ratio between 'total_debt' and total income as 'yearly_income' in the data Create these new columns to transform raw data into meaningful, analytical features that : • Improve segmentation • Enhance risk assessment • Strengthen predictive modeling • Provide deeper insights into user behavior • Standardize comparisons across the dataset

MILESTONE 2

EXPLORATORY DATA ANALYSIS (EDA)

Colab Link

DESCRIPTIVE ANALYSIS

6a.Calculate the total Net Profit from all transaction in the last 6 months through MDR (Merchant Discount Rate) 1.5%. (Hint : Net profit means all MDR fee profit minus all fraud amount. Calculate MDR fee profit by multiplying the MDR fe rate and total sales amount . Then, substract total fraud amount from that result).

```
1  mdr_fee_rate = 0.015
2  print(f"MDR Fee Rate set to: {mdr_fee_rate*100}%")
```

```
MDR Fee Rate set to: 1.5%
```

+ Kode + Teks

```
1  total_nonfraud_sales = df_card_filtered['amt_nonfraud_trx_L6M'].fillna(0).sum()
2  print(f"Total non-fraudulent sales in last 6 months: {total_nonfraud_sales}")
```

```
Total non-fraudulent sales in last 6 months: 455863486600.0
```

```
1  mdr_fee_profit = total_nonfraud_sales * mdr_fee_rate
2  print(f"MDR Fee Profit: {mdr_fee_profit}")
```

```
MDR Fee Profit: 6837952299.0
```

```
1  df_card_filtered['amt_fraud_trx_L6M'] = pd.to_numeric(df_card_filtered
   ['amt_fraud_trx_L6M'], errors='coerce').fillna(0)
2  total_fraud_amount = df_card_filtered['amt_fraud_trx_L6M'].sum()
3  print(f"Total fraud amount in last 6 months: {total_fraud_amount}")
```

```
Total fraud amount in last 6 months: 1013378300.0
/tmp/ipython-input-1493433811.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_card_filtered['amt_fraud_trx_L6M'] = pd.to_numeric(df_card_filtered['amt_fraud_trx_L6M'], errors='coerce').fillna(0)
```

```
1  net_profit = mdr_fee_profit - total_fraud_amount
2  print(f"Total Net Profit in last 6 months: {net_profit}")
```

```
Total Net Profit in last 6 months: 5824573999.0
```

```
1  print(f"The total net profit from all transactions in the last 6 months is:
   {net_profit}")
```

```
The total net profit from all transactions in the last 6 months is: 5824573999.0
```

INSIGHT FINDING

1. **Effectiveness of the MDR Fee**
   - With a 1.5% MDR (Merchant Discount Rate), the business generated **$6,837,952,299** in profit from **$455,863,486,600** in non-fraudulent sales.
   - This indicates that the MDR model is effective in generating revenue from legitimate transactions.
2. **Impact of Fraud on Profit**
   - A total of **$1,013,378,300** in fraudulent transactions directly reduced MDR profit.
   - After subtracting fraud, the **final net profit was $5,824,573,999**, meaning **approximately 14.8% of MDR profit was lost due to fraud**.
3. **Low Fraud Ratio**
   - Fraud accounted for only **0.22% of total non fraudulent sales**, suggesting strong fraud prevention and detection systems.
   - However, the absolute value of fraud remains significant due to its direct impact on profit.
4. **Optimization Potential**
   - Reducing fraud further could increase net profit without raising the MDR fee.
   - This presents an opportunity to invest in fraud prevention tools or transaction audits.
5. **Business Model Validation**
   - Even with a modest MDR fee of 1.5%, the business achieved multi billion rupiah net profit.
   - This confirms that high transaction volume is a key driver of profitability in this model.

The total net profit from all transactions in the last 6 months is 5,824,573,999.0

Key Findings

- The MDR (Merchant Discount Rate) fee was set at 1.5%.
- The total non-fraudulent sales in the last 6 months amounted to 455,863,486,600.0
- The calculated MDR fee profit was 6,837,952,299.0
- The total fraudulent amount recorded for the last 6 months was 1,013,378,300.0
- The final net profit, after subtracting fraud from MDR fee profit, was 5,824,573,999.0

EXPLORATORY DATA ANALYSIS (EDA)

DESCRIPTIVE ANALYSIS

6b.Calculate the fraud rateof RevoBank. Fraud rate is percentage of fraud amount from overall transaction volume.
(Hint : calculate total fraud and non fraud amount and divide total fraud by total all amounts to get the fraud rate.

```
total_transaction_volume = total_fraud_amount + total_nonfraud_sales
print(f"Overall transaction volume in last 6 months: {total_transaction_volume}
")
```

```
Overall transaction volume in last 6 months: 456876864900.0
```

```
fraud_rate = (total_fraud_amount / total_transaction_volume) * 100
print(f"Fraud rate of RevoBank in last 6 months: {fraud_rate:.2f}%")
```

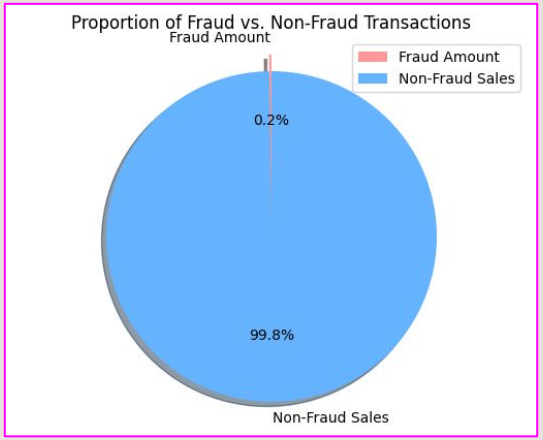
```
Fraud rate of RevoBank in last 6 months: 0.22%
```

INSIGHT FINDING

1. **Massive Transaction Volume**
   - RevoBank processed a total of **$456,876,864,900** in transactions, reflecting a large scale operation.
   - This suggests strong user trust and high financial activity across the platform.
2. **Exceptionally Low Fraud Rate**
   - The fraud rate was just **0.22%**, which is impressively low given the transaction volume.
   - Indicates that RevoBank fraud detection and prevention systems are highly effective.
3. **Nominal Fraud Still Has Financial Impact**
   - Despite the low percentage, the fraud amount could exceed **$1 billion**, based on the total volume.
   - Even small improvements in fraud prevention could lead to significant gains in profitability.
4. **Operational and System Strength**
   - High volume combined with low fraud validates the robustness of RevoBank transaction infrastructure.
   - This can be a strategic advantage when attracting investors or business partners.

Key Findings

- The overall transaction volume for RevoBank in the last 6 months was $456,876,864,900.0.
- The fraud rate for RevoBank in the last 6 months was 0.22%.



Colab Link

EXPLORATORY DATA ANALYSIS (EDA)

Colab Link

DESCRIPTIVE ANALYSIS

6c.See any difference in transaction behaviour per card brand. (Hint : group by card brand and calculate average of transaction count and amount).

```
1  df_card_brand_avg = df_card_filtered.groupby('card_brand')
   [['count_nonfraud_trx_L6M', 'amt_nonfraud_trx_L6M', 'count_fraud_trx_L6M',
   'amt_fraud_trx_L6M']].mean()
2  display(df_card_brand_avg)
```

card_brand	count_nonfraud_trx_L6M	amt_nonfraud_trx_L6M	count_fraud_trx_L6M	amt_fraud_trx_L6M
Amex	112.941624	1.492264e+08	0.126904	207462.944162
JCB	84.395122	1.114518e+08	0.136585	185500.000000
Mastercard	121.933523	1.209153e+08	0.118379	191013.579808
Visa	116.055293	1.243378e+08	0.117202	168378.638941

INSIGHT FINDING

1. Non-Fraudulent Transactions :

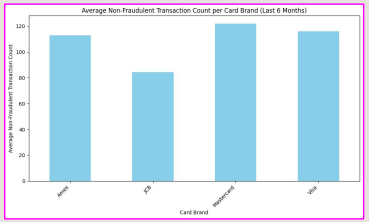
- **Mastercard** generally shows the highest average non-fraudulent transaction count, indicating a higher frequency of use for legitimate purchases.
- In terms of average non-fraudulent transaction amount, **Amex** appears to have the highest average value, suggesting that Amex cards are used for larger purchases on average. Mastercard and Visa follow with comparable average amounts, while JCB has the lowest.

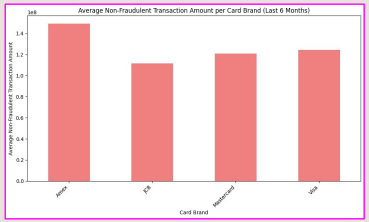
2. Fraudulent Transactions :

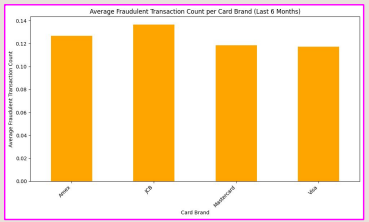
- **Amex** tends to have a slightly higher average fraudulent transaction count, though the differences across brands are not extremely pronounced. This could suggest Amex cards are targeted more frequently for fraudulent attempts.
- Similar to non-fraudulent transactions, **Amex** also shows the highest average fraudulent transaction amount. This implies that when fraud occurs with Amex cards, it often involves higher individual transaction values.

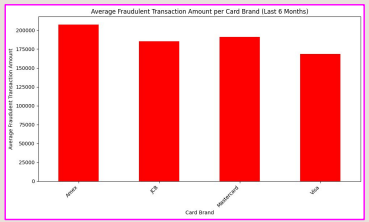
Overall Insights :

- There's a notable pattern where **Amex** cards, despite having a lower frequency of non-fraudulent transactions compared to Mastercard, tend to have higher average transaction values for both non fraudulent and fraudulent activities. This could indicate that Amex users generally conduct higher value transactions, which also translates to higher value fraud incidents.
- **Mastercard** shows the highest volume (count) of non-fraudulent transactions, but its average transaction amount (both non-fraud and fraud) is not the highest, suggesting a broad user base engaging in a mix of transaction sizes.
- **Visa** is consistently in the mid-range for both counts and amounts, reflecting its widespread acceptance and diverse user base.
- **JCB** generally exhibits the lowest average counts and amounts for both types of transactions, which might be due to its more limited market presence compared to the other major brands. buat bahsa Indonesia nya









EXPLORATORY DATA ANALYSIS (EDA)

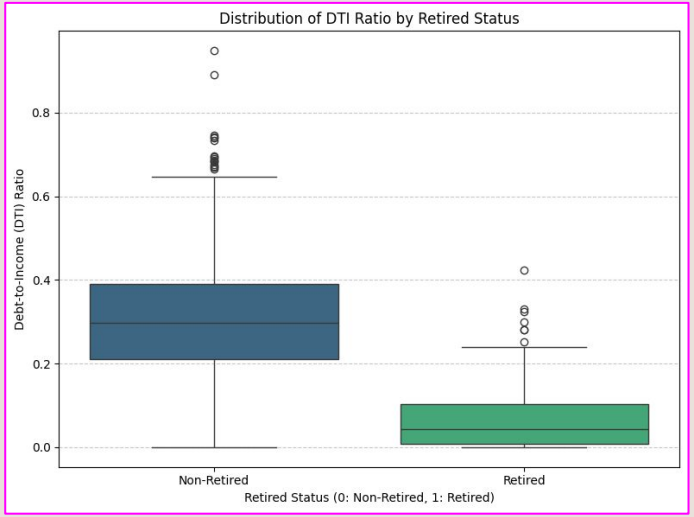
6d.Compare the debt to income ratios of retired vs non retired users. The debt to income ratio definition is the total debt per yearly income (Hint: Group data by retired_flag and use agregate statistics to check the difference in debt to income ratios between retired vs non retired users, by need to calculate the debt to income ratio first.

```
1  dti_stats = df_user.groupby('retired_flag')['DTI'].agg(['mean', 'median',
   'std'])
2  display(dti_stats)
```

retired_flag	mean	median	std
0	0.295747	0.297170	0.153898
1	0.067158	0.044024	0.070973

INSIGHT FINDING

- Non-retired users exhibit a significantly higher Debt-to-Income (DTI) ratio, with a mean of approximately 0.296 and a median of 0.297. Their DTI is roughly 4-5 times higher than that of retired users. Retired users have substantially lower DTI ratios, with a mean of approximately 0.067 and a median of 0.044, indicating a reduced debt burden.
- The DTI distribution for non-retired users shows a wider interquartile range (IQR) and more high-value outliers, suggesting a greater spread and instances of exceptionally high DTI.
- The DTI distribution for retired users has a much narrower IQR and fewer, lower outliers, indicating a more consistent and lower debt burden across this group.
- The boxplot visually confirms that the DTI ratios for retired individuals are clustered significantly lower on the y-axis compared to non-retired individuals.



Merge The Card Data and User Data

Merge the card data dan user data. Make sure data is grouped by client ID (1 user can have multiple cards), recency data takes the most recent number (choose the smallest number), sum all the monetary and count columns, and exclude all non important column.

```python
df_card_agg = df_card_filtered.groupby('client_id').agg(
    credit_limit=('credit_limit', 'sum'),
    count_nonfraud_trx_L6M=('count_nonfraud_trx_L6M', 'sum'),
    amt_nonfraud_trx_L6M=('amt_nonfraud_trx_L6M', 'sum'),
    count_fraud_trx_L6M=('count_fraud_trx_L6M', 'sum'),
    amt_fraud_trx_L6M=('amt_fraud_trx_L6M', 'sum'),
    recency=('days_since_last_trx', 'min')
).reset_index()

display(df_card_agg.head())
display(df_card_agg.info())
```

	client_id	credit_limit	count_nonfraud_trx_L6M	amt_nonfraud_trx_L6M	count_fraud_trx_L6M	amt_fraud_trx_L6M	recency
0	0	165775000.0	685	535262100.0	0	0.0	3
1	1	65592000.0	498	264007900.0	0	0.0	20
2	10	108249000.0	0	0.0	0	0.0	604
3	100	236386000.0	366	295701900.0	0	0.0	5
4	1000	36225000.0	0	0.0	0	0.0	604

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1938 entries, 0 to 1937
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   client_id               1938 non-null   object
 1   credit_limit            1938 non-null   float64
 2   count_nonfraud_trx_L6M  1938 non-null   int64
 3   amt_nonfraud_trx_L6M    1938 non-null   float64
 4   count_fraud_trx_L6M     1938 non-null   int64
 5   amt_fraud_trx_L6M       1938 non-null   float64
 6   recency                 1938 non-null   int64
dtypes: float64(3), int64(3), object(1)
memory usage: 106.1+ KB
None
```

Colab Link

The next step is to merge the aggregated card data (df_card_agg) with the user data (df_user) based on the common client identifier. This will combine the card related metrics with the user's demographic and financial information.

```python
df_merged = pd.merge(df_user, df_card_agg, left_on='id', right_on='client_id',
how='inner')

print("First 5 rows of the merged DataFrame:")
display(df_merged.head())

print("\nInformation about the merged DataFrame:")
df_merged.info()
```

First 5 rows of the merged DataFrame:

	id	retirement_age	birthdate	gender	per_capita_income	yearly_income	total_debt	credit_score	age	retired_flag	DTI	client_id	credit_limit	cou
0	825	66	1972-11-25	Female	45937000.0	93663000.0	38138095.0	787	52	0	0.407184	825	164867000.0	
1	1746	68	1972-12-16	Female	59451000.0	121212000.0	57186095.0	701	52	0	0.471786	1746	102033000.0	
2	1718	67	1944-11-04	Female	35586000.0	52535000.0	58666.0	698	80	1	0.001117	1718	209210000.0	
3	708	63	1963-01-12	Female	255975000.0	392132000.0	60467238.0	722	62	0	0.154201	708	558975000.0	
4	1164	70	1982-09-21	Male	84407000.0	172099000.0	54946285.0	675	42	0	0.319271	1164	54758000.0	

```
Information about the merged DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1938 entries, 0 to 1937
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     1938 non-null   object
 1   retirement_age         1938 non-null   int64
 2   birthdate              1938 non-null   datetime64[ns]
 3   gender                 1938 non-null   object
 4   per_capita_income      1938 non-null   float64
 5   yearly_income          1938 non-null   float64
 6   total_debt             1938 non-null   float64
 7   credit_score           1938 non-null   int64
 8   age                    1938 non-null   int64
 9   retired_flag           1938 non-null   int64
 10  DTI                    1938 non-null   float64
 11  client_id              1938 non-null   object
 12  credit_limit           1938 non-null   float64
 13  count_nonfraud_trx_L6M 1938 non-null   int64
 14  amt_nonfraud_trx_L6M   1938 non-null   float64
 15  count_fraud_trx_L6M    1938 non-null   int64
 16  amt_fraud_trx_L6M      1938 non-null   float64
 17  recency                1938 non-null   int64
dtypes: datetime64[ns](1), float64(7), int64(7), object(3)
memory usage: 272.7+ KB
```

The Result of Merge

The successful merge of the card and user data resulted in df_merged, a DataFrame combines user specific information from df_user with aggregated card activity data from df_card_agg.

The aggregated columns from the card data now present in df_merged are: credit_limit (summed), count_nonfraud_trx_L6M (summed), amt_nonfraud_trx_L6M (summed), count_fraud_trx_L6M (summed), amt_fraud_trx_L6M (summed), and recency (minimum days_since_last_trx).

- The df_card_filtered DataFrame was successfully aggregated by client_id, resulting in df_card_agg with 1938 entries and 7 columns.
- The aggregation involved summing credit_limit, count_nonfraud_trx_L6M, amt_nonfraud_trx_L6M, count_fraud_trx_L6M, and amt_fraud_trx_L6M for each client.
- The minimum days_since_last_trx was calculated and renamed to recency for each client.
- The df_user DataFrame was successfully merged with df_card_agg using an inner join on df_user['id'] and df_card_agg['client_id'].
- The final df_merged DataFrame contains 1938 entries and 18 columns, integrating both user demographics and the aggregated card transaction metrics.

Condition for next step :

- The df_merged DataFrame now provides a comprehensive view of each client, combining demographic information with their aggregated card usage and fraud-related statistics, which is crucial for client-centric analysis.
- This integrated dataset is ready for further analysis, such as building predictive models for customer behavior, fraud detection, or credit risk assessment, by leveraging the combined features.

EXPLORATORY DATA ANALYSIS (EDA)

Colab Link

6e.Calculate the average of Debt to income ratio (DTI) based on credit score category (poor, fair, good, very good, exceptional). Hint: Use pd.cut to categorise the credit score into different scoring also add the breakdown of the transaction behaviour (transaction count and transaction amount).

```
bins = [0, 579, 669, 739, 799, 850]
labels = ['Poor', 'Fair', 'Good', 'Very Good', 'Exceptional']

print(f"Credit Score Bins: {bins}")
print(f"Credit Score Labels: {labels}")
```

```
Credit Score Bins: [0, 579, 669, 739, 799, 850]
Credit Score Labels: ['Poor', 'Fair', 'Good', 'Very Good', 'Exceptional']
```

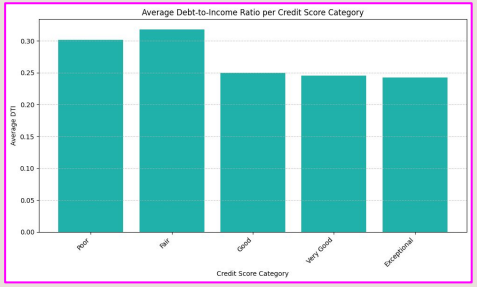
Now that the bins and labels for credit score categories are defined, use 'pd.cut' to categorize the 'credit_score' column in 'df_merged' into these defined categories and store them in a new column called 'credit_score_category'.

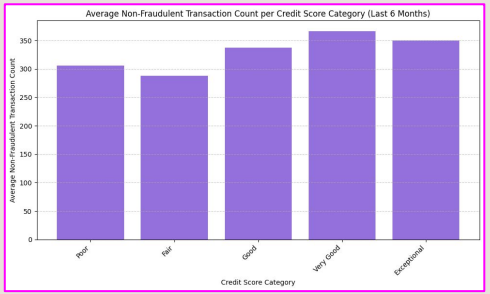
	credit_score_category	avg_dti	avg_count_nonfraud_trx	avg_amt_nonfraud_trx	avg_count_fraud_trx	avg_amt_fraud_trx
0	Poor	0.301839	305.480519	2.290265e+08	0.272727	309531.168831
1	Fair	0.318218	287.595092	2.019485e+08	0.312883	467417.177914
2	Good	0.250306	337.049288	2.419351e+08	0.342826	554249.288061
3	Very Good	0.245725	366.563043	2.478267e+08	0.371739	532647.826087
4	Exceptional	0.242909	350.302469	2.315197e+08	0.320988	531597.530864

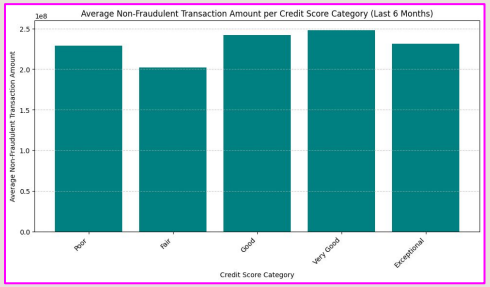
INSIGHT FINDING

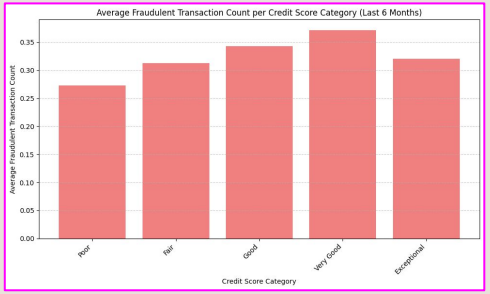
The key insights and differences in DTI ratios and transaction behavior across the various credit score categories are as follows:

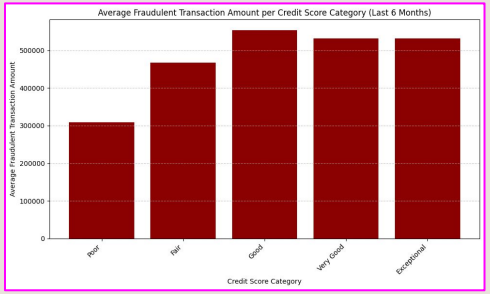
- **Debt-to-Income (DTI) Ratio :** There's an inverse relationship between credit score and DTI. The 'Fair' category exhibits the highest average DTI (approximately 0.318), followed closely by 'Poor' (approximately 0.302). DTI progressively decreases with higher credit scores, with 'Exceptional' having the lowest average DTI (approximately 0.243), indicating better debt management.
- **Non-Fraudulent Transaction Behavior :** Users with higher credit scores ('Good', 'Very Good', 'Exceptional') demonstrate more active legitimate transaction behavior. The 'Very Good' category shows the highest average non-fraudulent transaction count (approximately 366.56) and amount (approximately $247,800,000). Conversely, 'Poor' and 'Fair' categories have lower average counts and amounts.
- **Fraudulent Transaction Behavior :** While average fraudulent transaction counts are low across all categories, there's a slight increase from 'Poor' to 'Good' and 'Very Good', with 'Very Good' having the highest average count (approximately 0.372). Similarly, 'Good' and 'Exceptional' categories show slightly higher average fraudulent transaction amounts (around $554,000 and $531,000, respectively), suggesting that when fraud occurs in these segments, it might involve larger values.











EXPLORATORY DATA ANALYSIS (EDA)

6f.Present the findings in a line chart to see the relationship between user age and credit limit (Hint:Sum the credit limit by user age then devide by total number, trim the user age to prevent small and volatile data in the higher age).

Aggregate the 'df_merged' DataFrame by 'age' to calculate the average 'credit_limit' for each age, trimming the age range to exclude very old ages to prevent volatile data. Prepare the data for visualization as requested, group the 'df_merged' DataFrame by 'age' and calculate the mean of 'credit_limit'.

```
1  avg_credit_limit_by_age_trimmed = avg_credit_limit_by_age
   [avg_credit_limit_by_age['age'] <= 85]
2  display(avg_credit_limit_by_age_trimmed.head())
3  display(avg_credit_limit_by_age_trimmed.tail())
```

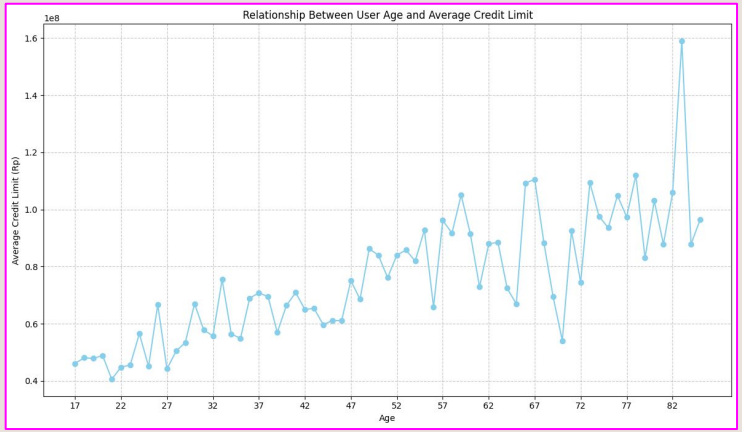
	age	credit_limit
0	17	4.609672e+07
1	18	4.808706e+07
2	19	4.789100e+07
3	20	4.886413e+07
4	21	4.057080e+07

	age	credit_limit
64	81	8.785650e+07
65	82	1.058914e+08
66	83	1.590255e+08
67	84	8.785744e+07
68	85	9.649889e+07

INSIGHT FINDING

Based on the line chart showing the average credit limit across different user ages, the following key observations can be made:

- **General Trend** : There appears to be a general upward trend in average credit limit as age increases, particularly from younger age groups up to a certain point.
- **Peak Credit Limit** : The average credit limit seems to peak around the middle-aged categories, often between 40-60 years, before potentially plateauing or slightly decreasing in very old age groups. This indicates that financial institutions tend to offer higher credit limits to individuals in their prime earning years, who have likely accumulated more assets and established stronger credit histories.
- **Younger Users** : Younger users (20s and early 30s) generally have lower average credit limits, which is expected as they are typically still building their careers and credit profiles.
- **Older Users (beyond peak)** : While the graph is trimmed at age 85 to prevent volatility, within the displayed range, credit limits for older users generally remain high or show a slight decline, possibly reflecting a shift in financial needs or reduced income post-retirement.



SUMMARY AND RECOMMENDATION

1. Contribution of Low Fraud Rate to Profit (Sections 6a and 6b)

   Insight : RevoBank has a remarkably low fraud rate of 0.22% with a net profit of approximately 5.82 billion ($). This low fraud rate directly minimizes financial losses from fraudulent transactions, which is crucial for profitability.

   Recommendation : A low fraud rate ensures that the majority of transaction revenue is retained as profit, rather than being eroded by fraud losses. This focus is highly effective as it protects revenue streams and maintains customer trust, indirectly encouraging more transaction volume.

2. DTI and Credit Scores for Lending Decisions (Sections 6d and 6e)

   Insight :

   2.1. Retired vs. Non-Retired DTI (6d) : Retired users have significantly lower DTI ratio (mean ~0.067) compared to non-retired users (mean ~0.296). This suggests retired users generally pose a lower debt related risk.

   2.2. Credit Score Categories, DTI, and Transaction Behavior (6e) : There's a clear inverse relationship between credit score and DTI; higher credit scores correlate with lower DTI. Moreover, users with 'Very Good' and 'Exceptional' credit scores show higher non-fraudulent transaction counts and amounts, indicating they are more active and profitable customers.

   Recommendation : Understanding DTI and credit scores is paramount for informed lending decisions.

   a. By segmenting customers based on these metrics, RevoBank can reduce risk Identify and mitigate risks by potentially offering lower credit limits or more stringent terms to customers with high DTI or poor credit scores.
   b. Increase Profitable Customers by target marketing and higher credit limit offers to customers with excellent credit scores and low DTI, as these individuals are more likely to generate higher transaction volumes and amounts, contributing significantly to MDR fee profits with lower default risk.

SUMMARY AND RECOMMENDATION

3. Leveraging Transaction Behavior Insights (Sections 6c and 6e)

Insight :

- Card Brand Behavior (6c)

  Amex cards exhibit the highest average non-fraudulent transaction amounts, suggesting they are used for larger purchases. Mastercard has the highest frequency of non-fraudulent transactions. Interestingly, Amex also shows higher average fraudulent transaction amounts.
- Credit Score and Transaction Behavior (6e)

  Users with 'Very Good' credit scores are not only the most active (highest count) but also have the highest average non-fraudulent transaction amounts.

Recommendation :

- Targeted Marketing : Promote specific card brands to segments that align with their transaction patterns (example : Amex for high-value spenders, Mastercard for frequent, everyday use).
- Fraud Prevention : Given Amex's higher average fraudulent amounts, dedicated fraud monitoring for Amex transactions could yield significant loss reduction.
- Customer Retention : Focus on retaining high-value, high-frequency customers (example : 'Very Good' credit score holders) through loyalty programs or premium services.

4. Relationship Between User Age and Credit Limit (Section 6f)

Insight : There's a general upward trend in average credit limits with age, peaking around middle age (40-60 years), and then plateauing or slightly declining for older users. Younger users have lower average limits.

Recommendation : This aligns with traditional credit risk models and income progression. It validates RevoBank's approach to credit limit allocation, ensuring that higher limits are extended to customers who have established financial stability and earning potential. This indirectly improves profit by minimizing default risk on larger credit lines.

MILESTONE 3

CLUSTER ANALYSIS

A. The reasoning why the K-Means Clustering method is selected

ASPECT	RFM Analysis	K-Means Clustering
Methodology	Based on three fixed metrics : Recency, Frequency, Monetary. Customers are scored and grouped manually.	Uses clustering group customers based on multiple variables (transactions, credit limit, DTI, income, etc.).
Flexibility	Limited to RFM dimensions and cannot easily incorporate other variables.	Highly flexible, can include any number of behavioral, demographic, or financial features.
Bias	Risk of arbitrary or subjective grouping.	Minimizes human bias, finds natural clusters in data.
Granularity	Produces broad segments (like “High RFM,” “Low RFM”).	Produces nuanced clusters that reflect actual customer behavior.
Actionable Insight	RFM tells you who is “recent, frequent, or high value.”	K-means reveals hidden customer groups (like “high spender but risky,” or “low spender but safe”), which are more actionable.
Scalability	Works well for small datasets.	Scales effectively for large datasets and complex variables.

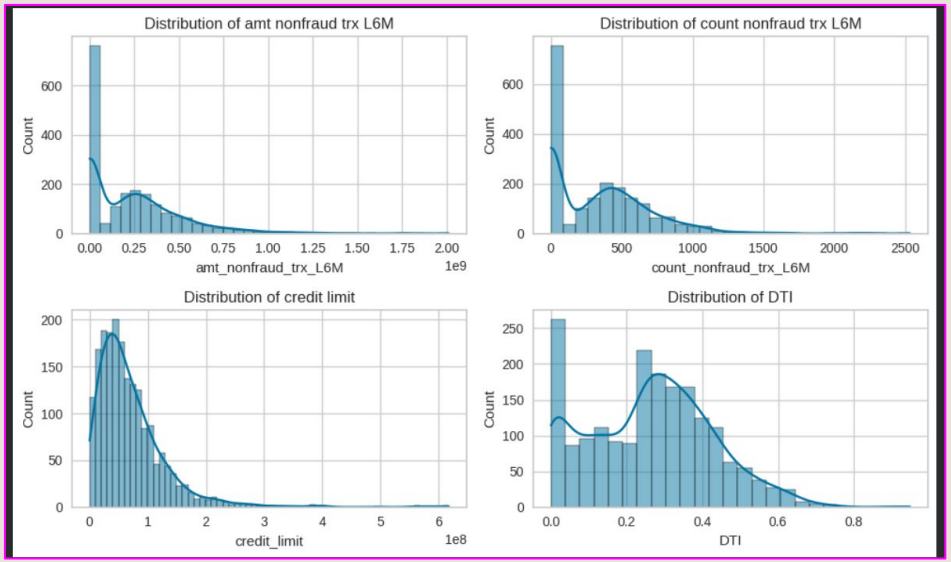
B. The reasoning why the column is selected

The Column	Reasoning
amt_nonfraud_trx_L6M	• Measures actual spending power : Reflects how much customers truly spend using their cards. • Direct profitability indicator : Higher transaction amounts translate into greater potential revenue from fees and interest. • Noise reduction : Focusing on non-fraud transactions ensures cleaner, more reliable behavioral data.
count_nonfraud_trx_L6M	• Captures activity and engagement : Shows customers are active or occasional users. • Loyalty & habit indicator : Customers with consistent transaction patterns are easier to retain and grow. • Complementary to amount : Helps distinguish between “big spenders with few transactions” versus “small spenders with frequent transactions.”
credit_limit	• Represents maximum spending capacity : Defines of how much a customer can spend. • Upsell opportunity : Customers with high limits but low usage can be targeted with promotions to increase activity. • Risk management factor : Credit limits are tied to risk exposure, so segmentation must balance potential profit with safety.
DTI (Debt-to-Income Ratio)	• Measures financial health : Indicates how much debt a customer carries relative to their income. • Risk filter : High DTI customers are riskier, while low DTI customers are safer for upselling or limit increases. • Profit vs risk balance : Ensures segmentation strategies are sustainable and not overly aggressive. • Credit risk dimension : Complements transaction behavior by adding a risk perspective.

STEP K-MEANS CLUSTERING

Colab Link

A. DATA PREPROCESSING
A.l. Distribution Check



l. amt_nonfraud_trx_L6M
- The distribution is heavily right skewed → most customers have low transaction amounts, with a few high-value outliers.
- Indicates that a small group contributes significantly to total revenue.

2. count_nonfraud_trx_L6M
- Also right skewed, but with a wider spread.
- Many customers transaction infrequently, while a subset is highly active.

3. credit_limit
- Smooth right skewed distribution → most customers have low to moderate credit limits.
- A small group holds very high limits.

4. DTI – Debt-to-Income Ratio
- The distribution appears bimodal or widely spread, with peaks at low and moderate DTI levels.
- Some customers have DTI = 0 → very low credit risk.

A.2. Applying Scaling Method

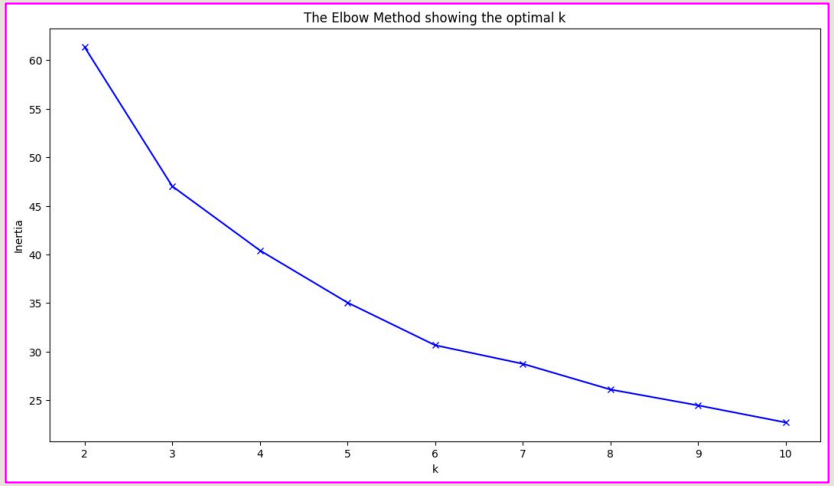
```python
from scipy.stats import shapiro, skew

def determine_scaler(data, column_name):
    # Perform Shapiro-Wilk test for normality and calculate skewness
    stat, p_value = shapiro(data[column_name])
    skewness = skew(data[column_name])

    # Print results for debugging
    print(f"{column_name} - Shapiro-Wilk Test: Statistic={stat}, p-value={p_value}")
    print(f"{column_name} - Skewness: {skewness}")
```

```
amt_nonfraud_trx_L6M - Shapiro-Wilk Test: Statistic=0.8266999771030535, p-value=1.369080023968494e-41
amt_nonfraud_trx_L6M - Skewness: 1.5673495371752115
count_nonfraud_trx_L6M - Shapiro-Wilk Test: Statistic=0.853345369640613, p-value=3.894318813234597e-39
count_nonfraud_trx_L6M - Skewness: 1.214398089050083
credit_limit - Shapiro-Wilk Test: Statistic=0.7667417937187556, p-value=3.772199036085486e-46
credit_limit - Skewness: 3.060395652602959
DTI - Shapiro-Wilk Test: Statistic=0.9721863852426098, p-value=6.244880265281098e-19
DTI - Skewness: 0.1862232300989867
Recommended Scaler for amt nonfraud trx L6M: RobustScaler
Recommended Scaler for count nonfraud trx L6M: RobustScaler
Recommended Scaler for credit limit: RobustScaler
Recommended Scaler for DTI: MinMaxScaler
```

B. DETERMINE CLUSTER NUMBER
B.1. Elbow Method



✅ Insight & Interpretation

- The inertia curve shows a sharp decline up to k = 6, after which the curve flattens. The most noticeable "elbow" appears at k = 6, traditionally interpreted as the optimal number of clusters based on inertia reduction.
- Use k = 3 for segmentation tasks that require clearly defined and interpretable clusters.
- Consider k = 6 if the goal is to capture more nuanced patterns in the data, even if the clusters are less distinct.

B.2. Silhouette Analysis

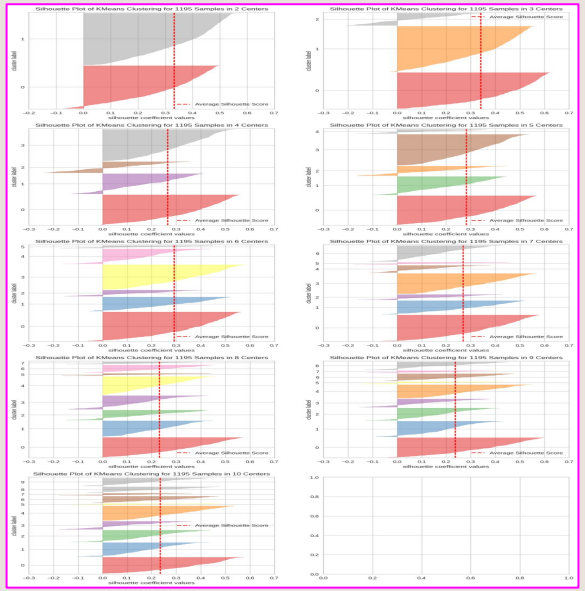
```python
from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer
import numpy as np
import matplotlib.pyplot as plt

K_range = range(2,11)
banyak_K = len(K_range)
n_baris = int(np.ceil(banyak_K/2))

fig, ax = plt.subplots(n_baris, 2, figsize=(15,30))

for index, K in enumerate(K_range):
    kmeanModel = KMeans(n_clusters=K, random_state=1000, n_init='auto')
    sil = SilhouetteVisualizer(kmeanModel, is_fitted=False, ax=ax.flatten()
    [index])
    sil.fit(df_segmentation_normalized)
    sil.finalize()
    print(f"For k={K}, the average silhouette score is {sil.silhouette_score_}")
```

```
For k=2, the average silhouette score is 0.33275344682480834
For k=3, the average silhouette score is 0.3403299730392498
For k=4, the average silhouette score is 0.26542059431711756
For k=5, the average silhouette score is 0.28311067883714736
For k=6, the average silhouette score is 0.29143855882955105
For k=7, the average silhouette score is 0.27010428848142676
For k=8, the average silhouette score is 0.23106630118159655
For k=9, the average silhouette score is 0.23649608521454032
For k=10, the average silhouette score is 0.23468904556223874
```



Colab Link

- The highest silhouette score is at **k = 3** with a value of **0.3403**.
  This is followed by **k = 2** with a score of **0.3327**.
  The score for **k = 4** is **0.2654**, which is significantly lower compared to k = 2 and k = 3.
  Scores for **k = 5 through k = 10** are all lower, indicating less well-defined clusters.
- Considering only the silhouette scores, **k = 3 is the best choice** as it has the highest silhouette score, suggesting that the clusters are the most compact and well-separated at this number of clusters.

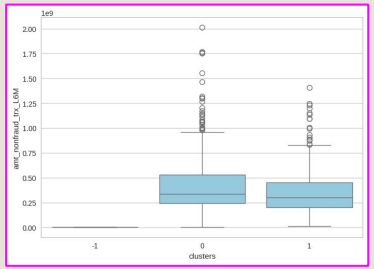
K-MEANS CLUSTERING MODELLING

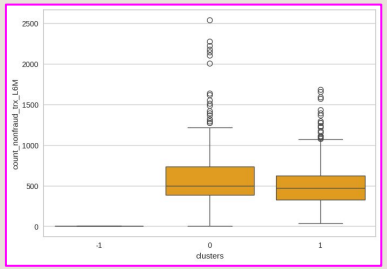
C.1. K-Means Clustering Modelling with 2 Clusters

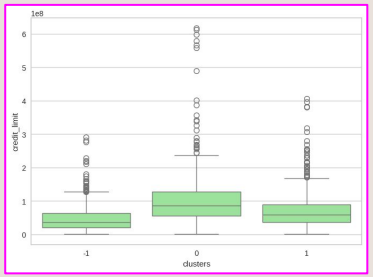
C.1.a. Cluster Distribution in amt nonfraud trx L6M, count nonfraud trx L6M, credit limit, and DTI

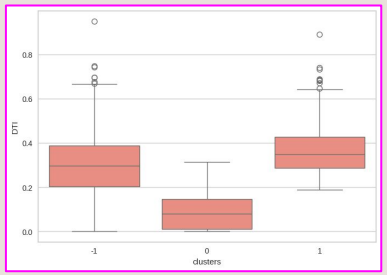
```
1  from sklearn.cluster import KMeans
2
3  cluster_model = KMeans(n_clusters=2, random_state=1000, n_init='auto')
4  cluster_model.fit(df2_norm)
```

```
KMeans
KMeans(n_clusters=2, random_state=1000)
```



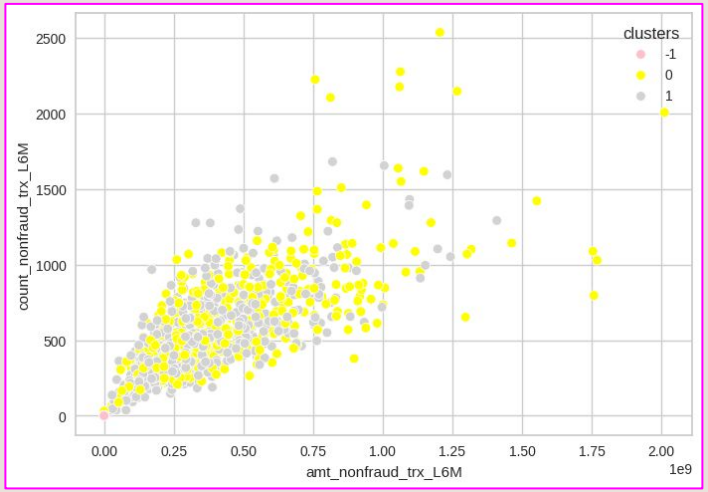






- **amt_nonfraud_trx_L6M (Total non-fraud transaction amount in the last 6 months)**
  a. Cluster -1 : very low transaction values → passive/dormant segment.
  b. Clusters 0 & 1 : high transaction values with many outliers → active segment, especially cluster 1 as high value.
- **count_nonfraud_trx_L6M (Number of non-fraud transactions in the last 6 months)**
  a. Cluster -1 : very low transaction counts → inactive customers.
  b. Clusters 0 & 1 : median around 500 transactions → active customers with high frequency.
- **credit_limit**
  a. Cluster -1 : low credit limit, almost no outliers → high-risk or limited-access segment.
  b. Clusters 0 & 1 : much higher credit limits, with cluster 1 having a higher median → high-value/premium segment.
- **DTI (Debt-to-Income ratio)**
  a. Cluster -1 : low DTI → low credit risk, conservative profile.
  b. Clusters 0 & 1 : higher and more variable DTI, with cluster 1 higher → segment with greater potential credit risk.

C.1.b. Cluster Visualisation



The scatterplot for the 2 cluster segmentation has been successfully generated. The visualization shows the distribution of two clusters based on **amt_nonfraud_trx_L6M** (X-axis) and **count_nonfraud_trx_L6M** (Y-axis).

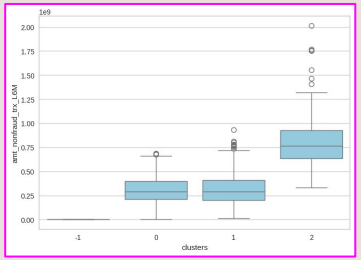
- **Yellow cluster (Cluster 0) :** Tends to have lower non-fraudulent transaction values and fewer transaction frequencies. This may represent a segment of less active customers or those with higher risk profiles, as concluded from the previous boxplot analysis (lower engagement, higher DTI).
- **Grey cluster (Cluster l) :** Shows higher non-fraudulent transaction values and greater transaction frequencies. This cluster likely represents highly engaged customers who perform large transactions and demonstrate better debt management (higher engagement, lower DTI).
- **Pink points (Cluster -l) :** Represent customers not included in these clusters due to initial filtering criteria (e.g., amt_nonfraud_trx_L6M or count_nonfraud_trx_L6M equal to zero). This cluster is often considered passive or inactive customers.

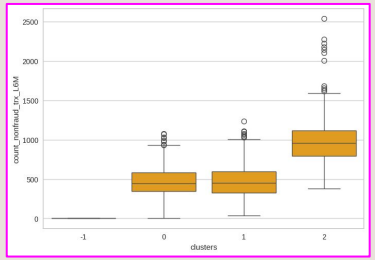
C.2. K-Means Clustering Modelling with 3 Clusters

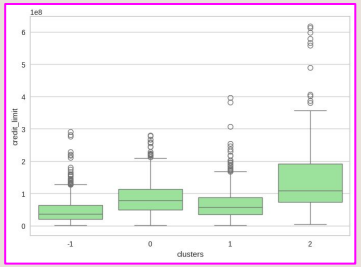
C.2.a. Cluster Distribution in amt nonfraud trx L6M, count nonfraud trx L6M, credit limit, and DTI

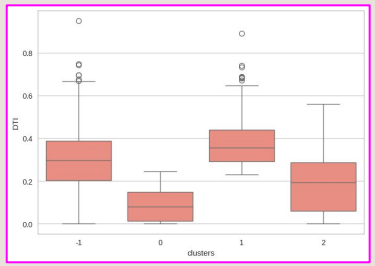
```
KMeans: cluster_model
sklearn.cluster._kmeans.KMeans instance
cluster_model.fit(df3_norm)
```

```
KMeans
KMeans(n_clusters=3, random_state=1000)
```









Clusters 0 & 1 : Regular Active Segment

- Transactions (amount & count) : moderate median (~500 transactions), relatively high activity with variation.
- Credit limit : medium → better credit access compared to cluster -1.
- DTI : medium → variation in debt management, with outliers in cluster 1 indicating higher-risk customers.

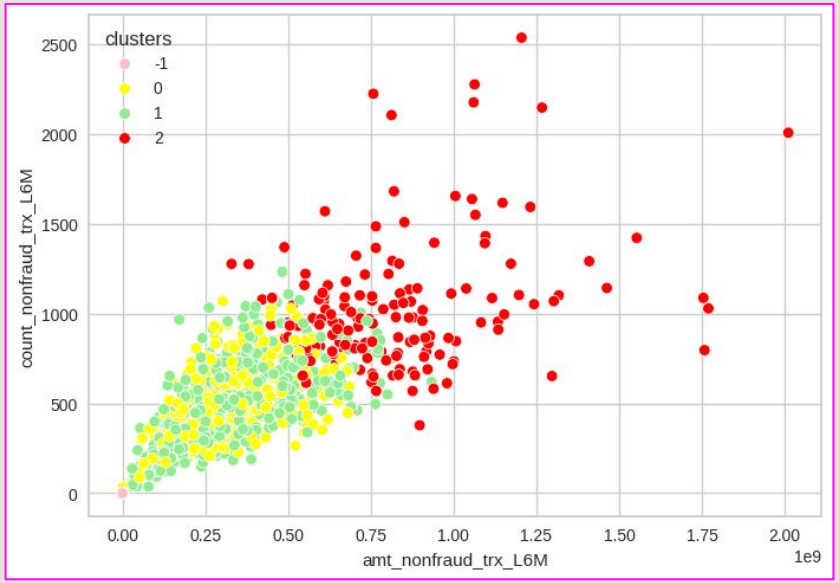
➡️ Characteristics : regular active customers, suitable for loyalty programs, mid-tier product upselling, and risk monitoring.

Cluster 2 : Premium / Priority Segment

- Transactions (amount & count) : highest, widest distribution → heavy users.
- Credit limit : highest, with many outliers → very high credit access.
- DTI : highest, more variable → significant debt burden.

➡️ Characteristics : premium customers, major contributors, suitable for exclusive services, but requiring strict credit risk control.

C.2.b. Cluster Visualisation



Colab Link

1. **Cluster 2 (Red) : Premium Segment**
   - Positioned in the upper right area → high transaction value and frequency.
   - Represents high value and highly active customers.
   - Suitable for exclusive services and priority retention, but requires risk monitoring due to high DTI.
2. **Cluster 1 (Green) : High Value Active Segment**
   - Spread across the mid-to-upper area → high frequency and moderately high transaction value.
   - Indicates active customers with strong potential value, but with varying risk levels (DTI higher than cluster 0).
   - Ideal for loyalty programs and upselling, with credit oversight.
3. **Cluster 0 (Yellow) : Regular Active Segment**
   - Positioned in the middle area → moderate transaction frequency and value.
   - Represents regular active customers, suitable for mid-tier product offerings and engagement enhancement.

COMPARISON BETWEEN 2 AND 3 CLUSTERS based on K-Means Clustering

Aspect	2 Clusters	3 Clusters
Elbow Method	suggests that the optimal number of clusters is around 3 or 6, where the decrease in inertia starts to slow down significantly.	
Silhouette Analysis	The average silhouette score for 2 clusters is approximately 0.3327	The average silhouette score for 3 clusters is approximately 0.3403
Clusters Distribution (Box Plots)	The box plots show a good separation for amt_nonfraud_trx_L6M, count_nonfraud_trx_L6M, and credit_limit, with one cluster having significantly higher values. For DTI, there's some separation, but also considerable overlap.	The box plots for 3 clusters show a more granular and distinct separation across amt_nonfraud_trx_L6M, count_nonfraud_trx_L6M, and credit_limit. You can observe more varied groups, such as low activity/limit, medium activity/limit, and high activity/limit segments. For DTI, the separation also appears more nuanced.
Decided and Reasoning	1. While 2 clusters offer a slightly higher silhouette score, indicating marginally better overall separation, 3 clusters appear to provide a more meaningful and actionable segmentation for business insights (The box plots for 3 clusters reveal more distinct subgroups that can be more useful for targeted strategies, The elbow method also supports 3 clusters as a reasonable choice) 2. Therefore, 3 clusters would generally be preferred for a more detailed understanding of customer behavior and for developing more refined strategies.	

CLUSTER INTERPRETATION

A. INTERPRET THE CREATED CLUSTERS AND THE OTHER SEGMENTS (per capita income, yearly income and total debt)

```
df3.groupby('clusters')[['amt_nonfraud_trx_L6M','count_nonfraud_trx_L6M',
'credit_limit', 'DTI']].agg(['count','mean','min','max','median'])
```

	amt_nonfraud_trx_L6M					count_nonfraud_trx_L6M				
	count	mean	min	max	median	count	mean	min	max	median
clusters										
0	458	3,10E+14	336100.0	6,85E+14	290924400.0	458	473.065.502	5	1079	447.5
1	580	3,18E+14	14359600.0	9,34E+14	292151350.0	580	470.163.793	33	1232	453.0
2	157	8,24E+14	329360400.0	2,01E+15	765102400.0	157	1.025.560.510	378	2535	959.0

	credit_limit				DTI				
count	mean	min	max	median	count	mean	min	max	median
458	8,74E+13	638000.0	279669000.0	77557500.0	458	89.612	0	245.416	80.488
580	6,77E+13	831000.0	397061000.0	57440500.0	580	376.973	22.962	891.082	356.791
157	1,52E+14	3844000.0	616927000.0	107987000.0	157	188.400	0	559.401	192.296

```
df3.groupby(['clusters'], as_index = False)['per_capita_income'].mean()
```

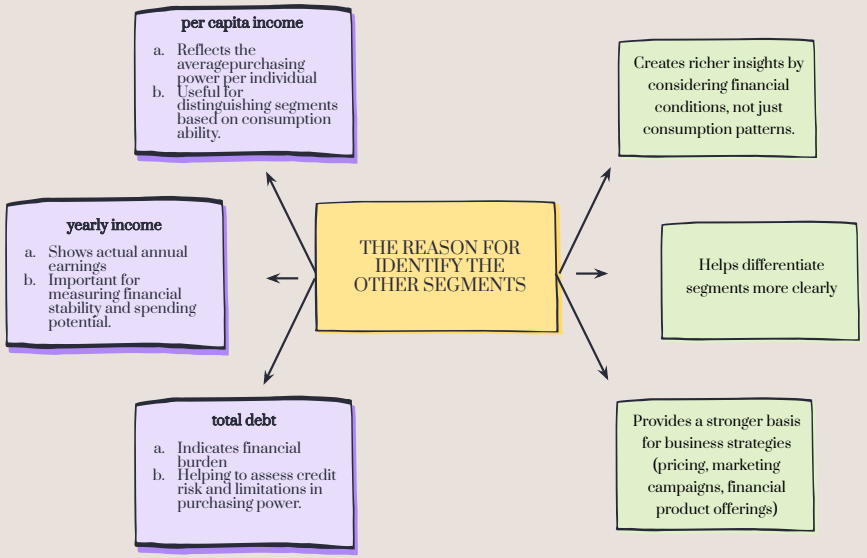
```
df3.groupby(['clusters'], as_index = False)['yearly_income'].mean()
```

```
df3.groupby(['clusters'], as_index = False)['total_debt'].mean()
```

clusters	per_capita_income	
0	0	3,35E+13
1	1	3,41E+13
2	2	5,26E+13

clusters	yearly_income	
0	0	6,25E+13
1	1	6,90E+13
2	2	1,02E+14

clusters	total_debt	
0	0	5,71E+12
1	1	2,60E+13
2	2	1,92E+13



The key statistics for cluster interpretation

Cluster 0
(Financially Healthy Users)

- **Average Amount Non-Fraud Transaction (L6M) :** ~Rp 309.73 million (Lowest Amount)
- **Average Count Non-Fraud Transaction (L6M) :** ~473 transactions
- **Average Credit Limit :** ~Rp 87.40 million
- **Average DTI :** ~0.090 (Lowest DTI)
- **Average Per Capita Income :** ~Rp 33.46 million (Lowest Per Capita Income)
- **Average Yearly Income :** ~Rp 62.48 million
- **Average Total Debt :** ~Rp 5.71 million (Lowest Debt)

Cluster 1
(Credit Dependent Users)

- **Average Amount Non-Fraud Transaction (L6M) :** ~Rp 318.22 million
- **Average Count Non-Fraud Transaction (L6M) :** ~470 transactions (Lowest Count)
- **Average Credit Limit :** ~Rp 67.72 million
- **Average DTI :** ~0.377 (Highest DTI)
- **Average Per Capita Income :** ~Rp 34.10 million
- **Average Yearly Income :** ~Rp 68.95 million
- **Average Total Debt :** ~Rp 26.04 million (Highest Debt)

Cluster 2
(Premium Users)

- **Average Amount Non-Fraud Transaction (L6M) :** ~Rp 824.47 million (Highest Amount)
- **Average Count Non-Fraud Transaction (L6M) :** ~1026 transactions (Highest Count)
- **Average Credit Limit :** ~Rp 151.94 million (Highest Limit)
- **Average DTI:** ~0.188
- **Average Per Capita Income :** ~Rp 52.59 million (Highest Per Capita Income)
- **Average Yearly Income :** ~Rp 102.26 million (Highest Income)
- **Average Total Debt :** ~Rp 19.18 million

Cluster 0 → Lowest Amount, Lowest DTI, Lowest Debt

Cluster 1 → Lowest Count, Highest DTI, Highest Debt

Cluster 2 → Highest Amount, Highest Count, Highest Limit, Highest Per Capita Income, Highest Income

BUSINESS OPPORTUNITIES

CLUSTER THAT GIVE MORE PROFIT FOR REVOBANK

Cluster 0
(Financially Healthy Users)

Clasifications

Steady profit, low risk, active transactions with very low debt make this group a reliable profit source

Cluster 1
(Credit Dependent Users)

Clasifications

Moderate profit, high risk, generates interest income but carries the highest debt burden, making them volatile.

Cluster 2
(Premium Users)

Clasifications

High profit, moderate risk. The most valuable segment : high transaction volume, high credit limits, strong income.

Cluster 0
(Financially Healthy Users)

Cluster 2
(Premium Users)

BUSINESS RECOMMENDATION

Business Recommendation

CLUSTER 0 (Financially Healthy Users)

- **Loyalty Programs :** Offer exclusive rewards for consistent usage (e.g., cashback tiers, travel perks).
- **Relationship Management :** Personalized financial health dashboards to reinforce their strong credit profile.
- **Cross-Selling :** Introduce premium savings accounts or investment starter packs.
- **Loan Terms :** Provide favorable loan rates as a reward for low-risk behavior.

CLUSTER 2 (Premium Users)

- **Loyalty Programs :** Exclusive VIP tiers (airport lounge access, concierge services, luxury partnerships).
- **Relationship Management :** Dedicated relationship managers for personalized financial planning.
- **Cross-Selling :** Wealth management services, higher-tier credit cards, investment products.
- **Loan Terms :** Preferential loan rates and priority approvals to strengthen trust and retention.

Thank You