Customer Segmentation

RevoBank

Venny Amilia Deslaweny
Data Analyst
www.linkedin.com/in/vennydeslaweny
https://github.com/venny-analystic
ve.200501@gmail.com

Project Overview

Project Objective

- Analyze customer transaction and credit behavior.
- Identify fraud patterns and customer segments.
- Support data-driven decision making in credit risk and marketing.

Dataset

- 6 months transaction data
- Customer, transaction, and card attributes

Tools

- Python (Pandas, Matplotlib, Seaborn, Scikit learn)

Business Requirements and Objective

RevoBank a bank based in Indonesia. The Performance Management (PM) team, whose goal is to encourage existing customers to use RevoBank credit card product more frequently (i.e complete more transactions).
A data summary from MIS (management Information System) team, detailing clients and their sales over the last half (6 months).
The dataset contains information about the card dataset and the user dataset.
The objective of this analysis is to evaluate the transaction behavior of RevoBank credit card customers over the past six months by cleaning and preparing accurate data, in order to identify customer segments, financial risks, and opportunities to increase transaction frequency and bank profitability.

Source of Dataset

01 The Card Dataset

- https://www.google.com/url?q=https%3A%2F%2Fdocs.google.com%2Fspreadsheets%2Fd%2F1FVPLOBbUH26R7bnIA3rcOnYebx_8VjRanFvqrPhPoOc%2Fedit%3Fgid%3D1725475419%23gid%3D1725475419

02 The User Dataset

- https://www.google.com/url?q=https%3A%2F%2Fdocs.google.com%2Fspreadsheets%2Fd%2F19r3TjR45sPF8WocQECYRR7GdpE16jgTPliKH1ryg1OQ%2Fedit%3Fgid%3D573531766%23gid%3D573531766

03 Dictionary

- https://www.google.com/url?q=https%3A%2F%2Fdocs.google.com%2Fspreadsheets%2Fd%2F1UciyMTRLse-EiyWLtFLN5psk9HNwegnf%2Fedit%3Fusp%3Dsharing%26ouid%3D106497928254019113442%26rtpof%3Dtrue%26sd%3Dtrue

Disclaimer!
This analysis is based on publicly available and is for educational purposes only. PinjamUang is a fictional company; results are not tied to real business impact.

01

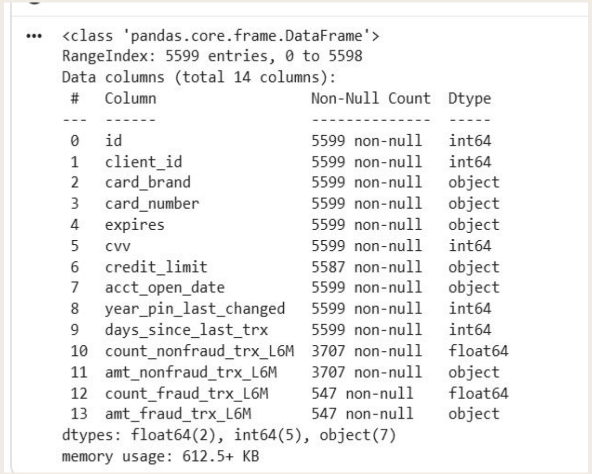
Milestone 1

Data Cleaning

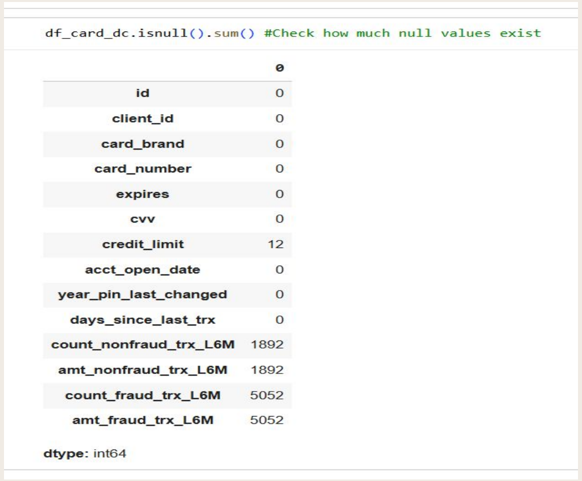
Link Google Colab

Data cleaning (Card dataset)

1. Check data type



```
<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   int64
 1   client_id              5599 non-null   int64
 2   card_brand             5599 non-null   object
 3   card_number            5599 non-null   object
 4   expires                5599 non-null   object
 5   cvv                    5599 non-null   int64
 6   credit_limit           5587 non-null   object
 7   acct_open_date         5599 non-null   object
 8   year_pin_last_changed  5599 non-null   int64
 9   days_since_last_trx    5599 non-null   int64
 10  count_nonfraud_trx_L6M 3707 non-null   float64
 11  amt_nonfraud_trx_L6M   3707 non-null   object
 12  count_fraud_trx_L6M    547 non-null    float64
 13  amt_fraud_trx_L6M      547 non-null    object
dtypes: float64(2), int64(5), object(7)
memory usage: 612.5+ KB
```



```
df_card_dc.isnull().sum() #Check how much null values exist
```

	0
id	0
client_id	0
card_brand	0
card_number	0
expires	0
cvv	0
credit_limit	12
acct_open_date	0
year_pin_last_changed	0
days_since_last_trx	0
count_nonfraud_trx_L6M	1892
amt_nonfraud_trx_L6M	1892
count_fraud_trx_L6M	5052
amt_fraud_trx_L6M	5052

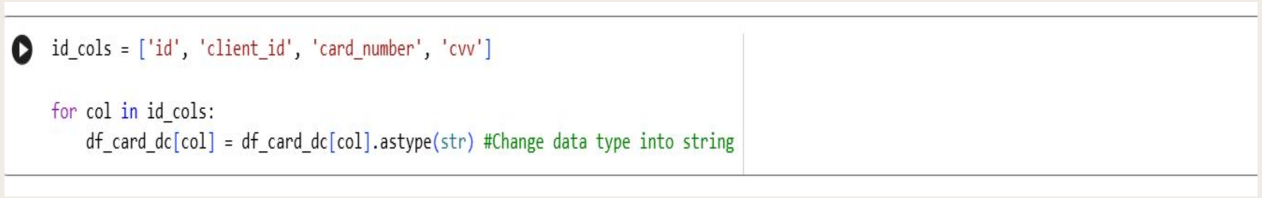
dtype: int64

Insight :

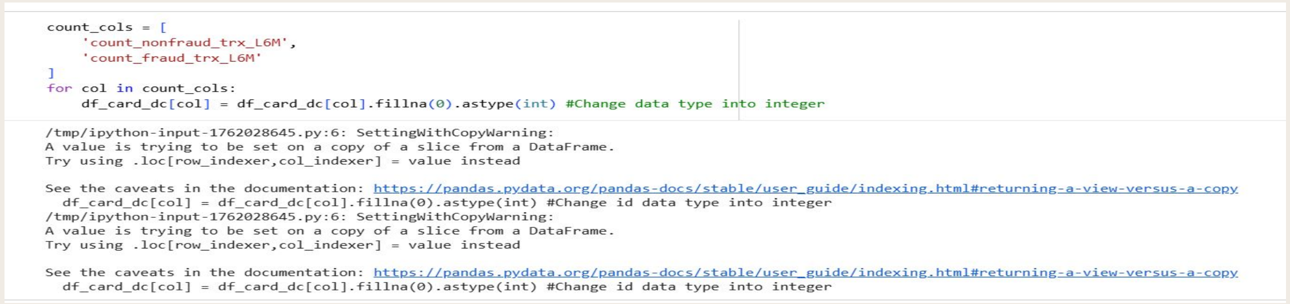
- id, client_id, card_number, ccv can be changed into string (For data type convertion demo purposes)
- count_nonfraud_trx_L6M, count_fraud_trx_L6M can be changed into integer (For data type convertion demo purposes)
- credit_limit, 12 row null
- count_nonfraud_trx_L6M, 1892 row null
- amt_nonfraud_trx_L6M, 1892 row null
- count_fraud_trx_L6M, 5052 row null
- amt_fraud_trx_L6M, 5052 row null

Data cleaning (Card dataset)

2. We will change the data type from int into string and integer



```
id_cols = ['id', 'client_id', 'card_number', 'cvv']

for col in id_cols:
    df_card_dc[col] = df_card_dc[col].astype(str) #Change data type into string
```



```
count_cols = [
    'count_nonfraud_trx_L6M',
    'count_fraud_trx_L6M'
]
for col in count_cols:
    df_card_dc[col] = df_card_dc[col].fillna(0).astype(int) #Change data type into integer
```

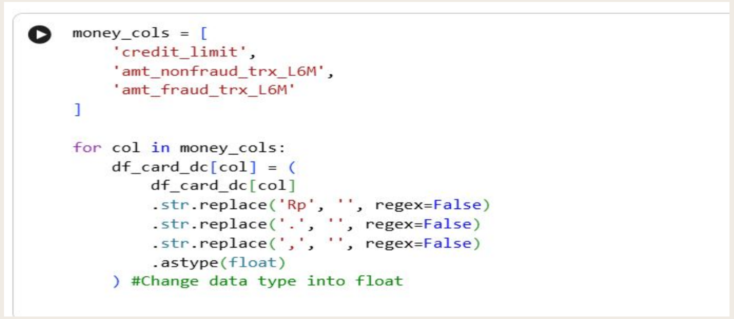
```
/tmp/ipython-input-1762028645.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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_dc[col] = df_card_dc[col].fillna(0).astype(int) #Change id data type into integer
/tmp/ipython-input-1762028645.py:6: 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_dc[col] = df_card_dc[col].fillna(0).astype(int) #Change id data type into integer
```

Insigh :

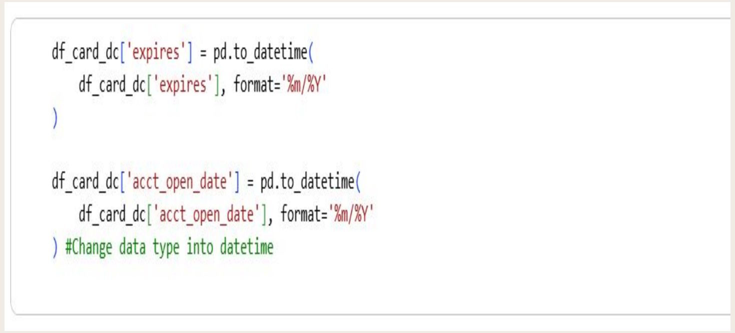
- Identifiers converted to string to prevent misuse in numerical analysis (they have no mathematical meaning).
- Change the data type into integer count must be an integer; NaN means there is no transaction.

Data cleaning (Card dataset)

3. We will change the data type from int into float and datetime



```
money_cols = [
    'credit_limit',
    'amt_nonfraud_trx_L6M',
    'amt_fraud_trx_L6M'
]

for col in money_cols:
    df_card_dc[col] = (
        df_card_dc[col]
        .str.replace('Rp', '', regex=False)
        .str.replace('.', '', regex=False)
        .str.replace(',', '', regex=False)
        .astype(float)
    ) #Change data type into float
```



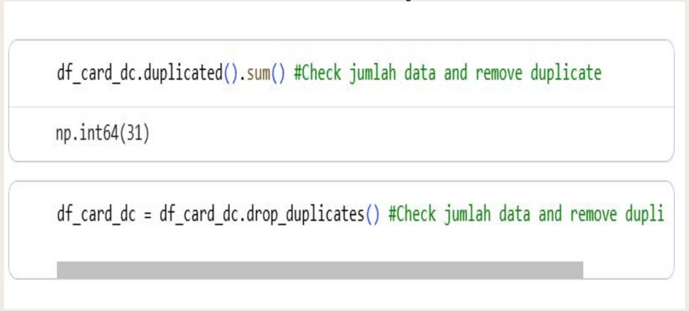
```
df_card_dc['expires'] = pd.to_datetime(
    df_card_dc['expires'], format='%m/%Y'
)

df_card_dc['acct_open_date'] = pd.to_datetime(
    df_card_dc['acct_open_date'], format='%m/%Y'
) #Change data type into datetime
```

Insight :

- Change the data type into float, monetary data must be numeric for risk and profit analysis.
- Change the data type into datetime, time-series analysis & filtering expiry

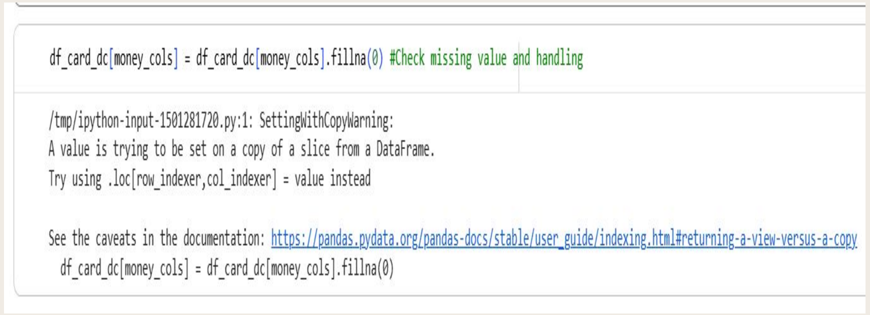
Data cleaning (Card dataset)

4. Check remove duplicate and missing value handling

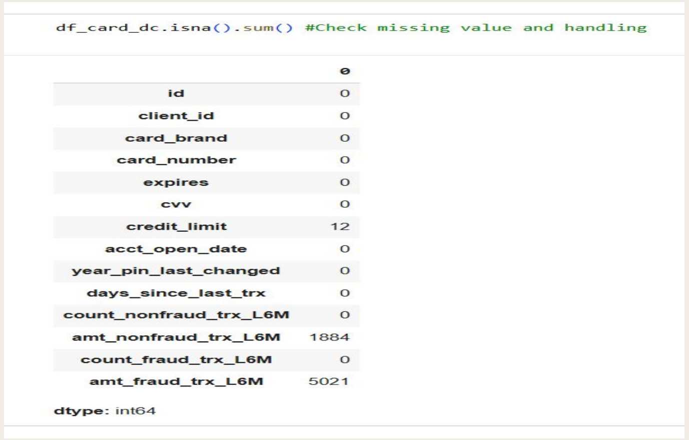


```
df_card_dc.duplicated().sum() #Check jumlah data and remove duplicate

np.int64(31)
```

```
df_card_dc = df_card_dc.drop_duplicates() #Check jumlah data and remove dupli
```



```
df_card_dc[money_cols] = df_card_dc[money_cols].fillna(0) #Check missing value and handling

/tmp/ipython-input-1501281720.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_dc[money_cols] = df_card_dc[money_cols].fillna(0)
```



```
df_card_dc.isna().sum() #Check missing value and handling
```

	0
id	0
client_id	0
card_brand	0
card_number	0
expires	0
cvv	0
credit_limit	12
acct_open_date	0
year_pin_last_changed	0
days_since_last_trx	0
count_nonfraud_trx_L6M	0
amt_nonfraud_trx_L6M	1884
count_fraud_trx_L6M	0
amt_fraud_trx_L6M	5021

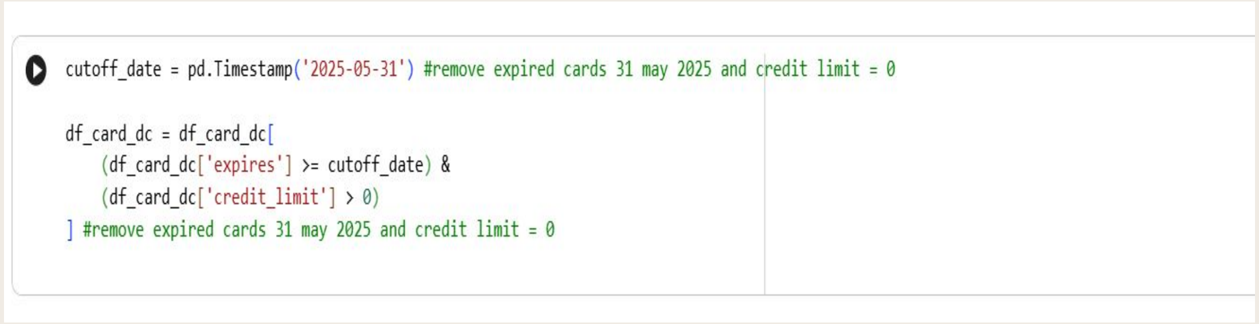
dtype: int64

Insight :

- It is showing that duplicate data are exist, so we need to remove it based on the id since 31 id, because One card = one observation
- Patterns found:
  Missing values in transactions → inactive card
  Already filled:
  count → 0
  amount → 0

Data cleaning (Card dataset)

5. Check remove expired cards and credit limit = 0



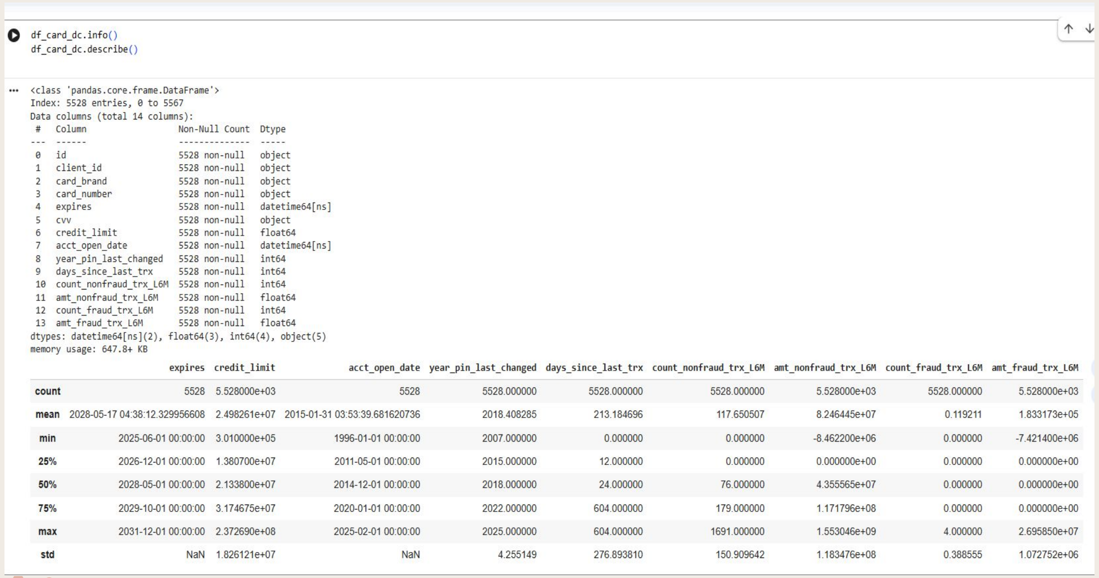
```
cutoff_date = pd.Timestamp('2025-05-31') #remove expired cards 31 may 2025 and credit limit = 0

df_card_dc = df_card_dc[
    (df_card_dc['expires'] >= cutoff_date) &
    (df_card_dc['credit_limit'] > 0)
] #remove expired cards 31 may 2025 and credit limit = 0
```

Insigh :

- The remove expired card and credit limit = 0
  Expired card → not relevant
  Credit limit 0 → inactive & does not generate profit

Data cleaning (Card dataset)

6. Final Validations



```python
df_card_dc.info()
df_card_dc.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5528 entries, 0 to 5567
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    5528 non-null   object
 1   client_id             5528 non-null   object
 2   card_brand            5528 non-null   object
 3   card_number           5528 non-null   object
 4   expires               5528 non-null   datetime64[ns]
 5   cvv                   5528 non-null   object
 6   credit_limit          5528 non-null   float64
 7   acct_open_date        5528 non-null   datetime64[ns]
 8   year_pin_last_changed 5528 non-null   int64
 9   days_since_last_trx   5528 non-null   int64
 10  count_nonfraud_trx_L6M 5528 non-null  int64
 11  amt_nonfraud_trx_L6M  5528 non-null   float64
 12  count_fraud_trx_L6M   5528 non-null   int64
 13  amt_fraud_trx_L6M     5528 non-null   float64
dtypes: datetime64[ns](2), float64(3), int64(4), object(5)
memory usage: 647.8+ KB
```

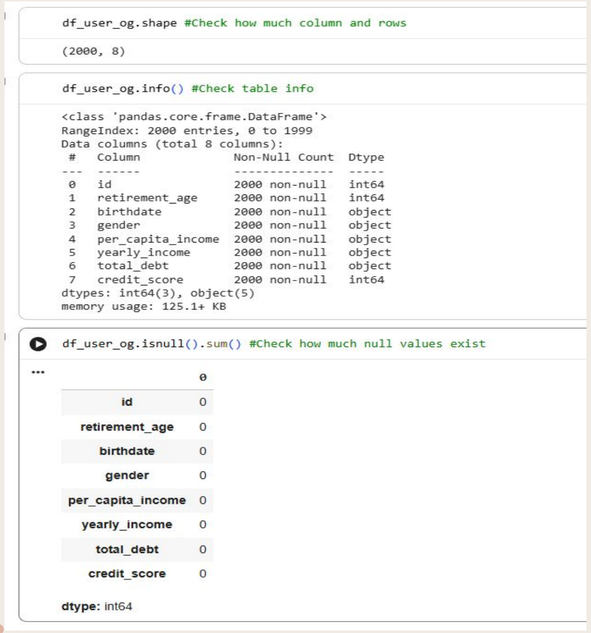
	expires	credit_limit	acct_open_date	year_pin_last_changed	days_since_last_trx	count_nonfraud_trx_L6M	amt_nonfraud_trx_L6M	count_fraud_trx_L6M	amt_fraud_trx_L6M
count	5528	5.528000e+03	5528	5528.000000	5528.000000	5528.000000	5.528000e+03	5528.000000	5.528000e+03
mean	2028-05-17 04:38:12.329956608	2.498261e+07	2015-01-31 03:53:39.681620736	2018.408285	213.184696	117.650507	8.246445e+07	0.119211	1.833173e+05
min	2025-06-01 00:00:00	3.010000e+05	1996-01-01 00:00:00	2007.000000	0.000000	0.000000	-8.462200e+06	0.000000	-7.421400e+06
25%	2026-12-01 00:00:00	1.380700e+07	2011-05-01 00:00:00	2015.000000	12.000000	0.000000	0.000000e+00	0.000000	0.000000e+00
50%	2028-05-01 00:00:00	2.133800e+07	2014-12-01 00:00:00	2018.000000	24.000000	76.000000	4.355565e+07	0.000000	0.000000e+00
75%	2029-10-01 00:00:00	3.174675e+07	2020-01-01 00:00:00	2022.000000	604.000000	179.000000	1.171796e+08	0.000000	0.000000e+00
max	2031-12-01 00:00:00	2.372690e+08	2025-02-01 00:00:00	2025.000000	604.000000	1691.000000	1.553046e+09	4.000000	2.695850e+07
std	NaN	1.826121e+07	NaN	4.255149	276.893810	150.909642	1.183476e+08	0.388555	1.072752e+06

Insight :

- Data Quality: The dataset is very clean (no missing values).
- Fraud Imbalance: The fraud data (both count and amount) is highly imbalanced, with over 75% of users having no fraud. This must be considered when creating predictive models.
- High Variance: The credit limit column (credit_limit) shows extreme variance (outliers), which needs attention during data pre-processing.
- Anomaly: The presence of negative fraud amounts (amt_fraud_trx_L6M) requires investigation to understand whether it represents a chargeback or a data entry error.

Data cleaning (User dataset)

1. Check data type



```
df_user_og.shape #Check how much column and rows
```

```
(2000, 8)
```

```
df_user_og.info() #Check table 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   int64
 1   retirement_age     2000 non-null   int64
 2   birthdate          2000 non-null   object
 3   gender             2000 non-null   object
 4   per_capita_income  2000 non-null   object
 5   yearly_income      2000 non-null   object
 6   total_debt         2000 non-null   object
 7   credit_score       2000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 125.1+ KB
```

```
df_user_og.isnull().sum() #Check how much null values exist
```

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

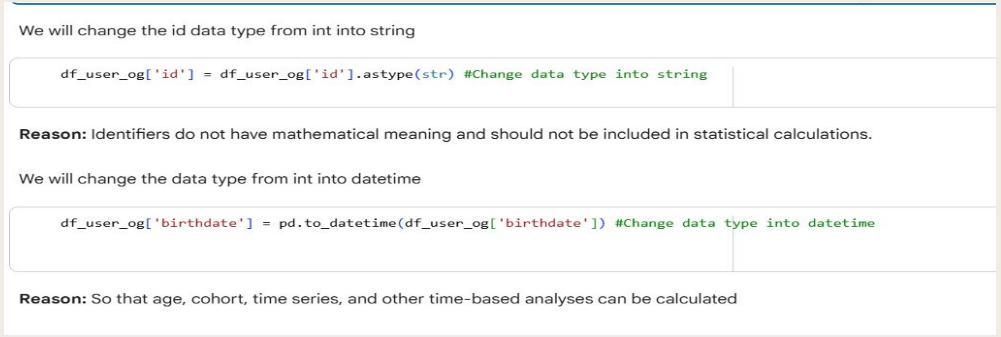
dtype: int64

Insight :

- id can be changed into string
- birthdate can be changed into datetime
- retired_age, credit_data can be changed into count/integer
- per_capita_income, yearly_income, total_debt can be changed into float
- No missing values were found across all columns
- The dataset is complete in terms of quantity

Data cleaning (User dataset)

2. We will change the data type from int into string and datetime



We will change the id data type from int into string

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

Reason: Identifiers do not have mathematical meaning and should not be included in statistical calculations.

We will change the data type from int into datetime

```
df_user_og['birthdate'] = pd.to_datetime(df_user_og['birthdate']) #Change data type into datetime
```

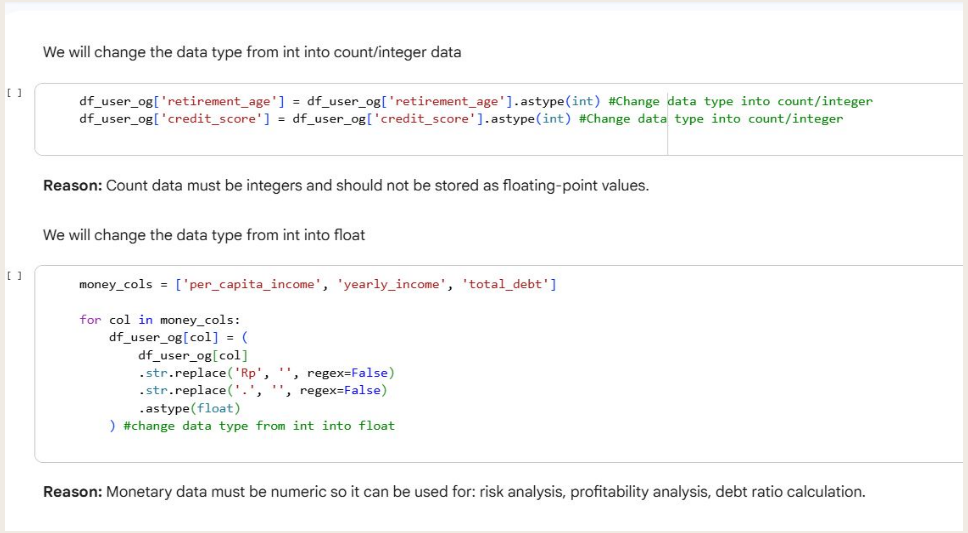
Reason: So that age, cohort, time series, and other time-based analyses can be calculated

Insight :

- Standardizing the id and birthdate data types improves analytical accuracy, prevents statistical misuse, and enables robust time-based feature engineering for advanced customer and risk analysis.

Data cleaning (User dataset)

3. We will change the data type from int into count/integer and float



We will change the data type from int into count/integer data

```
df_user_og['retirement_age'] = df_user_og['retirement_age'].astype(int) #Change data type into count/integer
df_user_og['credit_score'] = df_user_og['credit_score'].astype(int) #Change data type into count/integer
```

Reason: Count data must be integers and should not be stored as floating-point values.

We will change the data type from int into float

```
money_cols = ['per_capita_income', 'yearly_income', 'total_debt']

for col in money_cols:
    df_user_og[col] = (
        df_user_og[col]
        .str.replace('Rp', '', regex=False)
        .str.replace('.', '', regex=False)
        .astype(float)
    ) #change data type from int into float
```

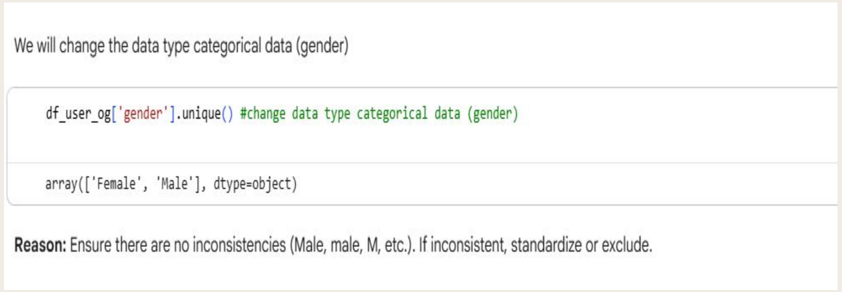
Reason: Monetary data must be numeric so it can be used for: risk analysis, profitability analysis, debt ratio calculation.

Insight :

- Standardizing count variables as integers and monetary variables as floats ensures numerical accuracy, prevents analytical distortion, and enables robust financial and risk-based analyses.

Data cleaning (User dataset)

4. We will change the data type categorical data gender



We will change the data type categorical data (gender)

```
df_user_og['gender'].unique() #change data type categorical data (gender)
```

```
array(['Female', 'Male'], dtype=object)
```

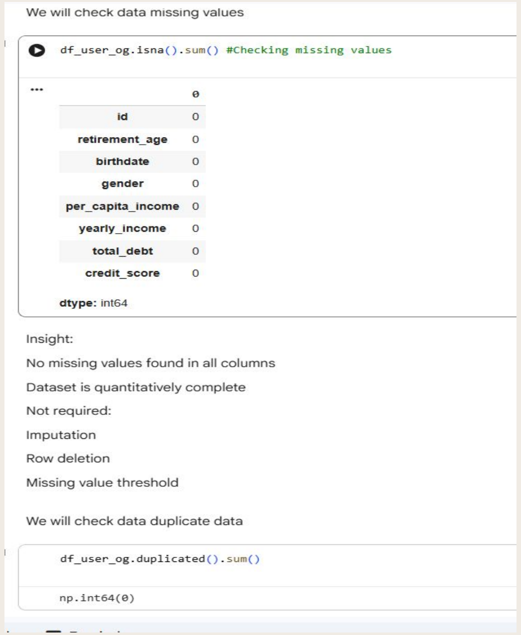
Reason: Ensure there are no inconsistencies (Male, male, M, etc.). If inconsistent, standardize or exclude.

Insight :

The gender variable contains only valid and consistent categories, ensuring high data integrity and readiness for categorical encoding and demographic analysis.

Data cleaning (User dataset)

5. Final Validations



We will check data missing values

```
df_user_og.isna().sum() #Checking missing values
```

```
                  0
              id  0
  retirement_age  0
       birthdate  0
          gender  0
per_capita_income 0
    yearly_income 0
      total_debt  0
    credit_score  0

dtype: int64
```

Insight:

No missing values found in all columns

Dataset is quantitatively complete

Not required:

Imputation

Row deletion

Missing value threshold

We will check data duplicate data

```
df_user_og.duplicated().sum()
```

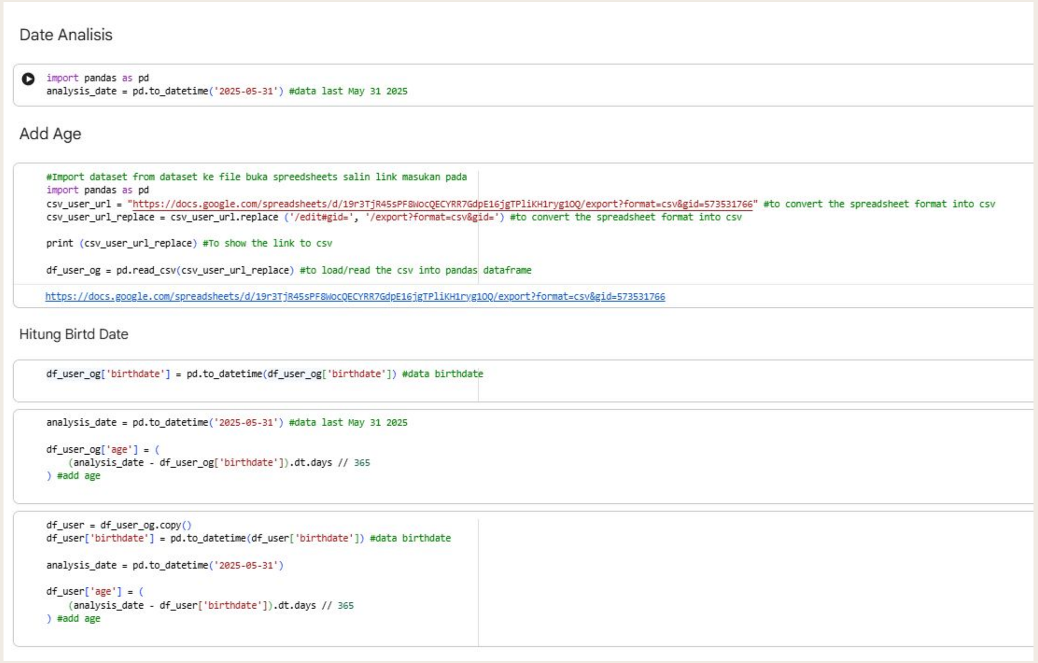
```
np.int64(0)
```

Insight :

No missing values found in all columns. Dataset is quantitatively complete. Not required: Imputation, Row deletion, Missing value threshold

Data cleaning (User dataset)

6. Add column age



Date Analisis

```
import pandas as pd
analysis_date = pd.to_datetime('2025-05-31') #data last May 31 2025
```

Add Age

```
#Import dataset from dataset ke file buka spreedsheets salin link masukan pada
import pandas as pd
csv_user_url = "https://docs.google.com/spreadsheets/d/19r3TjR45sPF8WocQECYRR7GdpE16jgTPliKH1ryg10Q/export?format=csv&gid=573531766" #to convert the spreadsheet format into csv
csv_user_url_replace = csv_user_url.replace ('/edit#gid=', '/export?format=csv&gid=') #to convert the spreadsheet format into csv

print (csv_user_url_replace) #To show the link to csv

df_user_og = pd.read_csv(csv_user_url_replace) #to load/read the csv into pandas dataframe
```

```
https://docs.google.com/spreadsheets/d/19r3TjR45sPF8WocQECYRR7GdpE16jgTPliKH1ryg10Q/export?format=csv&gid=573531766
```

Hitung Birtd Date

```
df_user_og['birthdate'] = pd.to_datetime(df_user_og['birthdate']) #data birthdate
```

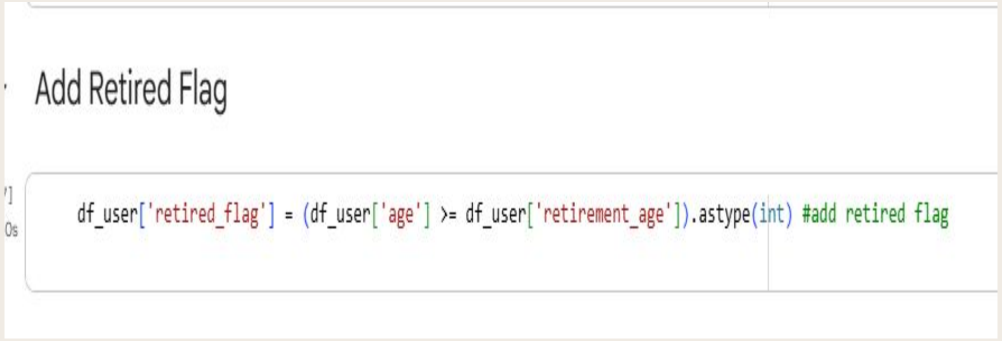
```
analysis_date = pd.to_datetime('2025-05-31') #data last May 31 2025

df_user_og['age'] = (
    (analysis_date - df_user_og['birthdate']).dt.days // 365
) #add age
```

```
df_user = df_user_og.copy()
df_user['birthdate'] = pd.to_datetime(df_user['birthdate']) #data birthdate

analysis_date = pd.to_datetime('2025-05-31')

df_user['age'] = (
    (analysis_date - df_user['birthdate']).dt.days // 365
) #add age
```

Import the pandas library.
Load the user dataset from Google Sheets into a pandas DataFrame.
Convert the birthdate column from string to datetime format.
Define the analysis reference date as May 31, 2025.
Create a copy of the original dataset to preserve the raw data.
Calculate the age by subtracting the birthdate from the analysis date and converting the result into years.
Age was calculated as the difference between the analysis date (May 31, 2025) and the user's birthdate, expressed in years.

Data cleaning (User dataset)

7. Add column retired flag



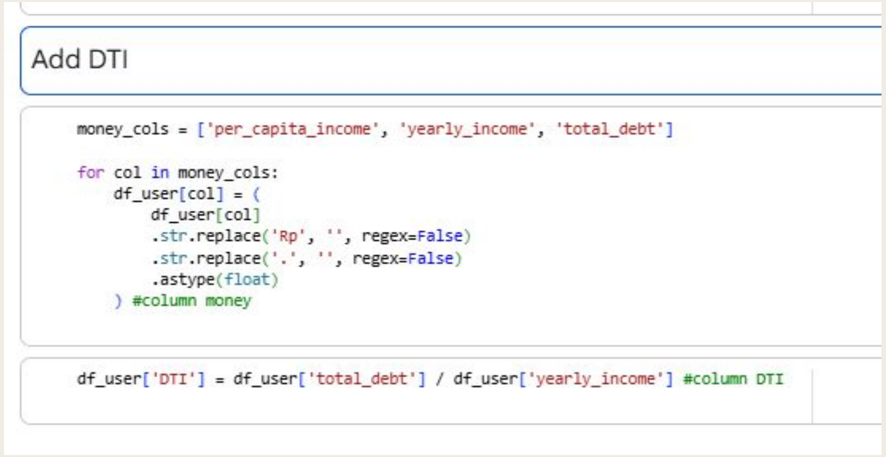
Add Retired Flag

```
df_user['retired_flag'] = (df_user['age'] >= df_user['retirement_age']).astype(int) #add retired flag
```

Ensure that the age column has been calculated based on the analysis date (May 31, 2025).
Ensure that retirement_age is stored as an integer.
Create a new column called retired_flag by comparing the user's age with the retirement age.
Assign value 1 if the user has reached or exceeded retirement age, otherwise assign 0.
A retirement flag was created to indicate whether a user has reached retirement age. Users whose age is greater than or equal to their retirement age are labeled as retired (1), while others are labeled as not retired (0).

Data cleaning (User dataset)

8. Add column DTI



Add DTI

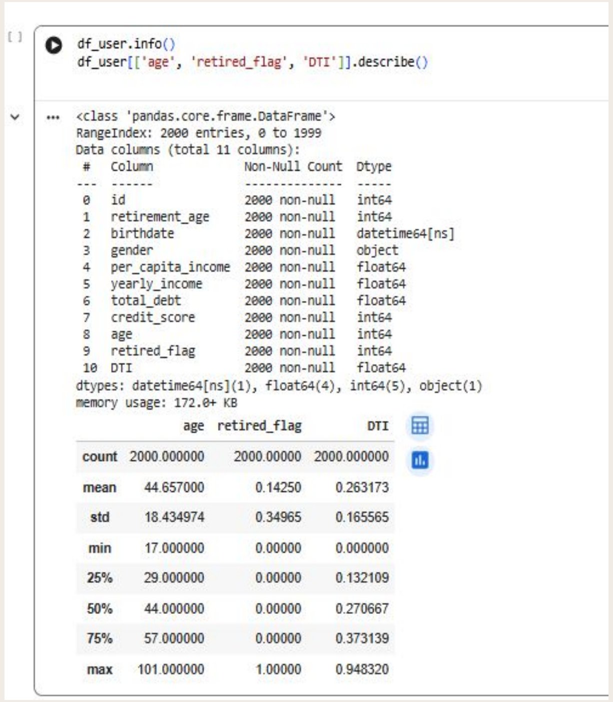
```
money_cols = ['per_capita_income', 'yearly_income', 'total_debt']

for col in money_cols:
    df_user[col] = (
        df_user[col]
        .str.replace('Rp', '', regex=False)
        .str.replace('.', '', regex=False)
        .astype(float)
    ) #column money
```

```
df_user['DTI'] = df_user['total_debt'] / df_user['yearly_income'] #column DTI
```

Identify financial columns that contain monetary values.
Remove currency symbols and thousand separators, then convert the values into float type.
Calculate the Debt-to-Income (DTI) ratio by dividing total debt by yearly income.
The Debt-to-Income (DTI) ratio was calculated by dividing total debt by yearly income. This metric reflects a user’s financial burden and is a key indicator for credit risk assessment.

Data cleaning (User dataset)

9. Validation



```
df_user.info()
df_user[['age', 'retired_flag', 'DTI']].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 2000 non-null   int64
 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
 8   age                2000 non-null   int64
 9   retired_flag       2000 non-null   int64
 10  DTI                2000 non-null   float64
dtypes: datetime64[ns](1), float64(4), int64(5), object(1)
memory usage: 172.0+ KB
```

	age	retired_flag	DTI
count	2000.000000	2000.00000	2000.000000
mean	44.657000	0.14250	0.263173
std	18.434974	0.34965	0.165565
min	17.000000	0.00000	0.000000
25%	29.000000	0.00000	0.132109
50%	44.000000	0.00000	0.270667
75%	57.000000	0.00000	0.373139
max	101.000000	1.00000	0.948320

The Debt-to-Income (DTI) ratio was calculated after converting all monetary variables into numeric format. Validation confirms that the dataset contains no missing values and that all derived features are suitable for further credit risk and profitability analysis.

Key insights:

- The average age is approximately 45 years
- About 14% of users are retired
- The average DTI is around 0.26, indicating a moderate debt burden
- No missing or invalid values are present in the derived variables

02

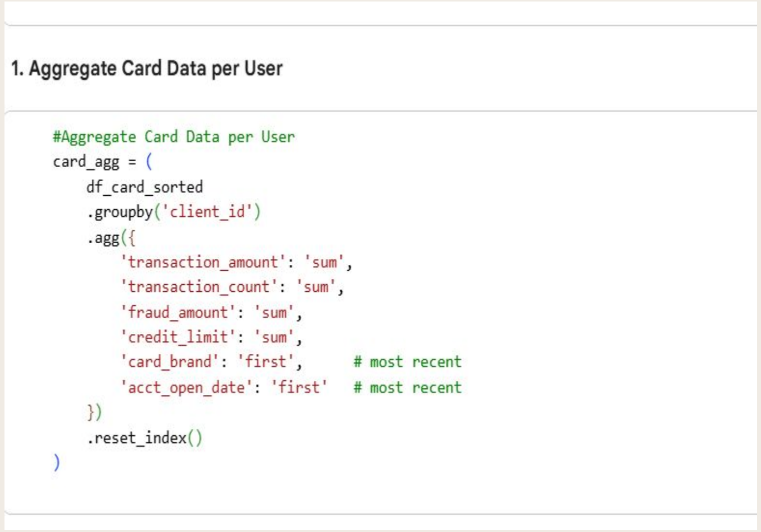
Milestone 2

EDA (Exploration Data Analysis)

Link Google Colab

EDA (Exploratory Data Analysis)

1. Aggregate Card Data a User.



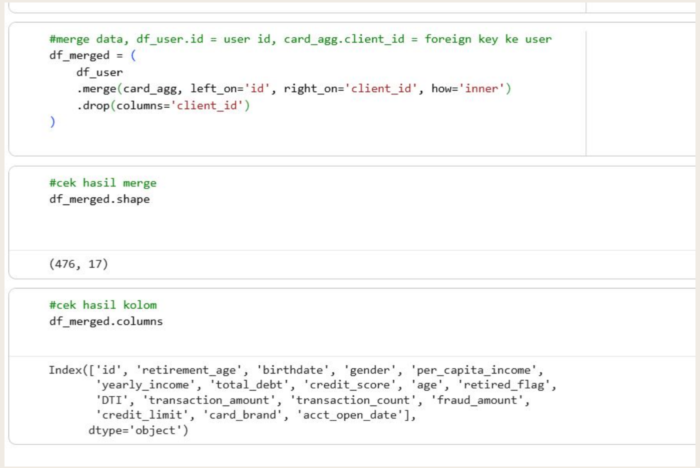
1. Aggregate Card Data per User

```
#Aggregate Card Data per User
card_agg = (
    df_card_sorted
    .groupby('client_id')
    .agg({
        'transaction_amount': 'sum',
        'transaction_count': 'sum',
        'fraud_amount': 'sum',
        'credit_limit': 'sum',
        'card_brand': 'first',      # most recent
        'acct_open_date': 'first'   # most recent
    })
    .reset_index()
)
```

Insight :
The credit card profit and risk analysis structure is well established. However, no transaction activity of economic value was detected during this observation period, so profitability cannot yet be evaluated. The next focus is to ensure the availability and completeness of transaction data before drawing business conclusions.

EDA (Exploratory Data Analysis)

2. Merge with the User Data.



```
#merge data, df_user.id = user id, card_agg.client_id = foreign key ke user
df_merged = (
    df_user
    .merge(card_agg, left_on='id', right_on='client_id', how='inner')
    .drop(columns='client_id')
)
```

```
#cek hasil merge
df_merged.shape
```

```
(476, 17)
```

```
#cek hasil kolom
df_merged.columns
```

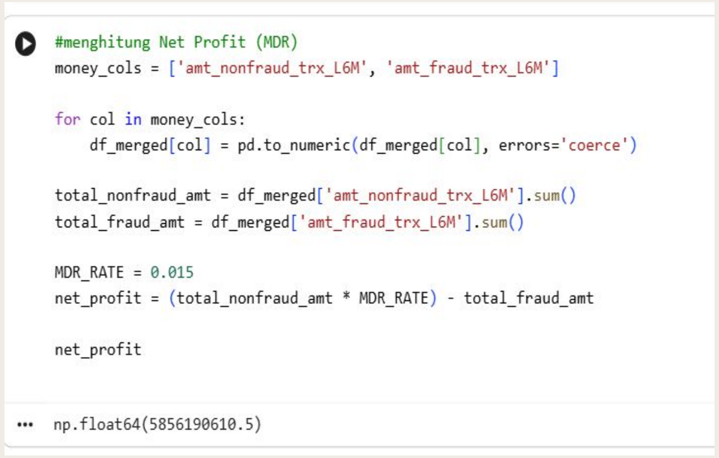
```
Index(['id', 'retirement_age', 'birthdate', 'gender', 'per_capita_income',
       'yearly_income', 'total_debt', 'credit_score', 'age', 'retired_flag',
       'DTI', 'transaction_amount', 'transaction_count', 'fraud_amount',
       'credit_limit', 'card_brand', 'acct_open_date'],
      dtype='object')
```

Insigh :

The analysis shows that despite having 476 cardholders with good financial capacity, the portfolio has not generated revenue due to a lack of recorded transaction activity. The business focus should be on activation and increasing usage, rather than tightening risk.

EDA (Exploratory Data Analysis)

A. Calculate the total **Net Profit** from all transactions in the last 6 months.



```
#menghitung Net Profit (MDR)
money_cols = ['amt_nonfraud_trx_L6M', 'amt_fraud_trx_L6M']

for col in money_cols:
    df_merged[col] = pd.to_numeric(df_merged[col], errors='coerce')

total_nonfraud_amt = df_merged['amt_nonfraud_trx_L6M'].sum()
total_fraud_amt = df_merged['amt_fraud_trx_L6M'].sum()

MDR_RATE = 0.015
net_profit = (total_nonfraud_amt * MDR_RATE) - total_fraud_amt

net_profit
```

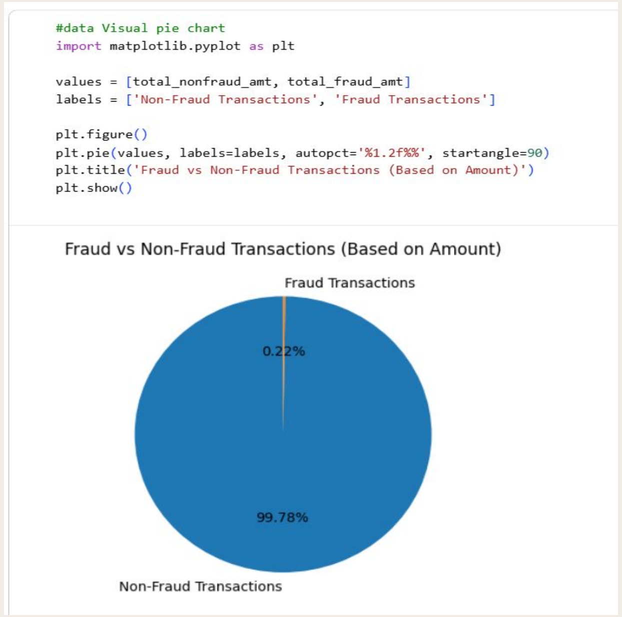
```
np.float64(5856190610.5)
```

Insight :

The calculated net profit indicates that the bank's revenue from transaction fees is sufficient to absorb fraud-related losses. This suggests that fraud risk, while present, does not materially threaten overall transaction profitability.

EDA (Exploratory Data Analysis)

B. Calculate the **Fraud Rate** of RevoBank visualise use a pie chart.

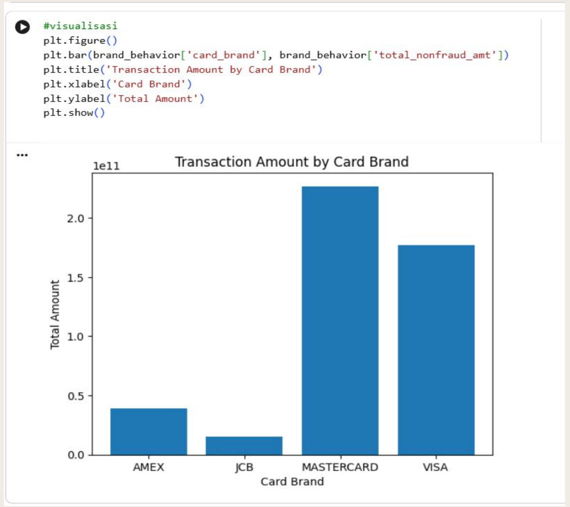


```
#data Visual pie chart
import matplotlib.pyplot as plt

values = [total_nonfraud_amt, total_fraud_amt]
labels = ['Non-Fraud Transactions', 'Fraud Transactions']

plt.figure()
plt.pie(values, labels=labels, autopct='%1.2f%%', startangle=90)
plt.title('Fraud vs Non-Fraud Transactions (Based on Amount)')
plt.show()
```

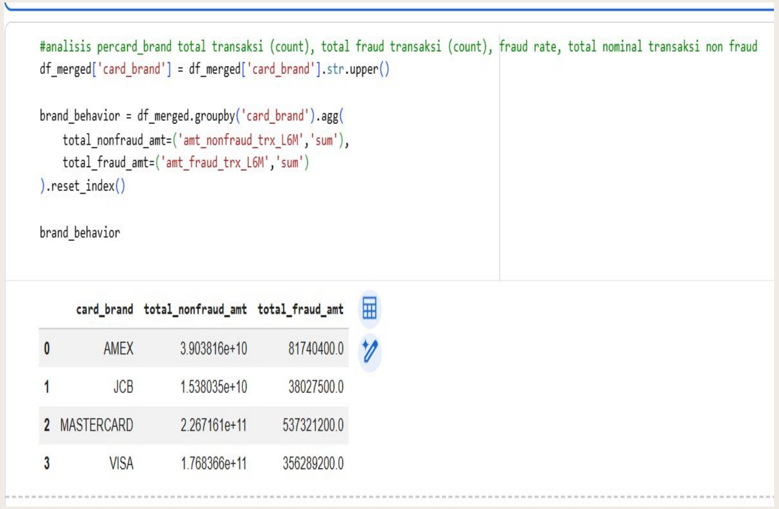
Insight :
The proportion of fraud transaction value is very small compared to total transaction value, suggesting that fraud exposure remains limited and does not significantly impact overall transaction performance.

EDA（Exploratory Data Analysis）

C. **Difference in transaction behaviour** per card brand.



```
#visualisasi
plt.figure()
plt.bar(brand_behavior['card_brand'], brand_behavior['total_nonfraud_amt'])
plt.title('Transaction Amount by Card Brand')
plt.xlabel('Card Brand')
plt.ylabel('Total Amount')
plt.show()
```



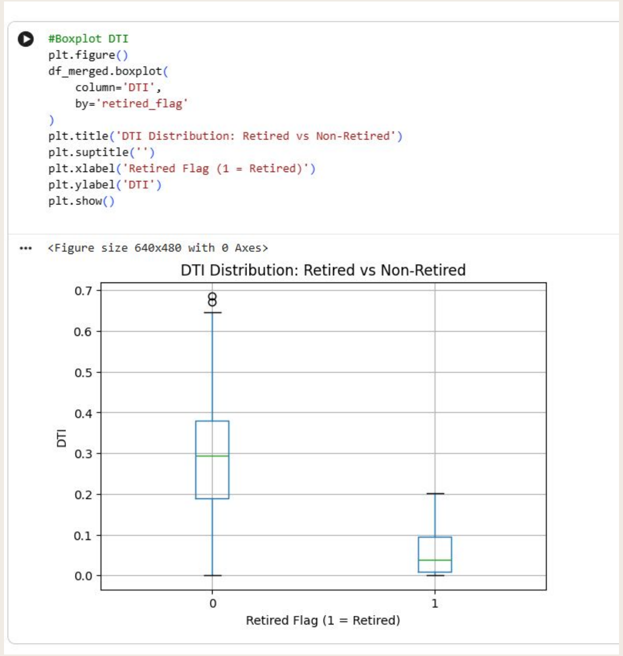
```
#analisis percard_brand total transaksi (count), total fraud transaksi (count), fraud rate, total nominal transaksi non fraud
df_merged['card_brand'] = df_merged['card_brand'].str.upper()

brand_behavior = df_merged.groupby('card_brand').agg(
    total_nonfraud_amt=('amt_nonfraud_trx_L6M','sum'),
    total_fraud_amt=('amt_fraud_trx_L6M','sum')
).reset_index()

brand_behavior
```

```
   card_brand  total_nonfraud_amt  total_fraud_amt
0        AMEX        3.903816e+10       81740400.0
1         JCB        1.538035e+10       38027500.0
2  MASTERCARD        2.267161e+11      537321200.0
3        VISA        1.768366e+11      356289200.0
```

Insight :
The distribution of transaction amounts indicates that Mastercard and Visa dominate transaction activity, suggesting higher customer adoption or usage frequency. In contrast, Amex and JCB represent a smaller share of total transaction value, indicating lower transaction penetration in the observed period. Mastercard and Visa account for the majority of transaction volume, while Amex and JCB contribute relatively smaller portions.

EDA (Exploratory Data Analysis)

D. **DTI** of Retired VS Non Retired Users.



```
#Boxplot DTI
plt.figure()
df_merged.boxplot(
    column='DTI',
    by='retired_flag'
)
plt.title('DTI Distribution: Retired vs Non-Retired')
plt.suptitle('')
plt.xlabel('Retired Flag (1 = Retired)')
plt.ylabel('DTI')
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

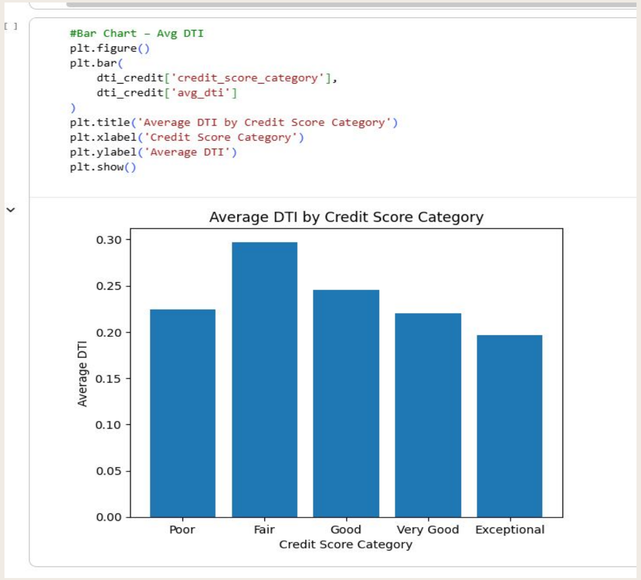
DTI Distribution: Retired vs Non-Retired
DTI
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
Retired Flag (1 = Retired)

Insight :

The DTI distribution shows significant differences in risk between retired and non-retired customers. Non-retired customers have higher leverage and are more volatile, while retired customers exhibit a low and stable DTI profile. Segmentation based on retirement status is highly relevant for credit policies and limit setting.

EDA（Exploratory Data Analysis）

E. **DTI** based on Credit Score Category.

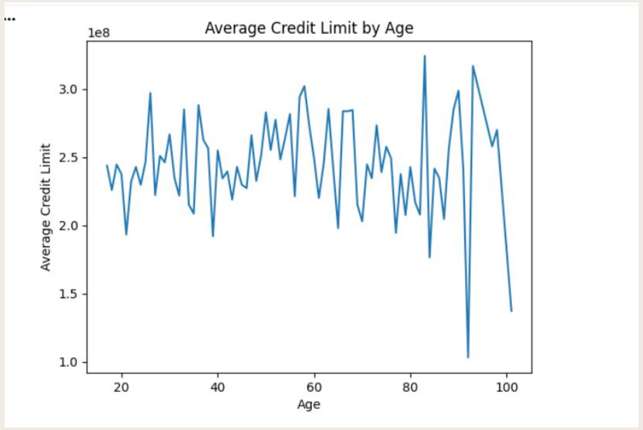


```
#Bar Chart - Avg DTI
plt.figure()
plt.bar(
    dti_credit['credit_score_category'],
    dti_credit['avg_dti']
)
plt.title('Average DTI by Credit Score Category')
plt.xlabel('Credit Score Category')
plt.ylabel('Average DTI')
plt.show()
```

Insight :
The average DTI shows a downward trend as credit score categories increase. Interestingly, the 'Fair' segment has the highest leverage, indicating the need for stricter oversight of this segment rather than solely focusing on customers with the lowest scores.

EDA（Exploratory Data Analysis）

F. Relationship between user age and credit limit.



Insight :

The distribution of average credit limits shows no clear upward or downward trend across age groups, suggesting that credit limit decisions are likely influenced by factors other than age, such as income, credit score, or financial history. Larger fluctuations at older ages may reflect lower customer counts in those groups.

Key Findings from EDA

- Fraud cases are rare but present clear behavioral patterns.
- Credit limit distribution is highly skewed with significant outliers.
- Retired customers show different DTI patterns compared to non-retired.
- Transaction behavior varies across card categories.

03

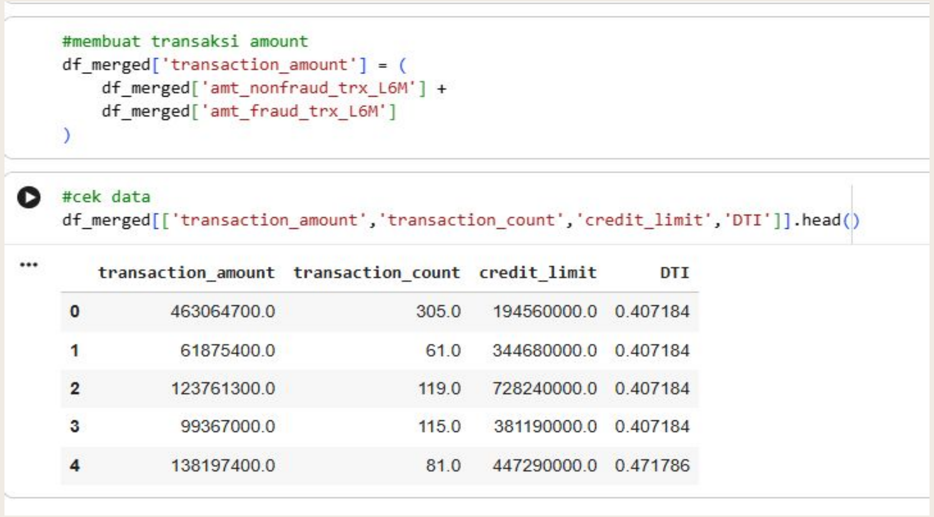
Milestone 3

Segmentation / Clustering

Link Google Colab

Segmentation/Clustering

a. What using apply method to form segmen for this case



```
#membuat transaksi amount
df_merged['transaction_amount'] = (
    df_merged['amt_nonfraud_trx_L6M'] +
    df_merged['amt_fraud_trx_L6M']
)
```

```
#cek data
df_merged[['transaction_amount','transaction_count','credit_limit','DTI']].head()
```

	transaction_amount	transaction_count	credit_limit	DTI
0	463064700.0	305.0	194560000.0	0.407184
1	61875400.0	61.0	344680000.0	0.407184
2	123761300.0	119.0	728240000.0	0.407184
3	99367000.0	115.0	381190000.0	0.407184
4	138197400.0	81.0	447290000.0	0.471786

Insight :
Method: K-Means Clustering

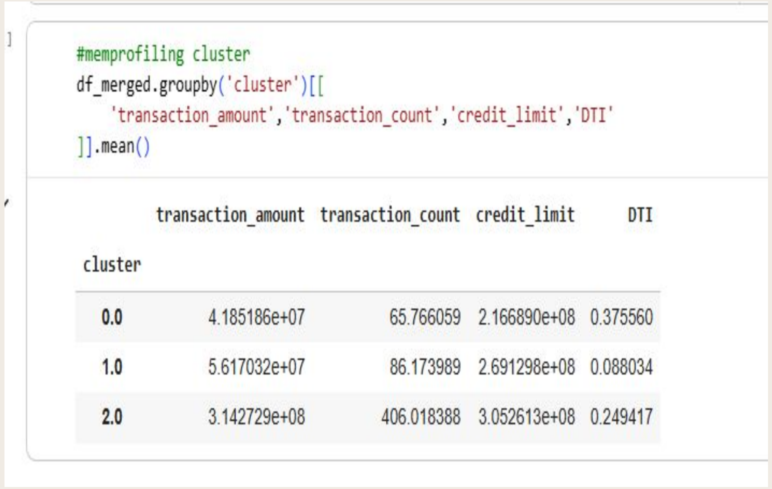
Reason for choosing this method:
1. Data is dominated by numerical features
2. Suitable for grouping customers based on behavioral similarities
3. Easy to interpret for banking business needs

Features used:
* transaction_amount
* transaction_count
* credit_limit
* DTI

These variables represent customer value, activity level, credit exposure, and financial risk.

Segmentation/Clustering

b. Feature selection



```
#memprofiling cluster
df_merged.groupby('cluster')[[
    'transaction_amount','transaction_count','credit_limit','DTI'
]].mean()
```

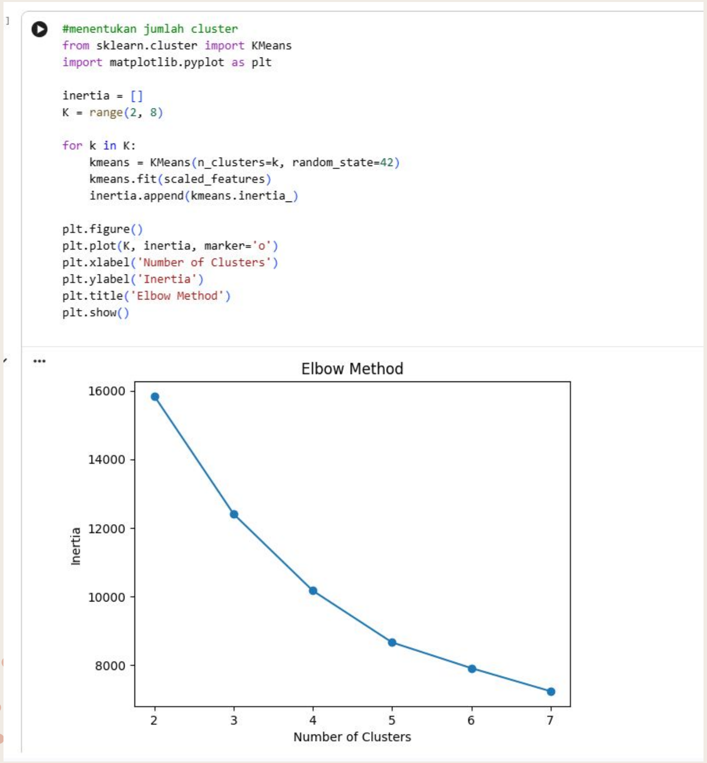
```
        transaction_amount  transaction_count  credit_limit       DTI
cluster
    0.0       4.185186e+07          65.766059  2.166890e+08  0.375560
    1.0       5.617032e+07          86.173989  2.691298e+08  0.088034
    2.0       3.142729e+08         406.018388  3.052613e+08  0.249417
```

Insight :

Transaction amount and transaction count were derived by combining fraud and non-fraud transaction metrics over the last six months. These derived variables were then used together with credit limit and DTI for K-Means clustering.

Segmentation/Clustering

c. Elbow Method



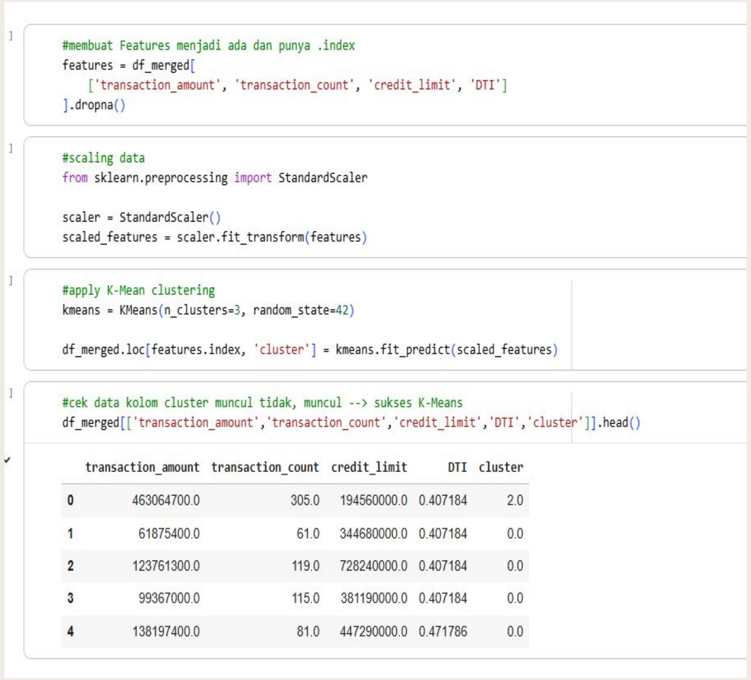
```
#menentukan jumlah cluster
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertia = []
K = range(2, 8)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)

plt.figure()
plt.plot(K, inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```

Insight :

The elbow point is observed at k = 3, where further increases in the number of clusters result in marginal improvements only.

Segmentation/Clustering

d. Apply K-Means



```
#membuat Features menjadi ada dan punya .index
features = df_merged[
    ['transaction_amount', 'transaction_count', 'credit_limit', 'DTI']
].dropna()
```

```
#scaling data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

```
#apply K-Mean clustering
kmeans = KMeans(n_clusters=3, random_state=42)

df_merged.loc[features.index, 'cluster'] = kmeans.fit_predict(scaled_features)
```

```
#cek data kolom cluster muncul tidak, muncul --> sukses K-Means
df_merged[['transaction_amount','transaction_count','credit_limit','DTI','cluster']].head()
```

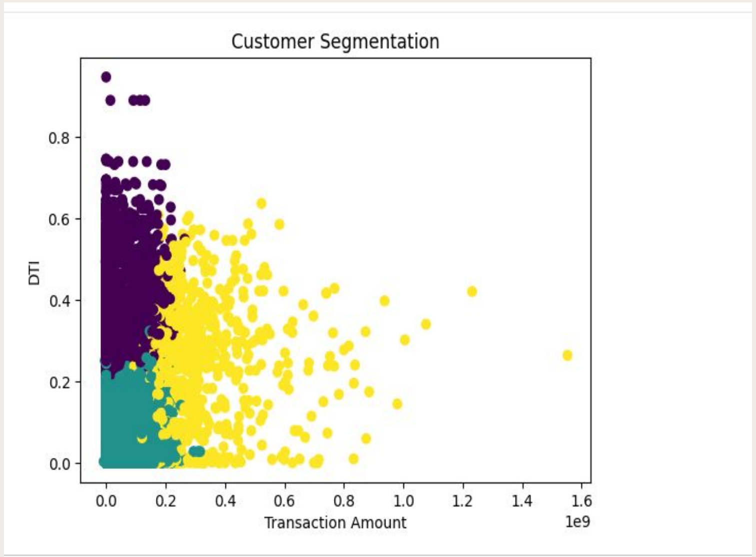
```
   transaction_amount  transaction_count  credit_limit       DTI  cluster
0         463064700.0              305.0   194560000.0  0.407184      2.0
1          61875400.0               61.0   344680000.0  0.407184      0.0
2         123761300.0              119.0   728240000.0  0.407184      0.0
3          99367000.0              115.0   381190000.0  0.407184      0.0
4         138197400.0               81.0   447290000.0  0.471786      0.0
```

Insight :

Transaction behaviour, credit exposure, and financial risk indicators were combined to form a comprehensive feature set for customer segmentation. This ensures that clustering results reflect both customer value and risk profile, enabling more targeted business strategies.

Segmentation/Clustering

e. Cluster Interpretation and Recommendation for Revobank's



Insight :
Customer segmentation reveals three distinct groups. Cluster 2 generates the highest transaction value and activity with manageable risk, making it the primary profit driver. Cluster 1 shows strong growth potential due to low risk and moderate activity, while Cluster 0 presents higher risk with limited contribution to revenue.

Business Interpretation

- High-value cluster represents primary revenue driver.
- Medium cluster shows potential for growth through targeted promotion.
- Low activity cluster may require retention strategy.

Recommendation for Revobank's :
RevoBank should prioritise Cluster 2 through loyalty programs, exclusive rewards, and personalised promotions to maximise transaction volume. Cluster 1 should be targeted with spending incentives and selective credit limit increases to stimulate growth. Cluster 0 requires conservative credit management and risk monitoring to prevent potential losses.

Business Impact

- Customer segmentation enables targeted marketing strategies.
- Fraud pattern identification supports risk mitigation.
- High-value customer group identified for premium targeting.
- Insights can improve transaction frequency and profitability.

Key Takeaway :
Data driven segmentation can improve profitability while managing credit risk.

Thanks!