Credit EDA & Credit Score Calculation with Python

Google Collab Link:

https://colab.research.google.com/drive/1HGGjRCDOHz6JCi55FgAuGrPFFzyzobul?authuser=0#scrollTo=Ng5SOW4noaGj

Problem statement:

To conduct a thorough exploratory data analysis (EDA) and deep analysis of a comprehensive dataset containing basic customer details and extensive credit-related information. The aim is to create new, informative features, calculate a hypothetical credit score, and uncover meaningful patterns, anomalies, and insights within the data.

Dataset:

https://drive.google.com/file/d/1pljm6_3nxcFS9UMIFm124HBsjNZP6ACA/view?usp=sharing

Data Dictionary:

The data dictionary is available here: https://docs.google.com/spreadsheets/d/1ZuK6o1MXFLmnhkFuDEedasDfVqu9ISPV/edit#gid=688359417

Expectations:

The project expects a deep dive into bank details and credit data, creating valuable features, a hypothetical credit score, and uncovering hidden patterns. This involves thorough EDA, strategic feature engineering, model-driven score calculation, and insightful analysis that reveals factors influencing creditworthiness and guides potential risk mitigation strategies.

Suggestions for learners:

Exploratory Data Analysis (EDA):

- Perform a comprehensive EDA to understand the data's structure, characteristics, distributions, and relationships.
- Identify and address any missing values, mismatch data types, inconsistencies, or outliers.
- Utilize appropriate visualizations (e.g., histograms, scatter plots, box plots, correlation matrices) to uncover patterns and insights.

Feature Engineering:

- Create new features that can be leveraged for the calculation of credit scores based on domain knowledge and insights from EDA.
- Aggregate the data on the customer level if required

Hypothetical Credit Score Calculation:

- Develop a methodology to calculate a hypothetical credit score using relevant features(use a minimum of 5 maximum of 10 features).
- Clearly outline the developed methodology in the notebook, providing a detailed explanation of the reasoning behind it. (use inspiration from FICO scores and try to use relevant features you created)
- Explore various weighting schemes to assign scores.
- Provide a score for each individual customer

Analysis and Insights

- Add valuable insights from EDA and credit score calculation
- Can credit score and aggregated features be calculated at different time frames like the last 3 months/last 6 months (recency based metrics)

Remember, your analysis isn't just about dissecting data but uncovering actionable insights. Create a credit score strategy that you think would be the best and mention your justifications for criteria, weightage for the features.
Suggestions are just general guidelines for the projects. It is not limited by that but serves as a starter and keeps it open to let you explore more, go into as much depth as you can, and actually make it your own project.

Credit EDA & Credit Score Calculation with Python

```
# IMPORT ALL THE REQUIRED LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from IPython.display import Image, display

image_path = '/content/drive/MyDrive/Credit Analytics EDA/download.jpg'

width, height = 1000, 300

display(Image(filename=image_path, width=width, height=height))
```

Reading the Data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
credit = pd.read_csv('/content/drive/MyDrive/Credit Analytics EDA/Credit_score.csv')
```

```
<ipython-input-5-aa7755b4d1e2>:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on import or set low_memory=Fal:
  credit = pd.read_csv('/content/drive/MyDrive/Credit Analytics EDA/Credit_score.csv')
```

```
credit.head()
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Mont
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	

5 rows × 27 columns

EDA

```
credit.shape
# 1L rows and 27 columns are present in the given data.
```

```
(100000, 27)
```

```
credit.describe()
```

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Dela
count	84998.000000	100000.000000	100000.00000	100000.000000	
mean	4194.170850	17.091280	22.47443	72.466040	
std	3183.686167	117.404834	129.05741	466.422621	
min	303.645417	-1.000000	0.00000	1.000000	
25%	1625.568229	3.000000	4.00000	8.000000	
50%	3093.745000	6.000000	5.00000	13.000000	
75%	5957.448333	7.000000	7.00000	20.000000	
max	15204.633330	1798.000000	1499.00000	5797.000000	

```
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 27 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  object
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  object
 7   Annual_Income             100000 non-null  object
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  object
 13  Type_of_Loan              88592 non-null   object
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   object
 16  Changed_Credit_Limit      100000 non-null  object
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object
 19  Outstanding_Debt          100000 non-null  object
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  object
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object
 25  Payment_Behaviour         100000 non-null  object
 26  Monthly_Balance           98800 non-null   object
dtypes: float64(4), int64(4), object(19)
memory usage: 20.6+ MB
```

--> There are 1L rows and 27 columns in the raw data.

--> ID is the primary key.

```
#counting the null values in the dataframe
credit.isna().sum()
```

```
ID                             0
Customer_ID                    0
Month                          0
Name                        9985
Age                            0
SSN                            0
Occupation                     0
Annual_Income                  0
Monthly_Inhand_Salary      15002
Num_Bank_Accounts              0
Num_Credit_Card                0
```

```
Interest_Rate                  0
Num_of_Loan                    0
Type_of_Loan               11408
Delay_from_due_date            0
Num_of_Delayed_Payment      7002
Changed_Credit_Limit           0
Num_Credit_Inquiries        1965
Credit_Mix                     0
Outstanding_Debt               0
Credit_Utilization_Ratio       0
Credit_History_Age          9030
Payment_of_Min_Amount          0
Total_EMI_per_month            0
Amount_invested_monthly     4479
Payment_Behaviour              0
Monthly_Balance             1200
dtype: int64
```

```
credit.dtypes
```

```
ID                           object
Customer_ID                  object
Month                        object
Name                         object
Age                          object
SSN                          object
Occupation                   object
Annual_Income                object
Monthly_Inhand_Salary       float64
Num_Bank_Accounts             int64
Num_Credit_Card               int64
Interest_Rate                 int64
Num_of_Loan                  object
Type_of_Loan                 object
Delay_from_due_date           int64
Num_of_Delayed_Payment       object
Changed_Credit_Limit         object
Num_Credit_Inquiries        float64
Credit_Mix                   object
Outstanding_Debt             object
Credit_Utilization_Ratio    float64
Credit_History_Age           object
Payment_of_Min_Amount        object
Total_EMI_per_month         float64
Amount_invested_monthly      object
Payment_Behaviour            object
Monthly_Balance              object
dtype: object
```

DATA CLEANING and ANALYSIS

```
# NULL VALUES are imputed and some exception or errors are handled for each columns
```

Columns ID - No issues

Customer_ID - No issues

Month - No issues

Name - Missing values is present (imputed with Customer_ID logic)

Age - Missing values and wrong entries (imputed with Customer_ID logic)

SSN - Missing values and wrong entries (imputed with Customer_ID logic)

Occupation - Missing values and wrong entries (imputed with Customer_ID logic)

Anuual_Income - Missing values and wrong entries (imputed with Customer_ID logic)

Monthly_Inhand_Salary - Missing values and wrong entries (imputed with Customer_ID logic)

Num_Bank_Accounts - Wrong entries

Num_Credit_Card = Wrong entries

Interest Rate - Wrong entries

Num_of_Loan - Wrong entries

Type_of_Loan - Wrong entries

```
credit.columns
```

```
Index(['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance'],
      dtype='object')
```

Credit History Age

```
credit['Credit_History_Age'].unique()
```

```
array(['22 Years and 1 Months', nan, '22 Years and 3 Months',
       '22 Years and 4 Months', '22 Years and 5 Months',
       '22 Years and 6 Months', '22 Years and 7 Months',
       '26 Years and 7 Months', '26 Years and 8 Months',
       '26 Years and 9 Months', '26 Years and 10 Months',
       '26 Years and 11 Months', '27 Years and 0 Months',
       '27 Years and 1 Months', '27 Years and 2 Months',
       '17 Years and 9 Months', '17 Years and 10 Months',
       '17 Years and 11 Months', '18 Years and 1 Months',
       '18 Years and 2 Months', '18 Years and 3 Months',
       '18 Years and 4 Months', '17 Years and 3 Months',
       '17 Years and 4 Months', '17 Years and 5 Months',
       '17 Years and 6 Months', '17 Years and 7 Months',
       '17 Years and 8 Months', '30 Years and 8 Months',
       '30 Years and 9 Months', '30 Years and 10 Months',
       '30 Years and 11 Months', '31 Years and 0 Months',
       '31 Years and 1 Months', '31 Years and 2 Months',
       '31 Years and 3 Months', '32 Years and 0 Months',
       '32 Years and 2 Months', '32 Years and 3 Months',
       '32 Years and 5 Months', '32 Years and 6 Months',
       '30 Years and 7 Months', '14 Years and 8 Months',
       '14 Years and 9 Months', '14 Years and 10 Months',
       '14 Years and 11 Months', '15 Years and 0 Months',
       '15 Years and 1 Months', '15 Years and 2 Months',
       '21 Years and 4 Months', '21 Years and 5 Months',
       '21 Years and 6 Months', '21 Years and 7 Months',
       '21 Years and 8 Months', '21 Years and 9 Months',
       '21 Years and 10 Months', '21 Years and 11 Months',
       '26 Years and 6 Months', '19 Years and 2 Months',
       '19 Years and 3 Months', '19 Years and 4 Months',
       '19 Years and 5 Months', '19 Years and 6 Months',
       '19 Years and 7 Months', '19 Years and 8 Months',
       '25 Years and 5 Months', '25 Years and 6 Months',
       '25 Years and 7 Months', '25 Years and 8 Months',
       '25 Years and 9 Months', '25 Years and 10 Months',
       '25 Years and 11 Months', '26 Years and 0 Months',
       '27 Years and 3 Months', '27 Years and 4 Months',
       '27 Years and 5 Months', '8 Years and 11 Months',
       '9 Years and 0 Months', '9 Years and 1 Months',
       '9 Years and 2 Months', '9 Years and 3 Months',
       '9 Years and 4 Months', '9 Years and 6 Months',
       '18 Years and 5 Months', '18 Years and 6 Months',
       '18 Years and 8 Months', '18 Years and 9 Months',
       '16 Years and 10 Months', '16 Years and 11 Months',
       '17 Years and 0 Months', '17 Years and 1 Months',
       '17 Years and 2 Months', '29 Years and 2 Months',
       '29 Years and 3 Months', '29 Years and 4 Months',
       '29 Years and 6 Months', '29 Years and 8 Months',
       '29 Years and 9 Months', '6 Years and 5 Months',
       '6 Years and 6 Months', '6 Years and 7 Months',
       '6 Years and 8 Months', '6 Years and 9 Months',
       '6 Years and 10 Months', '6 Years and 11 Months',
       '7 Years and 0 Months', '27 Years and 6 Months',
       '27 Years and 7 Months', '27 Years and 8 Months',
       '27 Years and 9 Months', '18 Years and 7 Months',
       '19 Years and 9 Months', '19 Years and 10 Months',
       '10 Years and 1 Months', '10 Years and 2 Months',
       '10 Years and 3 Months', '10 Years and 4 Months',
```

The above Credit_History_Age column in in string format.

The value has a pattern for each customer the Year is same and month is based on the row month.

This pattern can be used to impute the null values.

The string can be converted in to float.

```
# a cilumn Years is created with the year part alone and month part is added to it in the later part.
credit['Years'] = credit['Credit_History_Age'].str.extract('(\d+)')
```

In the columns 'Name','SSN','Age','Occupation','Annual_Income','Monthly_Inhand_Salary','Num_Bank_Accounts','Num_Credit_Card','Interest_Rate', 'Num_of_Loan', 'Type_of_Loan','Num_Credit_Inquiries','Years'

I could see a lot of missing values but the fact the these columns are same for each customer id so in the missing values I can impute the mode for each customer_id.

```
column = ['Name','SSN','Age','Occupation','Annual_Income','Monthly_Inhand_Salary','Num_Bank_Accounts','Num_Credit_Card','Interest_Rate',
          'Type_of_Loan','Num_Credit_Inquiries','Years']
```

```
credit['Age'] = credit['Age'].replace('-500', np.nan)
```

```
credit['Age'] = credit['Age'].replace('44_', np.nan)
```

```
credit['Age'] = credit['Age'].replace('41_', np.nan)
```

```
result = credit.groupby('Customer_ID')[column].apply(lambda x: x.mode().iloc[0]).reset_index()
```

```
result
```

	Customer_ID	Name	SSN	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card
0	CUS_0x1000	Alistair Barrf	913-74-1218	17	Lawyer	30625.94	2706.161667	6.0	5.0
1	CUS_0x1009	Arunah	063-67-6938	26	Mechanic	52312.68	4250.390000	6.0	5.0
2	CUS_0x100b	Shirboni	238-62-0395	18	Media_Manager	113781.39	9549.782500	1.0	4.0
3	CUS_0x1011	Schneyerh	793-05-8223	44	Doctor	58918.47	5208.872500	3.0	3.0
4	CUS_0x1013	Cameront	930-49-9615	44	Mechanic	98620.98	7962.415000	3.0	3.0
...	...	...	...	...	...	...	...	...	...
12495	CUS_0xff3	Somervilled	726-35-5322	55	Scientist	17032.785	1176.398750	0.0	6.0
12496	CUS_0xff4	Poornimaf	655-05-7666	37	Entrepreneur	25546.26	2415.855000	8.0	7.0
12497	CUS_0xff6	Shieldsb	541-92-8371	19	Doctor	117639.92	9727.326667	5.0	6.0
12498	CUS_0xffc	Brads	226-86-7294	17	Musician	60877.17	5218.097500	6.0	8.0
12499	CUS_0xffd	Damouniq	832-88-8320	29	Scientist	41398.44	3749.870000	8.0	7.0

12500 rows × 14 columns

```
credit_copy = credit.copy()
```

```
credit_copy.head()
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Cred
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	...	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	...	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	NaN	821-00-0265	Scientist	19114.12	NaN	3	...	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	...	
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	...	

5 rows × 28 columns

```
credit_lookup = pd.merge(credit, result, on= 'Customer_ID', how = 'left')
```

```
filteres_credit_lookup = credit_lookup.filter(regex='^(?!.*_x$)')
```

```
filteres_credit_lookup.head()
```

	ID	Customer_ID	Month	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Credit_Mix	Outstanding_Debt	Cr
0	0x1602	CUS_0xd40	January	3	7	11.27	_	809.98	
1	0x1603	CUS_0xd40	February	-1	NaN	11.27	Good	809.98	
2	0x1604	CUS_0xd40	March	3	7	_	Good	809.98	
3	0x1605	CUS_0xd40	April	5	4	6.27	Good	809.98	
4	0x1606	CUS_0xd40	May	6	NaN	11.27	Good	809.98	

5 rows × 28 columns

```
filteres_credit_lookup.isna().sum()
```

```
ID                           0
Customer_ID                  0
Month                        0
Delay_from_due_date          0
Num_of_Delayed_Payment    7002
Changed_Credit_Limit         0
Credit_Mix                   0
Outstanding_Debt             0
Credit_Utilization_Ratio     0
Credit_History_Age        9030
Payment_of_Min_Amount        0
Total_EMI_per_month          0
Amount_invested_monthly   4479
Payment_Behaviour            0
Monthly_Balance           1200
Name_y                       0
SSN_y                        0
Age_y                        0
```

```
Occupation_y                0
Annual_Income_y             0
Monthly_Inhand_Salary_y     0
Num_Bank_Accounts_y         0
Num_Credit_Card_y           0
Interest_Rate_y             0
Num_of_Loan_y               0
Type_of_Loan_y          11408
Num_Credit_Inquiries_y      0
Years_y                     0
dtype: int64
```

The above metioned columns are cleaned.

Credit Mix

Credit mix is an ordinal category so can be encoded with values.

```
filteres_credit_lookup['Credit_Mix'] = filteres_credit_lookup['Credit_Mix'].replace('_', np.nan)
```

```
<ipython-input-26-b4b713f75b8e>: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
  filteres_credit_lookup['Credit_Mix'] = filteres_credit_lookup['Credit_Mix'].replace('_', np.nan)
```

```
#We are replacing _ to Nan values so we can easily impute the values using fillna
```

```
filteres_credit_lookup['Credit_Mix'].value_counts()
```

```
Standard    36479
Good        24337
Bad         18989
Name: Credit_Mix, dtype: int64
```

```
# NaN can be replaced by the mode of each customer_id
```

```
mode_by_customer = filteres_credit_lookup.groupby('Customer_ID')['Credit_Mix'].agg(lambda x: x.mode().iloc[0])
mode_by_customer
```

```
Customer_ID
CUS_0x1000         Bad
CUS_0x1009    Standard
CUS_0x100b        Good
CUS_0x1011    Standard
CUS_0x1013        Good
                ...
CUS_0xff3         Good
CUS_0xff4     Standard
CUS_0xff6         Good
CUS_0xffc          Bad
CUS_0xffd     Standard
Name: Credit_Mix, Length: 12500, dtype: object
```

```
filteres_credit_lookup['Credit_Mix_y'] = filteres_credit_lookup.groupby('Customer_ID')['Credit_Mix'].transform(lambda x: x.fillna(x.mode
```

```
<ipython-input-31-afca6ec09cc6>: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
  filteres_credit_lookup['Credit_Mix_y'] = filteres_credit_lookup.groupby('Customer_ID')['Credit_Mix'].transform(lambda x: x.fillna
```

```
filteres_credit_lookup.drop(columns = ['Credit_Mix'],inplace = True)
```

```
<ipython-input-32-e510132646ff>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  filteres_credit_lookup.drop(columns = ['Credit_Mix'],inplace = True)
```

```
filteres_credit_lookup['Credit_Mix_y'].value_counts()
```

```
Standard    45848
Good        30384
Bad         23768
Name: Credit_Mix_y, dtype: int64
```

```
def fun(x):
  if x == 'Good':
    return 3
  elif x == 'Standard':
    return 2
  elif x == 'Bad':
    return 1
  else:
    return 0
```

```
filteres_credit_lookup['Credit_mix_encoded'] = filteres_credit_lookup['Credit_Mix_y'].apply(fun)
```

```
filteres_credit_lookup
```

```
<ipython-input-35-1fdd2eeb870c>: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-
  filteres_credit_lookup['Credit_mix_encoded'] = filteres_credit_lookup['Credit_Mix_y'].apply(fun)
```

	ID	Customer_ID	Month	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Outstanding_Debt	Credit_Ut
0	0x1602	CUS_0xd40	January	3	7	11.27	809.98	
1	0x1603	CUS_0xd40	February	-1	NaN	11.27	809.98	
2	0x1604	CUS_0xd40	March	3	7	_	809.98	
3	0x1605	CUS_0xd40	April	5	4	6.27	809.98	
4	0x1606	CUS_0xd40	May	6	NaN	11.27	809.98	
...	...	...	...	...	...	...	...	
99995	0x25fe9	CUS_0x942c	April	23	7	11.5	502.38	
99996	0x25fea	CUS_0x942c	May	18	7	11.5	502.38	
99997	0x25feb	CUS_0x942c	June	27	6	11.5	502.38	
99998	0x25fec	CUS_0x942c	July	20	NaN	11.5	502.38	
99999	0x25fed	CUS_0x942c	August	18	6	11.5	502.38	

100000 rows × 29 columns

Cleaning Credit History Age (Adding years and months)

```
filteres_credit_lookup['Years_y'] = filteres_credit_lookup['Years_y'].astype(int)
```

```
<ipython-input-36-5e674ccb3b7d>: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
  filteres_credit_lookup['Years_y'] = filteres_credit_lookup['Years_y'].astype(int)
```

```
filteres_credit_lookup['Month'].value_counts()
```

```
January     12500
February    12500
March       12500
April       12500
May         12500
June        12500
July        12500
August      12500
Name: Month, dtype: int64
```

```
def fun(x,y):
  if x == 'January':
    return y + 1/12
  elif x == 'February':
    return y + 2/12
  elif x == 'March':
    return y + 3/12
  elif x == 'April':
    return y + 4/12
  elif x == 'May':
    return y + 5/12
  elif x == 'June':
    return y + 6/12
  elif x == 'July':
    return y + 7/12
  elif x == 'August':
    return y + 8/12
  elif x == 'September':
    return y + 9/12
  elif x == 'October':
    return y + 10/12
  elif x == 'November':
    return y + 11/12
  elif x == 'December':
    return y + 12/12
  else:
    return y
```

```
filteres_credit_lookup['Result'] = filteres_credit_lookup.apply(lambda row: fun(row['Month'], row['Years_y']), axis=1)
```

```
<ipython-input-39-69a8cca0d973>: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
  filteres_credit_lookup['Result'] = filteres_credit_lookup.apply(lambda row: fun(row['Month'], row['Years_y']), axis=1)
```

```
df  = filteres_credit_lookup.copy()
```

```
df.drop(columns = ['Credit_History_Age','Credit_Mix_y','Years_y'],inplace = True)
```

```
df.rename(columns={'Result': 'Credit_History_Age'}, inplace=True)
```

```
df.head()
```

	ID	Customer_ID	Month	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Outstanding_Debt	Credit_Utiliza
0	0x1602	CUS_0xd40	January	3	7	11.27	809.98	

Cleaning Num of Delayed Payment

```
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].str.replace('_','').astype('float')
```

```
m = df.groupby('Customer_ID')['Num_of_Delayed_Payment'].transform('mean')
```

```
m
```

```
0        6.0
1        6.0
2        6.0
3        6.0
4        6.0
        ...
99995    6.4
99996    6.4
99997    6.4
99998    6.4
99999    6.4
Name: Num_of_Delayed_Payment, Length: 100000, dtype: float64
```

```
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].fillna(m)
```

Cleaning Amount_invested_monthly column

```
df['Amount_invested_monthly'] = df['Amount_invested_monthly'].str.replace('_','').astype('float')
```

```
df['Amount_invested_monthly']
```

```
0         80.415295
1        118.280222
2         81.699521
3        199.458074
4         41.420153
            ...
99995     60.971333
99996     54.185950
99997     24.028477
99998    251.672582
99999    167.163865
Name: Amount_invested_monthly, Length: 100000, dtype: float64
```

```
m = df.groupby('Customer_ID')['Amount_invested_monthly'].transform('mean')
```

```
df['Amount_invested_monthly'] = df['Amount_invested_monthly'].fillna(m)
```

Cleaning Monthly_balance

```
df['Monthly_Balance'] = df['Monthly_Balance'].str.replace('_','').astype('float')
```

```
df['Monthly_Balance']
```

```
0        312.494089
1        284.629163
2        331.209863
3        223.451310
4        341.489231
            ...
99995           NaN
99996           NaN
99997           NaN
99998           NaN
99999           NaN
Name: Monthly_Balance, Length: 100000, dtype: float64
```

```
m = df.groupby('Customer_ID')['Monthly_Balance'].transform('mean')
```

```
m
```

```
0        304.555294
1        304.555294
2        304.555294
3        304.555294
4        304.555294
            ...
99995           NaN
99996           NaN
99997           NaN
99998           NaN
99999           NaN
Name: Monthly_Balance, Length: 100000, dtype: float64
```

```
df['Monthly_Balance'] = df['Monthly_Balance'].fillna(0)
```

Cleaning Type_of_loan_y

```
df.loc[df['Type_of_Loan_y'][df['Type_of_Loan_y'].isna()].index]
```

	ID	Customer_ID	Month	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Outstanding_Debt	Credit_Ut
32	0x1632	CUS_0x1cdb	January	5	14.833333	2.58	943.86	
33	0x1633	CUS_0x1cdb	February	9	14.833333	2.58	943.86	
34	0x1634	CUS_0x1cdb	March	5	12.000000	2.58	943.86	
35	0x1635	CUS_0x1cdb	April	1	15.000000	2.58	943.86	
36	0x1636	CUS_0x1cdb	May	9	17.000000	2.58	943.86	
...	...	...	...	...	...	...	...	
99939	0x25f95	CUS_0xad4f	April	27	19.000000	5.31	642.46	
99940	0x25f96	CUS_0xad4f	May	30	18.000000	4.31	642.46	
99941	0x25f97	CUS_0xad4f	June	27	18.000000	5.31	642.46	
99942	0x25f98	CUS_0xad4f	July	27	17.000000	1.31	642.46	
99943	0x25f99	CUS_0xad4f	August	27	15.000000	5.31	642.46	

11408 rows × 27 columns

For some users we dont have the type of loan value in that case we have imputed Not Specified value.

```
df['Type_of_Loan_y'] = df['Type_of_Loan_y'].fillna('Not Specified')
```

Below is the transformed and cleaned data...

```
df.isna().sum()
```

```
ID                          0
Customer_ID                 0
Month                       0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Name_y                      0
SSN_y                       0
Age_y                       0
Occupation_y                0
Annual_Income_y             0
Monthly_Inhand_Salary_y     0
Num_Bank_Accounts_y         0
Num_Credit_Card_y           0
Interest_Rate_y             0
Num_of_Loan_y               0
Type_of_Loan_y              0
Num_Credit_Inquiries_y      0
Credit_mix_encoded          0
```

```
Credit_History_Age          0
dtype: int64
```

ANALYSIS ON COLUMNS

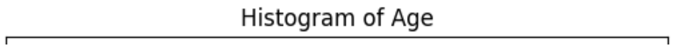
```
df['Age_y'].value_counts().sort_values()
```

```
56     328
47    1208
14    1256
54    1320
50    1328
51    1344
49    1352
52    1376
55    1416
53    1424
48    1464
16    1480
17    1568
15    1600
46    1704
18    2568
33    2640
45    2648
42    2664
24    2664
21    2744
40    2744
43    2744
29    2768
23    2792
30    2816
20    2816
19    2832
41    2840
32    2856
22    2896
35    2904
34    2920
27    2920
37    2920
44    2936
39    2952
36    2952
38    2992
25    3016
26    3048
31    3104
28    3136
Name: Age_y, dtype: int64
```

```
df['Age_y'] = df['Age_y'].astype('int')
```

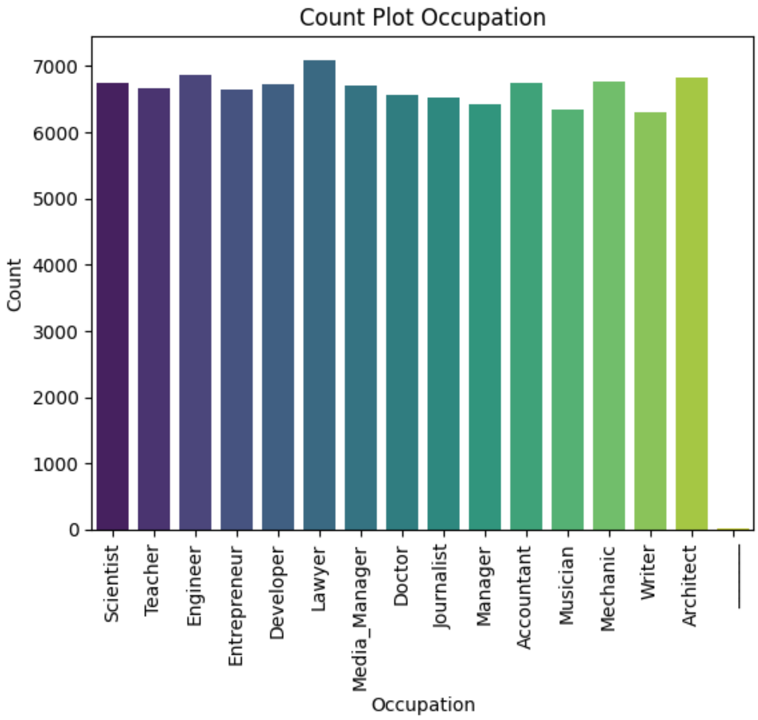
```
sns.histplot(data=df, x='Age_y', bins=10, kde=True, color='blue')
plt.xlabel('Age')
plt.ylabel('Count of Users')
plt.title('Histogram of Age')
```

```
Text(0.5, 1.0, 'Histogram of Age')
```



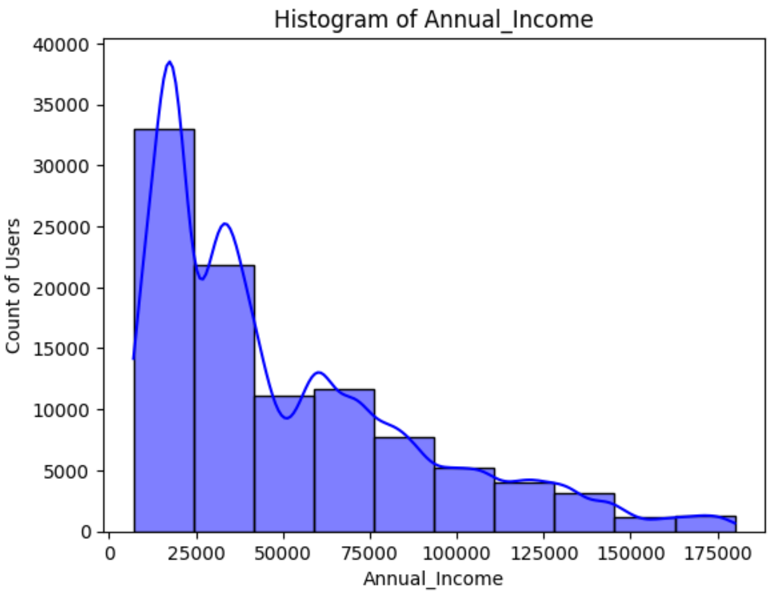
Credit is concentrared between the age of 20 to 45 (the working class)

```
ax = sns.countplot(data=df, x='Occupation_y', palette='viridis')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.xlabel('Occupation')
plt.ylabel('Count')
plt.title('Count Plot Occupation')

# Show the plot
plt.show()
```




```
df['Annual_Income_y'] = df['Annual_Income_y'].str.replace('_','').astype('float')
```

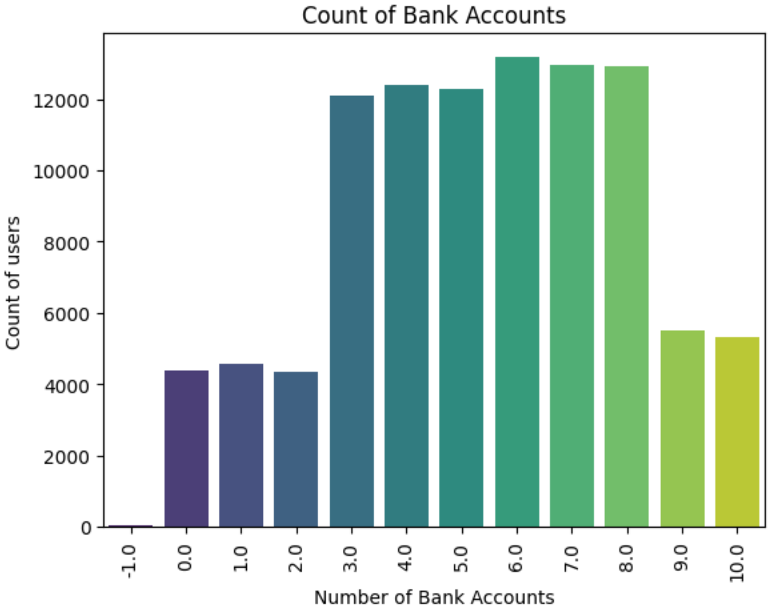
```
sns.histplot(data=df, x='Annual_Income_y', bins=10, kde=True, color='blue')
plt.xlabel('Annual_Income')
plt.ylabel('Count of Users')
plt.title('Histogram of Annual_Income')
plt.show()
```



The Annual salary of people getting loan are concaentrated towards the left end.Which proves the fact of pareto.

This gives us a conclusion of over spending bahaviour of people earning less which makes them to get more and more credit.

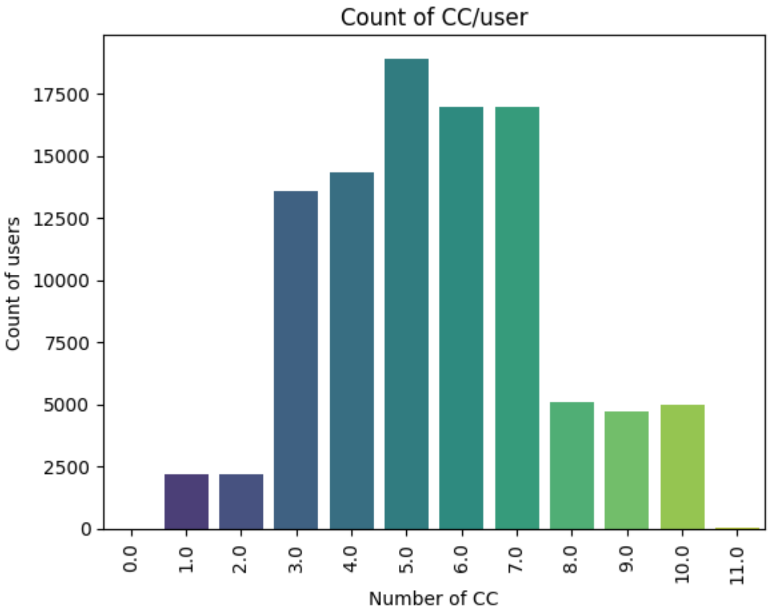
```
ax = sns.countplot(data=df, x='Num_Bank_Accounts_y', palette='viridis')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.xlabel('Number of Bank Accounts')
plt.ylabel('Count of users')
plt.title('Count of Bank Accounts')

# Show the plot
plt.show()
```



Lets ignore -1

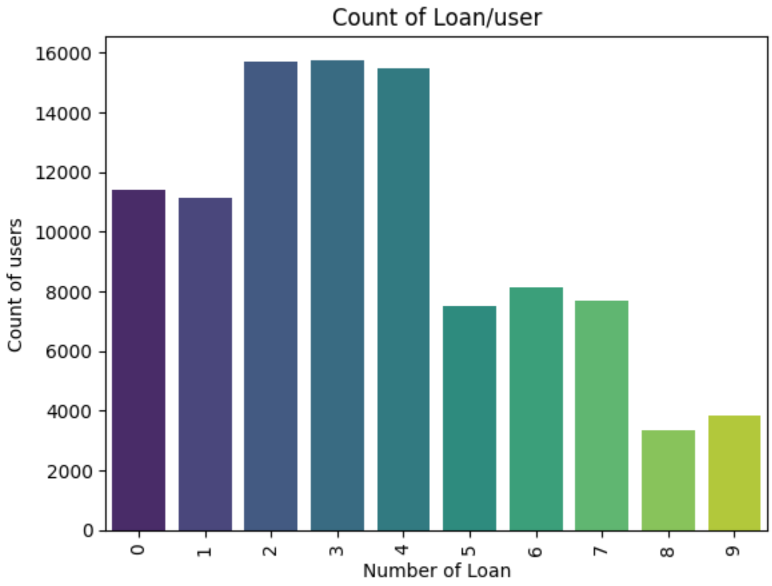
So most of the people have more than 4 accounts.

```
ax = sns.countplot(data=df, x='Num_Credit_Card_y', palette='viridis')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.xlabel('Number of CC')
plt.ylabel('Count of users')
plt.title('Count of CC/user')

# Show the plot
plt.show()
```



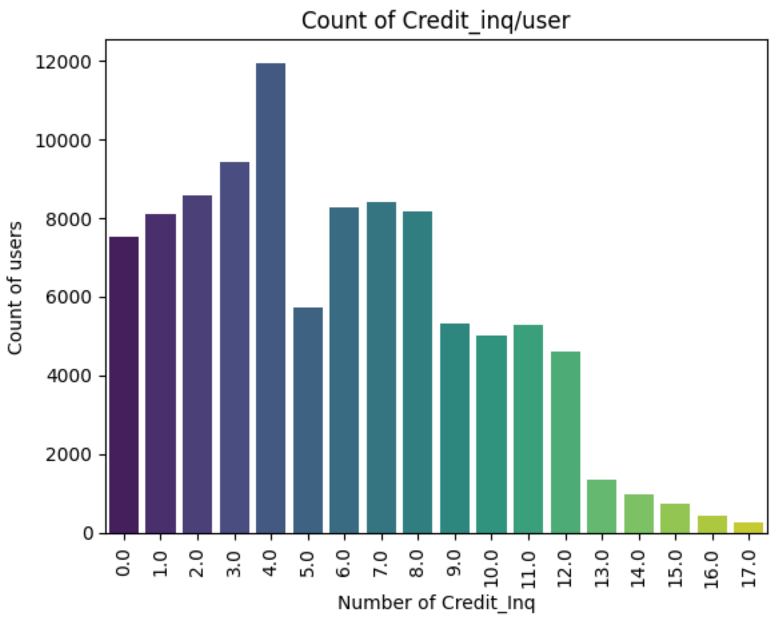
Most of the people own more than 5 credit cards.

```
df['Num_of_Loan_y'] = df['Num_of_Loan_y'].str.replace('_','').astype('int')


ax = sns.countplot(data=df, x='Num_of_Loan_y', palette='viridis')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.xlabel('Number of Loan')
plt.ylabel('Count of users')
plt.title('Count of Loan/user')

# Show the plot
plt.show()
```



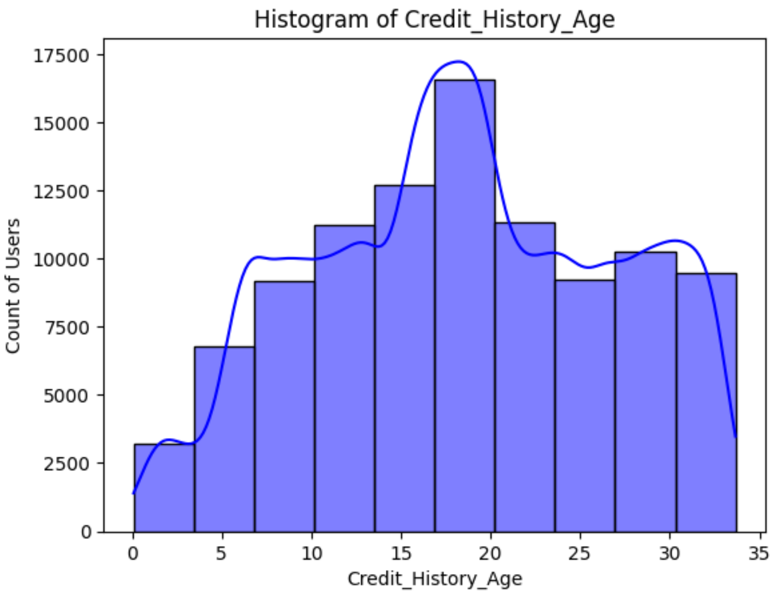
On an average people have around 3-4 loans.

```
ax = sns.countplot(data=df, x='Num_Credit_Inquiries_y', palette='viridis')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.xlabel('Number of Credit_Inq')
plt.ylabel('Count of users')
plt.title('Count of Credit_inq/user')

# Show the plot
plt.show()
```



As imagined the graph is left skewed and the average seems to 4 Credit inquries.

```
sns.histplot(data=df, x='Credit_History_Age', bins=10, kde=True, color='blue')
plt.xlabel('Credit_History_Age')
plt.ylabel('Count of Users')
plt.title('Histogram of Credit_History_Age')
plt.show()
```



The average Credit age of our user is between 16 - 21

```
df['Credit_mix_encoded'].value_counts()
```

```
2    45848
3    30384
1    23768
Name: Credit_mix_encoded, dtype: int64
```

The standard customers are 50% more compared to good customers.

On the other hand we have a large number of bad customers.

```
df["Payment_of_Min_Amount"] = df["Payment_of_Min_Amount"].replace({"Yes": 1, "No": 0, "NM": 0})
```

```
df["Payment_of_Min_Amount"].value_counts()
```

```
1    52326
0    47674
Name: Payment_of_Min_Amount, dtype: int64
```

If someone defaults the min payment amount then the person seems riskier.

In our data more than 60% pay atleast their min amount.

```
df['Payment_Behaviour'].value_counts()
```

```
Low_spent_Small_value_payments      25513
High_spent_Medium_value_payments    17540
Low_spent_Medium_value_payments     13861
High_spent_Large_value_payments     13721
High_spent_Small_value_payments     11340
Low_spent_Large_value_payments      10425
!@9#%8                               7600
Name: Payment_Behaviour, dtype: int64
```

```
df["Payment_Behaviour"] = df["Payment_Behaviour"].replace({
      "Low_spent_Small_value_payments": 1,
      "High_spent_Medium_value_payments": 2,
      "Low_spent_Medium_value_payments": 3,
      "High_spent_Large_value_payments": 4,
      "High_spent_Small_value_payments": 5,
      "Low_spent_Large_value_payments": 6
  })
```

From our data we can conclude most of the users belong to Low_spent_Small_value_payments we can try pushing some offers to make them spend more.

Credit EDA & Credit Score Calculation

Feature Engineering

```
df['Outstanding_Debt'] = df['Outstanding_Debt'].str.replace('_','').astype('float')
```

```
df['Monthly_Inhand_Salary_y'] = df['Monthly_Inhand_Salary_y'].astype('float')
```

```
#Debt to Income ratio
df['Monthly_Debt_to_Income_Ratio'] = df['Outstanding_Debt'] / df['Monthly_Inhand_Salary_y']
```

```
features = ['Monthly_Debt_to_Income_Ratio','Credit_History_Age','Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
            'Num_Credit_Inquiries_y', 'Outstanding_Debt', 'Credit_Utilization_Ratio']
```

```
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].replace('',np.nan).fillna(0)
```

```
0        2091
8.22      135
11.5      127
11.32     126
7.35      121
         ...
-5.78       1
30.1        1
35.89       1
-3.67       1
21.17       1
Name: Changed_Credit_Limit, Length: 3635, dtype: int64
```

```
# features = ['Monthly_Debt_to_Income_Ratio','Credit_History_Age','Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit
#             'Num_Credit_Inquiries_y', 'Outstanding_Debt', 'Credit_Utilization_Ratio']

# # Define weights for each feature
# weights = {'Credit_History_Age': 0.15}
```

```
# df['Credit_Score'] = df[features].multiply(pd.Series(weights), axis=1).sum(axis=1)
```

```
# df['Credit_Score'] = 900 + df['Credit_Score']

# df
```

	ID	Customer_ID	Month	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Outstanding_Debt	Credit_Ut
0	0x1602	CUS_0xd40	January	3	7.0	11.27	809.98	
1	0x1603	CUS_0xd40	February	-1	6.0	11.27	809.98	
2	0x1604	CUS_0xd40	March	3	7.0	_	809.98	

```
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].astype('float')
```

Credit Score Calculation

Monthly_Debt_to_Income_Ratio (Weight: -0.1):

A higher debt-to-income ratio may indicate financial stress. Assigning a negative weight suggests that a lower ratio contributes positively to the credit score. Credit_History_Age (Weight: 0.2):

Longer credit history is often associated with better creditworthiness. Assigning a positive weight indicates that a longer credit history contributes positively to the credit score. Delay_from_due_date (Weight: -0.05):

Delays in payments can be indicative of financial instability. Assigning a negative weight suggests that a shorter delay contributes positively to the credit score. Num_of_Delayed_Payment (Weight: -0.15):

A higher number of delayed payments may suggest a higher risk. Assigning a negative weight suggests that a lower number of delayed payments contributes positively to the credit score. Changed_Credit_Limit (Weight: 0.3):

Changes in credit limit may reflect changes in financial stability. Assigning a positive weight indicates that positive changes contribute positively to the credit score. Num_Credit_Inquiries_y (Weight: -0.1):

Multiple credit inquiries may indicate financial distress. Assigning a negative weight suggests that a lower number of credit inquiries contributes positively to the credit score. Outstanding_Debt (Weight: 0.25):

Higher outstanding debt may indicate higher risk. Assigning a positive weight suggests that lower outstanding debt contributes positively to the credit score. Credit_Utilization_Ratio (Weight: 0.2):

A lower credit utilization ratio is generally considered positive. Assigning a positive weight indicates that a lower ratio contributes positively to the credit score.

```
feature_weights = {'Monthly_Debt_to_Income_Ratio': -0.1,'Credit_History_Age': 0.2,'Num_of_Delayed_Payment': -0.15,
                   'Num_Credit_Inquiries_y': -0.1,'Outstanding_Debt': 0.25,'Credit_Utilization_Ratio': 0.2,'Changed_Credit_Limit':0.30}

# Calculate the weighted sum for each row
df['Credit_Score'] = df[features].mul(pd.Series(feature_weights)).sum(axis=1)

# Normalize the score to a 0-100 range
min_score, max_score = df['Credit_Score'].min(), df['Credit_Score'].max()
df['Credit_Score_Normalized'] = 100 * (df['Credit_Score'] - min_score) / (max_score - min_score)

# Display the resulting DataFrame with credit scores
print(df[['Credit_Score_Normalized']])
```

```
            Credit_Score_Normalized
     0                    43.703331
     1                    43.767357
     2                    43.542596
     3                    43.698550
     4                    43.693205
     ...                        ...
     99995                39.764214
     99996                39.828559
     99997                39.844935
     99998                39.760717
     99999                39.770796

     [100000 rows x 1 columns]
```

```
df['Credit_Score_Normalized'].describe()
```

```
     count    100000.000000
     mean         51.781793
```

```
std          15.578385
min           0.000000
25%          40.337662
50%          48.399838
75%          58.711237
max         100.000000
Name: Credit_Score_Normalized, dtype: float64
```

```
df.drop(columns = ['Credit_Score'],inplace = True)
```

```
grouped_score = df.groupby('Customer_ID')['Credit_Score_Normalized'].mean().reset_index()
```

```
grouped_score['Credit_Score_Normalized'].describe()
```

```
count    12500.000000
mean        51.781793
std         15.492820
min         28.054559
25%         40.314309
50%         48.334534
75%         58.701081
max         99.937687
Name: Credit_Score_Normalized, dtype: float64
```

We have 12500 unique customers.

Out of them more than 75% of the customers have less than 60 credit score.

From this it is very clear that most of the people aren't aware of the pros and cons of credit scores.

The average score is around 51