Credit EDA & Credit Score Calculation with Python

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 Features

The dataset contains the following features:

ID : Represents a unique identification of an entry

Customer_ID : Represents a unique identification of a person

Month : Represents the month of the year

Name : Represents the name of a person

Age : Represents the age of the person

SSN : Represents the social security number of a person

Occupation : Represents the occupation of the person

Annual_Income : Represents the annual income of the person

Monthly_Inhand_Salary : Represents the monthly base salary of a person

Num_Bank_Accounts : Represents the number of bank accounts a person holds

Num_Credit_Card : Represents the number of other credit cards held by a person

Interest_Rate : Represents the interest rate on credit card

Num_of_Loan : Represents the number of loans taken from the bank

Type_of_Loan : Represents the types of loan taken by a person

Delay_from_due_date : Represents the average number of days delayed from the payment date

Num_of_Delayed_Payment : Represents the average number of payments delayed by a person

Changed_Credit_Limit : Represents the percentage change in credit card limit

Num_Credit_Inquiries : Represents the number of credit card inquiries

Credit_Mix : Represents the classification of the mix of credits

Outstanding_Debt : Represents the remaining debt to be paid (in USD)

Credit_Utilization_Ratio : Represents the utilization ratio of credit card

Credit_History_Age : Represents the age of credit history of the person

Payment_of_Min_Amount : Represents whether only the minimum amount was paid by the person

Total_EMI_per_month : Represents the monthly EMI payments (in USD)

Amount_invested_monthly : Represents the monthly amount invested by the customer (in USD)

Payment_Behaviour : Represents the payment behavior of the customer (in USD)

Monthly_Balance : Represents the monthly balance amount of the customer (in USD)

Importing Libraries

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

Importing Dataset

```
data = pd.read_csv("https://drive.google.com/uc?id=1pljm6_3nxcFS9UMIFm124HBsjNZP6ACA&export=download")
```

```
<ipython-input-361-7188cf472b49>:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv("https://drive.google.com/uc?id=1pljm6_3nxcFS9UMIFm124HBsjNZP6ACA&export=download")
```

```
dt = pd.DataFrame(data)
```

```
dt.head(3)
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Num_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	-500	821-00-0265	Scientist	19114.12	NaN	3	...	

3 rows × 27 columns

```
dt.shape
```

```
(100000, 27)
```

```
dt.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')
```

```
df_c=dt.copy()
```

```
df_c.shape
```

```
(100000, 27)
```

```
df_c.head()
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Num_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	-500	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 × 27 columns

```
df_c.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
```

```
df_c.describe()
```

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Delay_from_due_date	Num_Credit_Inquiries	Credit_Utili
count	84998.000000	100000.000000	100000.00000	100000.000000	100000.000000	98035.000000	
mean	4194.170850	17.091280	22.47443	72.466040	21.068780	27.754251	
std	3183.686167	117.404834	129.05741	466.422621	14.860104	193.177339	
min	303.645417	-1.000000	0.00000	1.000000	-5.000000	0.000000	
25%	1625.568229	3.000000	4.00000	8.000000	10.000000	3.000000	
50%	3093.745000	6.000000	5.00000	13.000000	18.000000	6.000000	
75%	5957.448333	7.000000	7.00000	20.000000	28.000000	9.000000	
max	15204.633330	1798.000000	1499.00000	5797.000000	67.000000	2597.000000	

```
df_c.describe(include='object')
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Num_of_Loan	Type_of_Loan	Num_of_Delayed_P
count	100000	100000	100000	90015	100000	100000	100000	100000	100000	88592	
unique	100000	12500	8	10139	1788	12501	16	18940	434	6260	
top	0x1602	CUS_0xd40	January	Langep	38	#F%$D@*&8	_______	36585.12	3	Not Specified	
freq	1	8	12500	44	2833	5572	7062	16	14386	1408	

```
df_c.isna().sum()
```

	0
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

```
df_c.dtypes
```

	0
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

Finding Unique values

```
col = df_c.columns
col
```

```
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')
```

```
for c in col:
  print('Column Name:'+c)
  print('*'*10)
  print(df_c[c].value_counts())
  print('-'*20)
```

```
                    ...
36408.000000        1
23760.000000        1
24612.000000        1
24325.000000        1
58638.000000        1
Name: count, Length: 14950, dtype: int64
--------------------
Column Name:Amount_invested_monthly
**********
Amount_invested_monthly
__10000__      4305
0               169
80.41529544       1
36.66235139       1
89.73848936       1
                ...
36.54190859       1
93.45116319       1
140.8097222       1
38.7393767        1
167.1638652       1
Name: count, Length: 91049, dtype: int64
--------------------
Column Name:Payment_Behaviour
**********
Payment_Behaviour
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: count, dtype: int64
--------------------
Column Name:Monthly_Balance
**********
Monthly_Balance
__-333333333333333333333333333__    9
350.0148691                         2
695.0571561                         2
312.4940887                         1
604.3402009                         1
                                   ..
280.6862317                         1
366.289038                          1
151.1882696                         1
306.7502785                         1
393.673696                          1
Name: count, Length: 98790, dtype: int64
--------------------
```

Data Cleaning

```
def text_cleaning(data):
    if data is np.NaN or not isinstance(data, str):
        return data
    else:
        return str(data).strip('_ ,"')


df = df_c.applymap(text_cleaning).replace(['', 'nan', '!@9#%8', '#F%$D@*&8'], np.NaN)
df.head()
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Num_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	-500	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 × 27 columns

Removing Unwanted Columns

```
df = df.drop(['ID','Name','SSN'],axis=1)
```

```
df.head()
```

	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_
0	CUS_0xd40	January	23	Scientist	19114.12	1824.843333	3	4	3	
1	CUS_0xd40	February	23	Scientist	19114.12	NaN	3	4	3	
2	CUS_0xd40	March	-500	Scientist	19114.12	NaN	3	4	3	
3	CUS_0xd40	April	23	Scientist	19114.12	NaN	3	4	3	
4	CUS_0xd40	May	23	Scientist	19114.12	1824.843333	3	4	3	

5 rows × 24 columns

Change Data Type

```
df['Month']                  = pd.to_datetime(df.Month, format='%B').dt.month
df['Age']                    = df.Age.astype(int)
df['Annual_Income']          = df.Annual_Income.astype(float)
df['Num_of_Loan']            = df.Num_of_Loan.astype(int)
df['Num_of_Delayed_Payment'] = df.Num_of_Delayed_Payment.astype(float)
df['Changed_Credit_Limit']   = df.Changed_Credit_Limit.astype(float)
df['Outstanding_Debt']       = df.Outstanding_Debt.astype(float)
df['Amount_invested_monthly'] = df.Amount_invested_monthly.astype(float)
df['Monthly_Balance']        = df.Monthly_Balance.astype(float)
```

```
df.dtypes
```

	0
Customer_ID	object
Month	int32
Age	int64
Occupation	object
Annual_Income	float64
Monthly_Inhand_Salary	float64
Num_Bank_Accounts	int64
Num_Credit_Card	int64
Interest_Rate	int64
Num_of_Loan	int64
Type_of_Loan	object
Delay_from_due_date	int64
Num_of_Delayed_Payment	float64
Changed_Credit_Limit	float64
Num_Credit_Inquiries	float64
Credit_Mix	object
Outstanding_Debt	float64
Credit_Utilization_Ratio	float64
Credit_History_Age	object
Payment_of_Min_Amount	object
Total_EMI_per_month	float64
Amount_invested_monthly	float64
Payment_Behaviour	object
Monthly_Balance	float64

Credit History Age

```
def Month_Converter(x):
    if pd.notnull(x):
        num1 = int(x.split(' ')[0])
        num2 = int(x.split(' ')[3])

        return (num1*12)+num2
    else:
        return x
df['Credit_History_Age'] = df.Credit_History_Age.apply(lambda x: Month_Converter(x)).astype(float)


df.groupby('Customer_ID')['Credit_History_Age'].apply(list)
```

Customer_ID	Credit_History_Age
CUS_0x1000	[122.0, 123.0, 124.0, 125.0, 126.0, 127.0, 128...
CUS_0x1009	[365.0, 366.0, 367.0, nan, 369.0, 370.0, 371.0...
CUS_0x100b	[183.0, nan, 185.0, 186.0, 187.0, 188.0, 189.0...
CUS_0x1011	[183.0, 184.0, 185.0, 186.0, 187.0, 188.0, 189...
CUS_0x1013	[207.0, 208.0, 209.0, 210.0, nan, 212.0, 213.0...
...	...
CUS_0xff3	[201.0, 202.0, 203.0, 204.0, 205.0, 206.0, 207...
CUS_0xff4	[218.0, 219.0, 220.0, 221.0, 222.0, nan, 224.0...
CUS_0xff6	[292.0, 293.0, 294.0, 295.0, 296.0, 297.0, 298...
CUS_0xffc	[151.0, nan, 153.0, 154.0, 155.0, 156.0, 157.0...
CUS_0xffd	[218.0, 219.0, 220.0, 221.0, 222.0, 223.0, 224...

12500 rows × 1 columns

Diff Loan Type

```
col = df['Type_of_Loan'].dropna()
val = []
sep = ','
replace = ' '
for i in col:
  i = i.replace(' and',', ')
  for data in map(lambda x:x.strip(), re.sub(replace, ' ', i).split(sep)):
      if data and data.lower() not in val:
        val.append(data.lower())
```

```
dict(enumerate(sorted(val)))
```

```
{0: 'auto loan',
 1: 'credit-builder loan',
 2: 'debt consolidation loan',
 3: 'home equity loan',
 4: 'mortgage loan',
 5: 'not specified',
 6: 'payday loan',
 7: 'personal loan',
 8: 'student loan'}
```

Null_Fix : Objects - Mode()

Occupation

```
df['Occupation'].unique()
```

```
array(['Scientist', nan, 'Teacher', 'Engineer', 'Entrepreneur',
       'Developer', 'Lawyer', 'Media_Manager', 'Doctor', 'Journalist',
       'Manager', 'Accountant', 'Musician', 'Mechanic', 'Writer',
       'Architect'], dtype=object)
```

```
df['Occupation'].value_counts(dropna= False)
```

Occupation	count
NaN	7062
Lawyer	6575
Architect	6355
Engineer	6350
Scientist	6299
Mechanic	6291
Accountant	6271
Developer	6235
Media_Manager	6232
Teacher	6215
Entrepreneur	6174
Doctor	6087
Journalist	6085
Manager	5973
Musician	5911
Writer	5885

```
df['Occupation'] = df.groupby(['Customer_ID'])['Occupation'].transform(lambda x: x.mode()[0] if not x.mode().empty else x)
```

```
df['Occupation'].value_counts(dropna= False)
```

Occupation	count
Lawyer	7096
Engineer	6864
Architect	6824
Mechanic	6776
Scientist	6744
Accountant	6744
Developer	6720
Media_Manager	6720
Teacher	6672
Entrepreneur	6648
Doctor	6568
Journalist	6536
Manager	6432
Musician	6352
Writer	6304

Type of Loan

```
df['Type_of_Loan'].value_counts(dropna= False)
```

Type_of_Loan	count
NaN	11408
Not Specified	1408
Credit-Builder Loan	1280
Personal Loan	1272
Debt Consolidation Loan	1264
...	...
Not Specified, Mortgage Loan, Auto Loan, and Payday Loan	8
Payday Loan, Mortgage Loan, Debt Consolidation Loan, and Student Loan	8
Debt Consolidation Loan, Auto Loan, Personal Loan, Debt Consolidation Loan, Student Loan, and Credit-Builder Loan	8
Student Loan, Auto Loan, Student Loan, Credit-Builder Loan, Home Equity Loan, Debt Consolidation Loan, and Debt Consolidation Loan	8
Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan	8

6261 rows × 1 columns

```
df['Type_of_Loan'].replace([np.NaN], 'No Loan', inplace=True)

df['Type_of_Loan'].value_counts(dropna= False)
```

Type_of_Loan	count
No Loan	11408
Not Specified	1408
Credit-Builder Loan	1280
Personal Loan	1272
Debt Consolidation Loan	1264
...	...
Not Specified, Mortgage Loan, Auto Loan, and Payday Loan	8
Payday Loan, Mortgage Loan, Debt Consolidation Loan, and Student Loan	8
Debt Consolidation Loan, Auto Loan, Personal Loan, Debt Consolidation Loan, Student Loan, and Credit-Builder Loan	8
Student Loan, Auto Loan, Student Loan, Credit-Builder Loan, Home Equity Loan, Debt Consolidation Loan, and Debt Consolidation Loan	8
Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan	8

6261 rows × 1 columns

Credit Mix

```
df['Credit_Mix'].value_counts(dropna = False)
```

Credit_Mix	count
Standard	36479
Good	24337
NaN	20195
Bad	18989

```
mode_val = df.groupby(['Customer_ID'])['Credit_Mix'].transform(lambda x : x.mode()[0] if not x.mode().empty else x)
df['Credit_Mix'] = df['Credit_Mix'].fillna(mode_val)
```

```
df['Credit_Mix'].value_counts(dropna = False)
```

Credit_Mix	count
Standard	45848
Good	30384
Bad	23768

```
df[['Customer_ID','Credit_Mix']].value_counts(dropna = False).reset_index().sort_values('Customer_ID')
```

	Customer_ID	Credit_Mix	count
0	CUS_0x1000	Bad	8
1563	CUS_0x1009	Standard	8
8330	CUS_0x100b	Good	8
8331	CUS_0x1011	Standard	8
8332	CUS_0x1013	Good	8
...	...	...	...
4172	CUS_0xff3	Good	8
4173	CUS_0xff4	Standard	8
4174	CUS_0xff6	Good	8
4175	CUS_0xffc	Bad	8
12499	CUS_0xffd	Standard	8

12500 rows × 3 columns

Payment Behaviour

```
df[['Customer_ID','Payment_Behaviour']].value_counts(dropna = False).reset_index().sort_values('Customer_ID')
```

	Customer_ID	Payment_Behaviour	count
43171	CUS_0x1000	Low_spent_Small_value_payments	1
24325	CUS_0x1000	Low_spent_Large_value_payments	2
16474	CUS_0x1000	High_spent_Small_value_payments	2
14633	CUS_0x1000	High_spent_Medium_value_payments	2
43172	CUS_0x1000	NaN	1
...	...	...	...
24340	CUS_0xffd	Low_spent_Small_value_payments	2
26865	CUS_0xffd	High_spent_Large_value_payments	1
26741	CUS_0xffd	High_spent_Small_value_payments	1
26784	CUS_0xffd	Low_spent_Large_value_payments	1
56021	CUS_0xffd	NaN	1

56022 rows × 3 columns

```
df['Payment_Behaviour'] = df.groupby(['Customer_ID','Occupation'])['Payment_Behaviour'].transform(lambda x : x.fillna(x.mode()[0] if not x.m
```

```
df[['Customer_ID','Payment_Behaviour']].value_counts(dropna = False).reset_index().sort_values('Customer_ID')
```

	Customer_ID	Payment_Behaviour	count
10090	CUS_0x1000	High_spent_Medium_value_payments	3
33327	CUS_0x1000	Low_spent_Small_value_payments	1
13653	CUS_0x1000	High_spent_Small_value_payments	2
23111	CUS_0x1000	Low_spent_Large_value_payments	2
33319	CUS_0x1009	Low_spent_Medium_value_payments	1
...	...	...	...
33322	CUS_0xffd	High_spent_Large_value_payments	1
33321	CUS_0xffd	High_spent_Small_value_payments	1
33320	CUS_0xffd	Low_spent_Large_value_payments	1
11977	CUS_0xffd	High_spent_Medium_value_payments	3
24537	CUS_0xffd	Low_spent_Small_value_payments	2

50122 rows × 3 columns

Payment_of_Min_Amount

```
df['Payment_of_Min_Amount'].value_counts(dropna=False)
```

Payment_of_Min_Amount	count
Yes	52326
No	35667
NM	12007

Null_Fix : Numeric

Month

```
df['Month'].value_counts(dropna = False)
```

Month	count
1	12500
2	12500
3	12500
4	12500
5	12500
6	12500
7	12500
8	12500

Age Fix

```
df[['Age','Customer_ID']].value_counts(dropna = False).reset_index().sort_values('Customer_ID')
```

	Age	Customer_ID	count
10926	17	CUS_0x1000	5
15066	18	CUS_0x1000	3
16868	25	CUS_0x1009	2
8646	26	CUS_0x1009	6
11697	19	CUS_0x100b	4
...	...	...	...
11136	19	CUS_0xff6	5
14825	18	CUS_0xffc	3
10151	17	CUS_0xffc	5
16365	30	CUS_0xffd	2
7062	29	CUS_0xffd	6

22453 rows × 3 columns

```
df['Age'].unique()
```

```
array([  23, -500,   28, ..., 4808, 2263, 1342])
```

```
df_c = df[(df['Age'] <= 0) | (df['Age'] >= 100)]
df_c[['Customer_ID','Age']].value_counts()
```

		count
Customer_ID	**Age**	
CUS_0x1c5d	**-500**	3
CUS_0x9891	**-500**	3
CUS_0x16ae	**-500**	2
CUS_0x1758	**-500**	2
CUS_0xc2eb	**-500**	2
...	**...**	...
CUS_0x4fa3	**2638**	1
CUS_0x4faf	**1154**	1
CUS_0x4fc1	**6159**	1
CUS_0x4fd4	**-500**	1
CUS_0xff4	**-500**	1

2751 rows × 1 columns

```
df['Age'] = df.groupby(['Customer_ID'])['Age'].transform(lambda x: x.mode()[0] if not x.mode().empty else x)
```

```
df_c = df[(df['Age'] <= 0) | (df['Age'] >= 100)]
df_c[['Customer_ID','Age']].value_counts()
```

		count
Customer_ID	**Age**	

```
df['Age'].unique()
```

```
array([23, 28, 34, 55, 21, 31, 30, 44, 40, 33, 35, 39, 37, 20, 46, 26, 41,
       32, 48, 43, 36, 16, 18, 42, 22, 19, 15, 27, 38, 14, 25, 45, 47, 17,
       53, 24, 54, 29, 49, 51, 50, 52, 56])
```

Annual_Income

```
df['Annual_Income'].value_counts(dropna = False)
```

Annual_Income	count
20867.67	16
22434.16	16
32543.38	16
9141.63	16
40341.16	16
...	...
20286590.00	1
20025646.00	1
18758037.00	1
1893725.00	1
5524914.00	1

13613 rows × 1 columns

```
dg = df[['Annual_Income','Occupation','Customer_ID']].value_counts().reset_index().sort_values('Customer_ID')
dg[dg['count']<8]
```

	Annual_Income	Occupation	Customer_ID	count
11627	61194.81	Accountant	CUS_0x1018	7
12649	17117486.00	Accountant	CUS_0x1018	1
12152	86617.16	Writer	CUS_0x1057	7
13114	1105753.00	Writer	CUS_0x1057	1
12569	15741145.00	Architect	CUS_0x107e	1
...	...	...	...	...
13015	20189519.00	Musician	CUS_0xf20	1
12324	78443.48	Entrepreneur	CUS_0xf55	7
13022	20350298.00	Entrepreneur	CUS_0xf55	1
13619	7711339.00	Manager	CUS_0xfa4	1
12104	15035.19	Manager	CUS_0xfa4	7

2203 rows × 4 columns

```
df['Annual_Income'] = df.groupby(['Customer_ID','Occupation'])['Annual_Income'].transform(lambda x: x.mode()[0] if not x.mode().empty else x
```

```
dg = df[['Annual_Income','Occupation','Customer_ID']].value_counts().reset_index().sort_values('Customer_ID')
dg[dg['count']<8]
```

Annual_Income	Occupation	Customer_ID	count

```
df['Annual_Income'].value_counts(dropna = False)
```

Annual_Income	count
20867.670	16
9141.630	16
32543.380	16
40341.160	16
22434.160	16
...	...
18317.260	8
14784.450	8
60573.960	8
18413.795	8
39628.990	8

12488 rows × 1 columns

Monthly_Inhand_Salary

```
df['Monthly_Inhand_Salary'].value_counts(dropna = False)
```

Monthly_Inhand_Salary	count
NaN	15002
2295.058333	15
6082.187500	15
6769.130000	15
6358.956667	15
...	...
1087.546445	1
3189.212103	1
5640.117744	1
7727.560450	1
2443.654131	1

13236 rows × 1 columns

```
df['Monthly_Inhand_Salary'] = df.groupby(['Customer_ID','Occupation'])['Monthly_Inhand_Salary'].transform(lambda x: x.mode()[0] if not x.moc
```

```
df['Monthly_Inhand_Salary'].value_counts(dropna = False)
```

Monthly_Inhand_Salary	count
6639.560000	16
5766.491667	16
2295.058333	16
6769.130000	16
3080.555000	16
...	...
1499.731667	8
5620.681667	8
3498.526667	8
6578.040000	8
3359.415833	8

12489 rows × 1 columns

```
df['Monthly_Inhand_Salary'].isna().sum()
```

```
0
```

Num_Bank_ACC

```
df['Num_Bank_Accounts'].value_counts(dropna = False)
```

Num_Bank_Accounts	count
6	13001
7	12823
8	12765
4	12186
5	12118
...	...
1626	1
1470	1
887	1
211	1
697	1

943 rows × 1 columns

```
df['Num_Bank_Accounts'].nunique()
```

```
943
```

```
df['Num_Bank_Accounts'] = df.groupby(['Customer_ID', 'Occupation'])['Num_Bank_Accounts'].transform(lambda x: x.mode()[0] if not x.mode().emp
```

```
df['Num_Bank_Accounts'].unique()
```

```
array([ 3,  2,  1,  7,  4,  0,  8,  5,  6,  9, 10])
```

num_credit_card

```
df['Num_Credit_Card'].value_counts(dropna= False).reset_index().sort_values('Num_Credit_Card')
```

	Num_Credit_Card	count
11	0	13
9	1	2132
8	2	2149
4	3	13277
3	4	14030
...	...	...
865	1495	1
354	1496	2
209	1497	3
180	1498	3
312	1499	2

1179 rows × 2 columns

```
df['Num_Credit_Card'] = df.groupby(['Customer_ID','Occupation'])['Num_Credit_Card'].transform(lambda x: x.mode()[0] if not x.mode().empty el
```

```
df['Num_Credit_Card'].unique()
```

```
array([ 4,  5,  1,  7,  6,  8,  3,  9,  2, 10, 11,  0])
```

Interest_Rate

```
df['Interest_Rate'].value_counts(dropna = False)
```

Interest_Rate	count
8	5012
5	4979
6	4721
12	4540
10	4540
...	...
4995	1
1899	1
2120	1
5762	1
5729	1

1750 rows × 1 columns

```
df['Interest_Rate'] = df.groupby(['Customer_ID','Occupation'])['Interest_Rate'].transform(lambda x: x.mode()[0] if not x.mode().empty else x
```

```
df['Interest_Rate'].unique()
```

```
array([ 3,  6,  8,  4,  5, 15,  7, 12, 20,  1, 14, 32, 16, 17, 10, 31, 25,
       18, 19,  9, 24, 13, 33, 11, 21, 29, 28, 30, 23, 34,  2, 27, 26, 22])
```

Num_of_loan

```
df['Num_of_Loan'].value_counts(dropna = True)
```

Num_of_Loan	count
3	15104
2	15032
4	14743
0	10930
1	10606
...	...
119	1
321	1
1439	1
663	1
966	1

414 rows × 1 columns

```
df['Num_of_Loan'].unique()
```

```
array([   4,    1,    3,  967, -100,    0,    2,    7,    5,    6,    8,
          9, 1464,  622,  352,  472, 1017,  945,  146,  563,  341,  444,
        720, 1485,   49,  737, 1106,  466,  728,  313,  843,  597,  617,
        119,  663,  640,   92, 1019,  501, 1302,   39,  716,  848,  931,
       1214,  186,  424, 1001, 1110, 1152,  457, 1433, 1187,   52, 1480,
       1047, 1035, 1347,   33,  193,  699,  329, 1451,  484,  132,  649,
        995,  545,  684, 1135, 1094, 1204,  654,   58,  348,  614, 1363,
        323, 1406, 1348,  430,  153, 1461,  905, 1312, 1424, 1154,   95,
       1353, 1228,  819, 1006,  795,  359, 1209,  590,  696, 1185, 1465,
        911, 1181,   70,  816, 1369,  143, 1416,  455,   55, 1096, 1474,
        420, 1131,  904,   89, 1259,  527, 1241,  449,  983,  418,  319,
         23,  238,  638,  138,  235,  280, 1070, 1484,  274,  494, 1459,
        404, 1354, 1495, 1391,  601, 1313, 1319,  898,  231,  752,  174,
        961, 1046,  834,  284,  438,  288, 1463, 1151,  719,  198, 1015,
        855,  841,  392, 1444,  103, 1320,  745,  172,  252,  630,  241,
         31,  405, 1217, 1030, 1257,  137,  157,  164, 1088, 1236,  777,
       1048,  613,  330, 1439,  321,  661,  952,  939,  562, 1202,  302,
        943,  394,  955, 1318,  936,  781,  100, 1329, 1365,  860,  217,
        191,   32,  282,  351, 1387,  757,  416,  833,  292, 1225, 1227,
        639,  859,  243,  267,  510,  332,  996,  311,  492,  820,  336,
        123,  540,  131, 1311, 1441,  895,  891,   50,  940,  935,  596,
         29, 1182, 1129, 1014,  251,  365,  291, 1447,  742, 1085,  148,
        462,  832,  881, 1412,  785, 1127,  910,  538,  999,  733,  101,
        237,   87,  659,  633,  387,  447,  629,  831, 1384,  773,  621,
       1419,  289,  285, 1393,   27, 1359, 1482, 1189, 1294,  201,  579,
        814,  141,  581, 1171,  295,  290,  433,  679, 1040, 1054, 1430,
       1023, 1077, 1457, 1150,  701, 1382,  889,  437,  372, 1222,  126,
       1159,  868,   19, 1297,  227,  190,  809, 1216, 1074,  571,  520,
       1274, 1340,  991,  316,  697,  926,  873, 1002,  378,   65,  875,
        867,  548,  652, 1372,  606, 1036, 1300,   17, 1178,  802, 1219,
       1271, 1137, 1496,  439,  196,  636,  192,  228, 1053,  229,  753,
       1296, 1371,  254,  863,  464,  515,  838, 1160, 1289, 1298,  799,
        182,  574,  242,  415,  869,  958,   54, 1265,  656,  275,  778,
        208,  147,  350,  507,  463,  497,  927,  653,  662,  529,  635,
       1027,  897, 1039, 1345,  924, 1279,  546, 1112, 1210,  526,  300,
       1103,  504,  136, 1400,   78,  686, 1091,  344,  215,   84,  628,
       1470,  968, 1478,   83, 1196, 1307, 1132, 1008,  917,  657,   56,
         18,   41,  801,  978,  216,  349,  966])
```

```
df['Num_of_Loan'] = df.groupby(['Customer_ID','Occupation'])['Num_of_Loan'].transform(lambda x: x.mode()[0] if not x.mode().empty else x)
```

```
df['Num_of_Loan'].unique()
```

```
array([4, 1, 3, 0, 2, 7, 5, 6, 8, 9])
```

Delay from Due Date

```
df['Delay_from_due_date'].isna().sum()
```

```
0
```

Func for NaN treatment

> Find Q1 & Q3 by IQR

> Replace outliers with NaN

> Fill NaN with Most freq value - mode()

```
def nan_treat(df, grp, col):

  null_cnt = df[col].isna().sum()
  print(f"Initial NaN count in '{col}': {null_cnt}")

  q1 = df[col].quantile(.25)
  q3 = df[col].quantile(.75)
  iqr = q3-q1
  lower = q1 - iqr*1.5
  upper = q3 + iqr*1.5
  print(f'Lower: {lower}\nQ1: {q1}\nIQR: {iqr}\nQ3: {q3}\nUpper: {upper}')

  df_drop = df[df[col].notna()].groupby(grp)[col].apply(list)
  x, y = df_drop.apply(lambda x: stats.mode(x)).apply([min, max])
  mini, maxi = x[0], y[0]

  df[col] = df[col].apply(lambda x: np.NaN if ((x<mini)|(x>maxi)) else x)
  mod = df.groupby(grp)[col].transform(lambda x: x.mode()[0] if not x.mode().empty else np.NaN)
  df[col] = df[col].fillna(mod)

  null_cnt_after = df[col].isna().sum()
  print(f"NaN count in '{col}' after treatment: {null_cnt_after}")
```

Num_of_Delayed_Payment

```
df['Num_of_Delayed_Payment'].isna().sum()
```

```
7002
```

```
nan_treat(df, 'Customer_ID', 'Num_of_Delayed_Payment')
```

```
Initial NaN count in 'Num_of_Delayed_Payment': 7002
Lower: -4.5
Q1: 9.0
IQR: 9.0
Q3: 18.0
Upper: 31.5
NaN count in 'Num_of_Delayed_Payment' after treatment: 0
```

Changed_Credit_Limit

```
df['Changed_Credit_Limit'].isna().sum()
```

```
2091
```

```
nan_treat(df, 'Customer_ID', 'Changed_Credit_Limit')
```

```
Initial NaN count in 'Changed_Credit_Limit': 2091
Lower: -9.004999999999999
Q1: 5.32
IQR: 9.549999999999999
Q3: 14.87
Upper: 29.195
NaN count in 'Changed_Credit_Limit' after treatment: 0
```

Num_Credit_Inquiries

```
df['Num_Credit_Inquiries'].isna().sum()
```

```
1965
```

```
df[['Customer_ID','Num_Credit_Inquiries']].value_counts(dropna = False).reset_index().sort_values('Customer_ID')
```

	Customer_ID	Num_Credit_Inquiries	count
5159	CUS_0x1000	11.0	7
18691	CUS_0x1000	10.0	1
11679	CUS_0x1009	2.0	4
11678	CUS_0x1009	4.0	4
16739	CUS_0x100b	4.0	2
...	...	...	...
6649	CUS_0xffc	13.0	7
17659	CUS_0xffc	8.0	1
17672	CUS_0xffd	1801.0	1
8373	CUS_0xffd	7.0	6
23057	CUS_0xffd	NaN	1

23058 rows × 3 columns

```
nan_treat(df, 'Customer_ID', 'Num_Credit_Inquiries')
```

```
Initial NaN count in 'Num_Credit_Inquiries': 1965
Lower: -6.0
Q1: 3.0
IQR: 6.0
Q3: 9.0
Upper: 18.0
NaN count in 'Num_Credit_Inquiries' after treatment: 0
```

```
df[['Customer_ID','Num_Credit_Inquiries']].value_counts(dropna = False).reset_index().sort_values('Customer_ID')
```

	Customer_ID	Num_Credit_Inquiries	count
19580	CUS_0x1000	10.0	1
6607	CUS_0x1000	11.0	7
11920	CUS_0x1009	2.0	4
11914	CUS_0x1009	4.0	4
8832	CUS_0x100b	1.0	6
...	...	...	...
5167	CUS_0xff4	5.0	8
5136	CUS_0xff6	2.0	8
18050	CUS_0xffc	8.0	1
6831	CUS_0xffc	13.0	7
0	CUS_0xffd	7.0	8

19581 rows × 3 columns

Outstanding_Debt

```
df['Outstanding_Debt'].isna().sum()
```

```
0
```

Credit_Utilization_Ratio

```
df['Credit_Utilization_Ratio'].isna().sum()
```

```
0
```

Credit_History_Age

```
df['Credit_History_Age'].isna().sum()
```

```
9030
```

```
nan_treat(df, 'Customer_ID', 'Credit_History_Age')
```

```
Initial NaN count in 'Credit_History_Age': 9030
Lower: -93.0
Q1: 144.0
IQR: 158.0
Q3: 302.0
Upper: 539.0
NaN count in 'Credit_History_Age' after treatment: 0
```

Total_EMI_per_month

```
df['Total_EMI_per_month'].isna().sum()
```

```
0
```

Amount_invested_monthly

```
df['Amount_invested_monthly'].isna().sum()
```

```
4479
```

```
nan_treat(df, 'Customer_ID', 'Amount_invested_monthly')
```

```
Initial NaN count in 'Amount_invested_monthly': 4479
Lower: -212.26259564999998
Q1: 74.53400154
IQR: 191.19773146
Q3: 265.731733
Upper: 552.52833019
NaN count in 'Amount_invested_monthly' after treatment: 0
```

Monthly_Balance

```
df['Monthly_Balance'].isna().sum()
```

```
1200
```

```
nan_treat(df, 'Customer_ID', 'Monthly_Balance')
```

```
Initial NaN count in 'Monthly_Balance': 1200
Lower: -30.099756687499905
Q1: 270.092209075
IQR: 200.12797717499996
Q3: 470.22018625
Upper: 770.4121520125
NaN count in 'Monthly_Balance' after treatment: 0
```

Cleaned Data

```
df.head()
```

	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_I
0	CUS_0xd40	1	23	Scientist	19114.12	1824.843333	3	4	3	
1	CUS_0xd40	2	23	Scientist	19114.12	1824.843333	3	4	3	
2	CUS_0xd40	3	23	Scientist	19114.12	1824.843333	3	4	3	
3	CUS_0xd40	4	23	Scientist	19114.12	1824.843333	3	4	3	
4	CUS_0xd40	5	23	Scientist	19114.12	1824.843333	3	4	3	

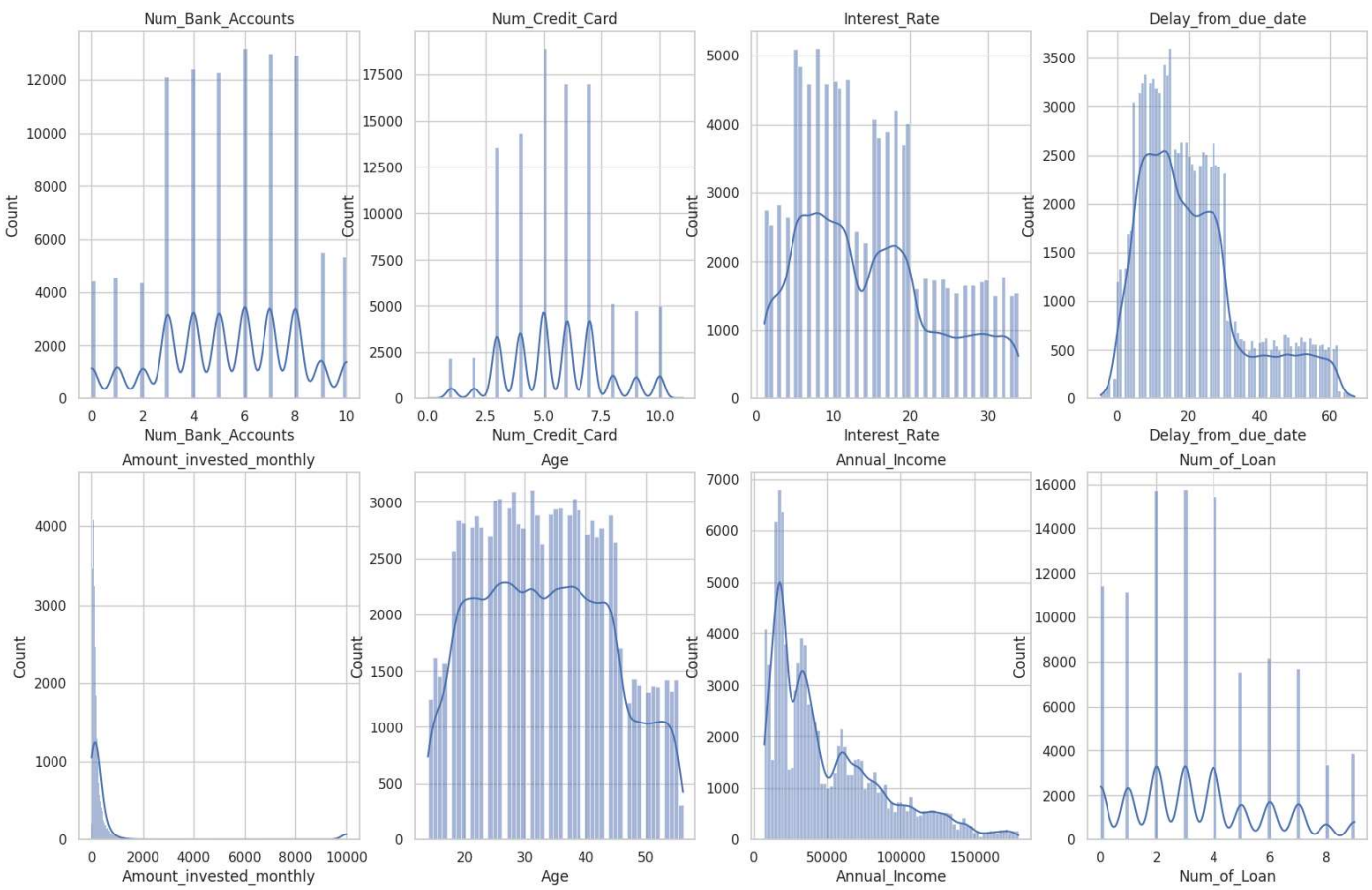
5 rows × 24 columns

```
df.columns
```

```
Index(['Customer_ID', 'Month', 'Age', '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')
```

Key Features

```
column = ['Num_Bank_Accounts','Num_Credit_Card','Interest_Rate','Delay_from_due_date','Amount_invested_monthly','Age','Annual_Income','Num_c
plt.figure(figsize=(19, 12))
for i,col in enumerate(column):
  plt.subplot(2, 4, i + 1)
  sns.histplot(df[col],kde = True)
  plt.title(col)
plt.show()
```



Majority of customers has no.of bank accounts between 3 to 8.

Number of credit cards range from 0 to 11 with most of the customers having credit cards in the range of 3 to 7 with peak at 5.

Interest rate ranges from 1% to 34%.

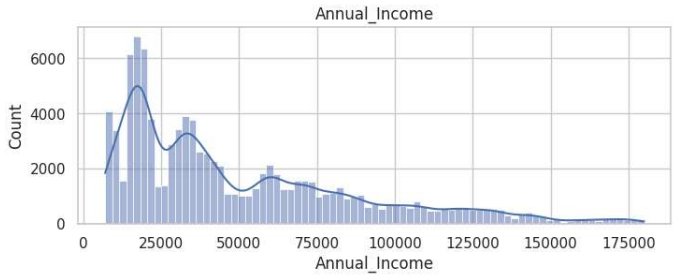
Delay from due date is concentrated between 0 to 30 days.

Very few customers invest greater than 2k amount per month.

Age fo the customer range from 14 to 56. Most customers are of age 18 to 47.

Customers typically take anywhere from 2 to 4 loans, with the maximum number being 9.

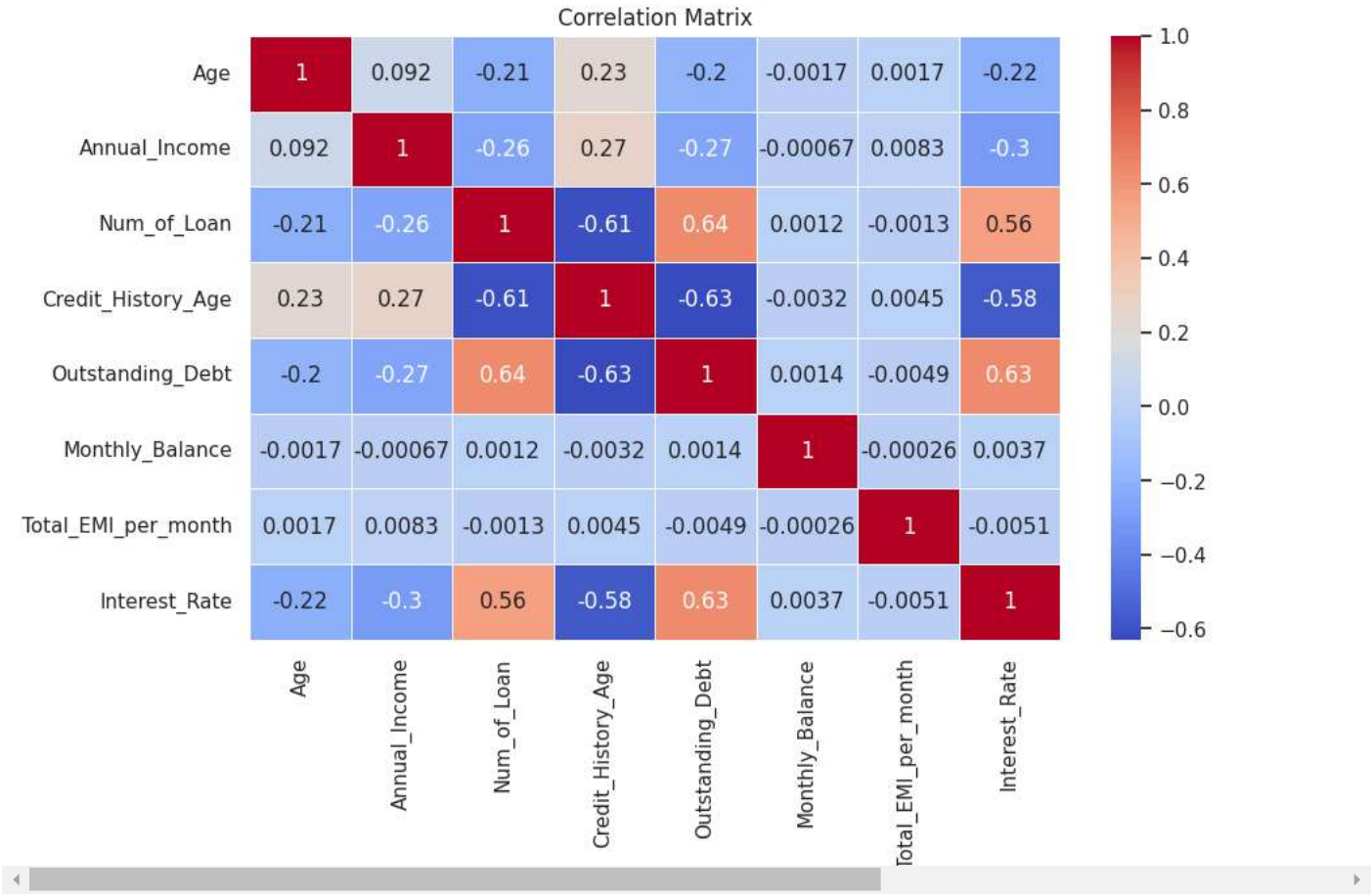
```
column = ['Monthly_Inhand_Salary','Annual_Income']
plt.figure(figsize=(18, 6))
for i,col in enumerate(column):
  plt.subplot(2,2, i + 1)
  sns.histplot(df[col],kde = True)
  plt.title(col)
plt.show()
```



Most customers have a low Annual income and Distribution is right skewed.

Most customers have a low monthly income. Distribution is right skewed.

```
corr_col = ['Age', 'Annual_Income', 'Num_of_Loan', 'Credit_History_Age', 'Outstanding_Debt', 'Monthly_Balance', 'Total_EMI_per_month','Inter

corr_matrix = df[corr_col].corr(method = 'pearson')

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths= 0.5)
plt.title('Correlation Matrix')
plt.show()

# 1 means a perfect positive linear relationship.
# -1 means a perfect negative linear relationship.
# 0 means no linear relationship.
```



Insights:

Strong Positive Correlations:

Credit_History_Age and Annual Income have a positive correlation (0.27). This suggests that as age increases, so does the credit history age.

Num_of_Loan and Outstanding_Debt have a strong positive correlation (0.64). This means that individuals with more loans tend to have higher outstanding debt.

Strong Negative Correlations:

Num_of_Loan and Credit_History_Age have a strong negative correlation (-0.61). This indicates that individuals with more loans tend to have shorter credit histories.

Credit_History_Age and Outstanding_Debt also have a strong negative correlation (-0.63). This suggests that longer credit histories are associated with lower outstanding debt.

Weak Correlations:

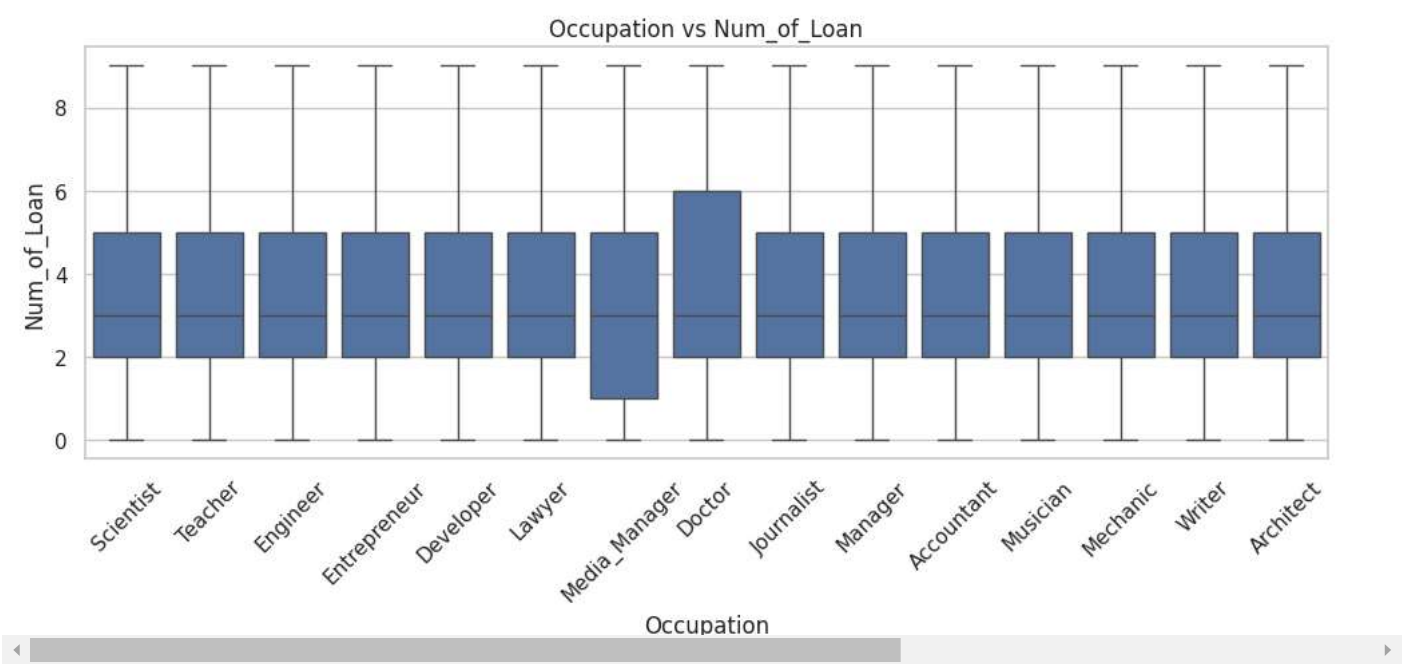
Age and Annual_Income have a weak positive correlation (0.092). This means there's a slight tendency for income to increase with age.

Monthly_Balance and Total_EMI_per_month have a very weak negative correlation (-0.00026). This indicates that there's almost no relationship between these two variables.

--

The strong correlations may indicate potential multicollinearity issues, which should be considered during modeling. Understanding these relationships can inform business decisions, such as targeting marketing campaigns or developing credit risk models.

```
plt.figure(figsize=(12,4))
sns.boxplot(x = 'Occupation', y = 'Num_of_Loan', data = df)
plt.xticks(rotation=45)
plt.title('Occupation vs Num_of_Loan')
plt.show()
```



Insights:

Median Values: Most occupations have a similar median number of loans, around 4. Spread and Variability: The interquartile range (IQR), which is the box itself, is similar across occupations, indicating similar variability in the number of loans.

Outliers and Extremes: There are no visible outliers, and the range (whiskers) is similar, suggesting that the maximum and minimum values of loans are consistent across occupations.

Doctor: This occupation seems to have a slightly higher median and variability compared to others

---

Features

Debit-to-Income Ratio

```
# Measures the percentage of a customer's income that goes towards debt payments.

df['Total_Debt_Payments'] = df['Total_EMI_per_month'] + df['Outstanding_Debt']
df['DTI'] = df['Total_Debt_Payments'] / df['Monthly_Inhand_Salary']
```

> Credit_Utilization_Ratio

> Delay_from_due_date

> Num_Credit_Inquiries

> Loan Diversity Index

```
# Measures the variety of loan types a customer holds.

df['Loan_Diversity'] = df['Type_of_Loan'].apply(lambda x: len(x.split(',')) if isinstance(x, str) else 0)
```

> Credit_History_Age (in months)

> Investment to Income Ratio

```
# Measures the proportion of income invested monthly.

df['ITIR'] = df['Amount_invested_monthly'] / df['Monthly_Inhand_Salary']
```

> Payment Behavior Score

```
# Encodes payment behavior into numerical scores based on categories.

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

Aggregating at customer level

```
df_agg = df.groupby('Customer_ID').agg({
    'Annual_Income': 'mean',
    'Monthly_Inhand_Salary': 'mean',
    'Num_Bank_Accounts': 'mean',
    'Num_Credit_Card': 'mean',
    'Interest_Rate': 'mean',
    'Num_of_Loan': 'mean',
    'Total_Debt_Payments': 'sum',
    'DTI': 'mean',
    'Outstanding_Debt': 'mean',
    'Credit_Utilization_Ratio': 'mean',
   # 'Num_of_Delayed_Payment': 'mode',
    'Credit_History_Age': 'mean',
    'Num_Credit_Inquiries': 'mean',
    'Loan_Diversity': 'mean',
    'ITIR': 'mean',
    'Payment_Behavior_Score': 'mean'
}).reset_index()
```

```
categorical_cols = ['Credit_Mix', 'Occupation', 'Payment_of_Min_Amount','Payment_Behaviour','Num_of_Delayed_Payment']
for col in categorical_cols:
    df_agg[col] = df.groupby('Customer_ID')[col].agg(lambda x: x.mode()[0] if not x.mode().empty else np.nan).values
```

```
df_agg['Payment_Behavior_Score'] = df_agg['Payment_Behavior_Score'].fillna(0).astype(int)
```

```
df_agg.head()
```

	Customer_ID	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Total_Debt_Payment
0	CUS_0x1000	30625.94	2706.161667	6.0	5.0	27.0	2.0	12846.80872
1	CUS_0x1009	52312.68	4250.390000	6.0	5.0	17.0	4.0	2488.37173
2	CUS_0x100b	113781.39	9549.782500	1.0	4.0	1.0	0.0	8241.60000
3	CUS_0x1011	58918.47	5208.872500	3.0	3.0	17.0	3.0	4772.59951
4	CUS_0x1013	98620.98	7962.415000	3.0	3.0	6.0	3.0	60002.20658

5 rows × 21 columns

```
df_agg.shape
```

```
(12500, 21)
```

```
df_agg.isna().sum()
```

	0
Customer_ID	0
Annual_Income	0
Monthly_Inhand_Salary	0
Num_Bank_Accounts	0
Num_Credit_Card	0
Interest_Rate	0
Num_of_Loan	0
Total_Debt_Payments	0
DTI	0
Outstanding_Debt	0
Credit_Utilization_Ratio	0
Credit_History_Age	0
Num_Credit_Inquiries	0
Loan_Diversity	0
ITIR	0
Payment_Behavior_Score	0
Credit_Mix	0
Occupation	0
Payment_of_Min_Amount	0
Payment_Behaviour	0
Num_of_Delayed_Payment	0

dtype: int64

Hypothetical Credit Score Calculation

> Feature Selection

Credit_Utilization_Ratio: A higher utilization ratio can indicate that a customer is close to maxing out their credit, which is a risk factor.

Num_of_Loan: The number of loans a person has taken can reflect their debt load. More loans can indicate a higher risk, especially if they're not being managed well.

Num_Delayed_Payment: This feature reflects the customer's payment history. More delayed payments indicate a higher risk of default, making this feature highly impactful for creditworthiness.

Credit_History_Age: Longer credit histories usually indicate more experience with managing credit.

Num_Credit_Inquiries: This feature represents the number of credit card inquiries made by an individual. Which highlights the activeness of loan seeking.

> Added Features

Debit-to-Income Ratio: Measures the percentage of a customer's income that goes towards debt payments.

Loan_Diversity: Measures the variety of loan types a customer holds.

Investment_to_Income_Ratio: Measures the proportion of income invested monthly.

Payment_Behavior_Score: Encodes payment behavior into numerical scores based on categories.

> Weight Assignment

Different features contribute differently to creditworthiness, and assigning weights based on importance or statistical correlation ensures that we capture this variation in impact.

Credit_Utilization_Ratio (CUR) -> 30% : High utilization suggests potential difficulty in repaying debt.

Debt-to-Income_Ratio (DTI) -> 20% : High debt increases financial strain.

Number_of_Loans (Num_of_Loan) -> 10% : Measures the individual's existing debt load.

Number_of_Credit_Inquiries -> 10% : Fewer inquiries suggest responsible credit-seeking behavior.

Average_Delayed_Payments -> 10% : Major indicator of credit risk, showing payment behavior.

Credit_History_Age -> 10% : Longer history suggests better experience managing credit.

Loan_Diversity -> 5% : Diverse loans indicate a well-rounded credit profile.

Payment_Behavior_Score -> 5% : Positive payment behavior reflects reliability.

Investment_Income_Ratio -> 5% : Higher investment signals better financial management.

Credit_Mix -> 5% : Balanced mix shows diversified and healthy credit usage.

Credit_Score : [Credit_Utilization_Ratio * 0.30 + Debt-to-Income_Ratio * 0.20 + Number_of_Loans * 0.10 + Number_of_Credit_Inquiries * 0.10 + Average_Delayed_Payments * 0.10 + Credit_History_Age * 0.10 + Loan_Diversity * 0.05 + Payment_Behavior_Score * 0.05 + Investment_Income_Ratio * 0.05 + Credit_Mix *0.05]

Scaled Credit Score : 300 + (Credit_Score / max(Credit_Score)) * 550

Credit Score Calculation

Relevant Features:

-> Credit Utilization Ratio (CUR)

-> Debt-to-Income Ratio (DTI)

-> Number of Loans (Num_of_Loan)

-> Average Number of Delayed Payments (Num_of_Delayed_Payment)

-> Credit History Age in Months (Credit_History_Age_Months)

-> Loan Diversity

-> Payment Behavior Score

-> Number of Credit Inquiries

-> Investment Income Ratio

-> Credit Mix

Weight to each feature

Credit Utilization Ratio (CUR) -> 30%

Debt-to-Income Ratio (DTI) -> 20%

Number of Loans (Num_of_Loan) -> 10%

Number of Credit Inquiries -> 10%

Average Delayed Payments -> 10%

Credit History Age -> 10%

Loan Diversity -> 5%

Payment Behavior Score -> 5%

Investment Income Ratio -> 5%

Credit Mix -> 5%

Total -> 100%

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

Credit_Mix	count
Standard	5731
Good	3798
Bad	2971

```
df_agg['Credit_Mix_Score'] = df_agg['Credit_Mix'].map({
    'Good': 1,
    'Standard': 2,
    'Bad': 3
})
```

```
df_agg['Credit_Mix_Score'].value_counts()
```

Credit_Mix_Score	count
2	5731
1	3798
3	2971

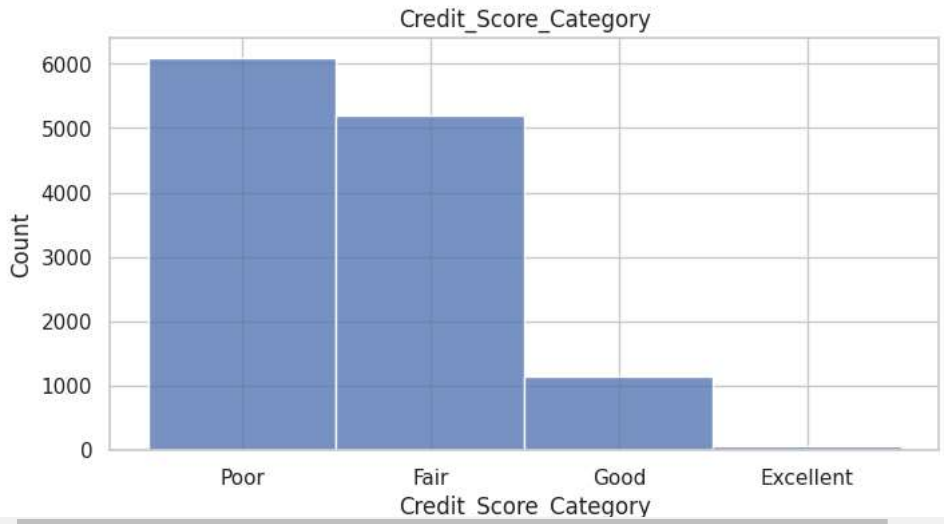
```
df_agg['Num_of_Delayed_Payment-1'] = df_agg['Num_of_Delayed_Payment']
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

features_to_scale = ['Credit_Utilization_Ratio', 'DTI', 'Num_of_Loan', 'Num_Credit_Inquiries', 'Num_of_Delayed_Payment',
                     'Credit_History_Age', 'Loan_Diversity', 'Payment_Behavior_Score', 'ITIR', 'Credit_Mix_Score']

df_agg[features_to_scale] = scaler.fit_transform(df_agg[features_to_scale])
```

```
weights = {
    'Credit_Utilization_Ratio': 0.30,
    'DTI': 0.20,
    'Num_of_Loan': 0.10,
    'Num_Credit_Inquiries': 0.10,
    'Num_of_Delayed_Payment': 0.10,
    'Credit_History_Age': 0.10,
    'Loan_Diversity': 0.05,
    'Payment_Behavior_Score': 0.05,
    'ITIR': 0.05,
    'Credit_Mix_Score': 0.05

}
df_agg['Credit_Score'] = (
    df_agg['Credit_Utilization_Ratio'] * weights['Credit_Utilization_Ratio'] +
    df_agg['DTI'] * weights['DTI'] +
    df_agg['Num_of_Loan'] * weights['Num_of_Loan'] +
    df_agg['Num_Credit_Inquiries'] * weights['Num_Credit_Inquiries'] +
    df_agg['Num_of_Delayed_Payment'] * weights['Num_of_Delayed_Payment'] +
    df_agg['Credit_History_Age'] * weights['Credit_History_Age'] +
    df_agg['Loan_Diversity'] * weights['Loan_Diversity'] +
    df_agg['Payment_Behavior_Score'] * weights['Payment_Behavior_Score'] +
    df_agg['ITIR'] * weights['ITIR'] +
    df_agg['Credit_Mix_Score'] * weights['Credit_Mix_Score']
)


df_agg['Credit_Score_Scaled'] = 300 + (df_agg['Credit_Score']/df_agg['Credit_Score'].max())*550


bins = [0, 300, 580, 670, 740, 850]
labels = ['Very Poor','Poor','Fair','Good','Excellent']

df_agg['Credit_Score_Category'] = pd.cut(df_agg['Credit_Score_Scaled'], bins= bins, labels=labels, right = False)


df_agg[['Customer_ID','Credit_Score_Scaled','Credit_Score_Category']].sample(10)
```

	Customer_ID	Credit_Score_Scaled	Credit_Score_Category
607	CUS_0x19cc	503.107846	Poor
9082	CUS_0x97e1	659.274565	Fair
6799	CUS_0x75d8	596.728877	Fair
10885	CUS_0xb264	445.860142	Poor
6621	CUS_0x7330	551.594142	Poor
7350	CUS_0x7e24	589.982354	Fair
6177	CUS_0x6c90	661.681509	Fair
9645	CUS_0xa022	562.922362	Poor
3209	CUS_0x41f4	615.132823	Fair
1239	CUS_0x2451	753.591802	Excellent

```
plt.figure(figsize=(8, 4))
sns.histplot(df_agg['Credit_Score_Category'])
plt.title('Credit_Score_Category')
plt.show()
```



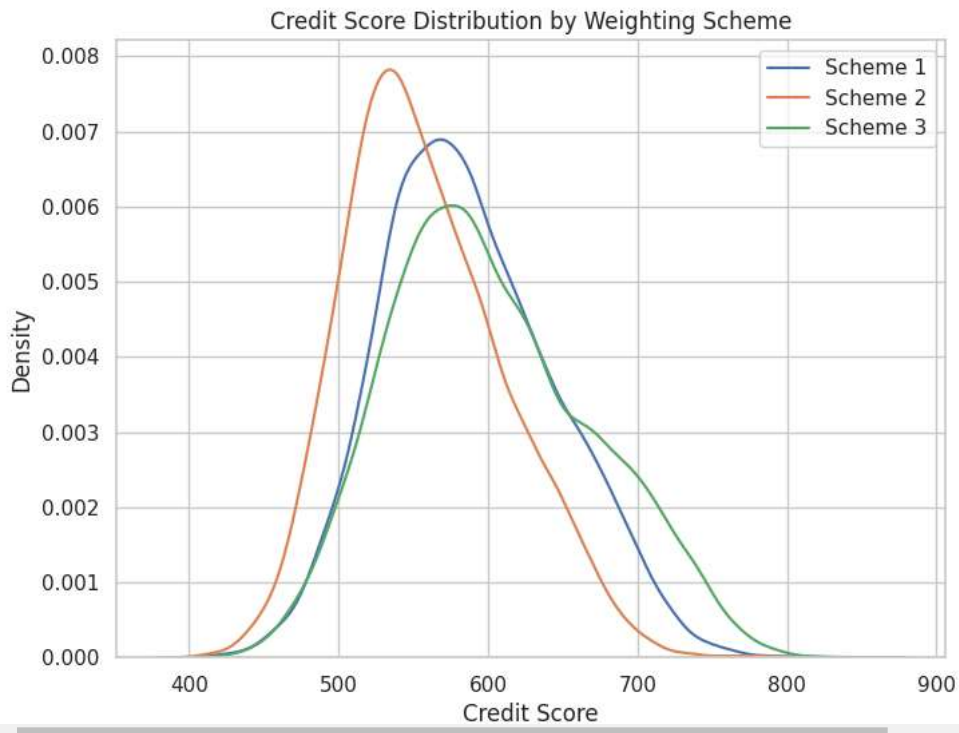
Different Weighting Scheme

```
weight_schemes = [
    {
        'Credit_Utilization_Ratio': 0.30,
        'DTI': 0.20,
        'Num_of_Loan': 0.10,
        'Num_Credit_Inquiries': 0.10,
        'Num_of_Delayed_Payment': 0.10,
        'Credit_History_Age': 0.10,
        'Loan_Diversity': 0.05,
        'Payment_Behavior_Score': 0.05,
        'ITIR': 0.05,
        'Credit_Mix_Score': 0.05
    },
    # Another scheme with different weights
    {
        'Credit_Utilization_Ratio': 0.25,
        'DTI': 0.25,
        'Num_of_Loan': 0.15,
        'Num_Credit_Inquiries': 0.10,
        'Num_of_Delayed_Payment': 0.10,
        'Credit_History_Age': 0.10,
        'Loan_Diversity': 0.05,
        'Payment_Behavior_Score': 0.05,
        'ITIR': 0.05,
        'Credit_Mix_Score': 0.00
    },
    # This scheme places slightly more emphasis on credit history age and delayed payments, while reducing the importance of investment incc
    {
        'Credit_Utilization_Ratio': 0.20,
        'DTI': 0.15,
        'Num_of_Loan': 0.12,
        'Num_Credit_Inquiries': 0.08,
        'Num_of_Delayed_Payment': 0.12,
        'Credit_History_Age': 0.15,
        'Loan_Diversity': 0.08,
        'Payment_Behavior_Score': 0.08,
        'ITIR': 0.02,
        'Credit_Mix_Score': 0.10
    }
]


for idx, weights in enumerate(weight_schemes, start=1):
    df_agg[f'Credit_Score_{idx}'] = (
        df_agg['Credit_Utilization_Ratio'] * weights['Credit_Utilization_Ratio'] +
        df_agg['DTI'] * weights['DTI'] +
        df_agg['Num_of_Loan'] * weights['Num_of_Loan'] +
        df_agg['Num_Credit_Inquiries'] * weights['Num_Credit_Inquiries'] +
        df_agg['Num_of_Delayed_Payment'] * weights['Num_of_Delayed_Payment'] +
        df_agg['Credit_History_Age'] * weights['Credit_History_Age'] +
```

```
            df_agg['Loan_Diversity'] * weights['Loan_Diversity'] +
            df_agg['Payment_Behavior_Score'] * weights['Payment_Behavior_Score'] +
            df_agg['ITIR'] * weights['ITIR'] +
            df_agg['Credit_Mix_Score'] * weights['Credit_Mix_Score']
        )

        df_agg[f'Credit_Score_Scaled_{idx}'] = 300 + (df_agg[f'Credit_Score_{idx}'] / df_agg[f'Credit_Score_{idx}'].max()) * 550

        df_agg[f'Credit_Rating_{idx}'] = pd.cut(df_agg[f'Credit_Score_Scaled_{idx}'], bins=bins, labels=labels, right=False)

        print(f"Scheme {idx} Credit Scores and Ratings:")
        print(df_agg[['Customer_ID', f'Credit_Score_Scaled_{idx}', f'Credit_Rating_{idx}']].head(), "\n")
```

```
Scheme 1 Credit Scores and Ratings:
  Customer_ID  Credit_Score_Scaled_1 Credit_Rating_1
0  CUS_0x1000             634.307229            Fair
1  CUS_0x1009             573.223061            Poor
2  CUS_0x100b             514.542428            Poor
3  CUS_0x1011             516.842511            Poor
4  CUS_0x1013             518.636842            Poor

Scheme 2 Credit Scores and Ratings:
  Customer_ID  Credit_Score_Scaled_2 Credit_Rating_2
0  CUS_0x1000             573.640285            Poor
1  CUS_0x1009             549.195478            Poor
2  CUS_0x100b             484.476645            Poor
3  CUS_0x1011             496.115444            Poor
4  CUS_0x1013             507.609072            Poor

Scheme 3 Credit Scores and Ratings:
  Customer_ID  Credit_Score_Scaled_3 Credit_Rating_3
0  CUS_0x1000             646.283672            Fair
1  CUS_0x1009             615.959957            Fair
2  CUS_0x100b             491.238842            Poor
3  CUS_0x1011             551.060957            Poor
4  CUS_0x1013             513.908757            Poor
```

```
# Example: Compare distributions
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
for idx in range(1, len(weight_schemes)+1):
    sns.kdeplot(df_agg[f'Credit_Score_Scaled_{idx}'], label=f'Scheme {idx}')
plt.title('Credit Score Distribution by Weighting Scheme')
plt.xlabel('Credit Score')
plt.ylabel('Density')
plt.legend()
plt.show()
```



Scheme 1 (blue line) has a slightly broader distribution compared to the others, suggesting it allows for a wider range of credit scores. It peaks around the 600-650 range but has more outliers in both the higher and lower ends of the spectrum.

Scheme 2 (orange line) appears to be more tightly concentrated, peaking higher around 600, indicating a more consistent and perhaps stricter scoring mechanism, with fewer outliers.

Scheme 3 (green line) has the most spread-out distribution, with a lower and broader peak around 575-625, indicating that it might allow for greater variance in credit scores, covering both lower and higher extremes more evenly.

Scheme 1 is selected as it provides a more balanced approach, capturing a moderate spread of credit scores, with a slight emphasis on the 600-650 range.

```
df_agg['Final_Credit_Score'] = df_agg['Credit_Score_Scaled_1']
df_agg['Final_Credit_Rating'] = df_agg['Credit_Rating_1']
```

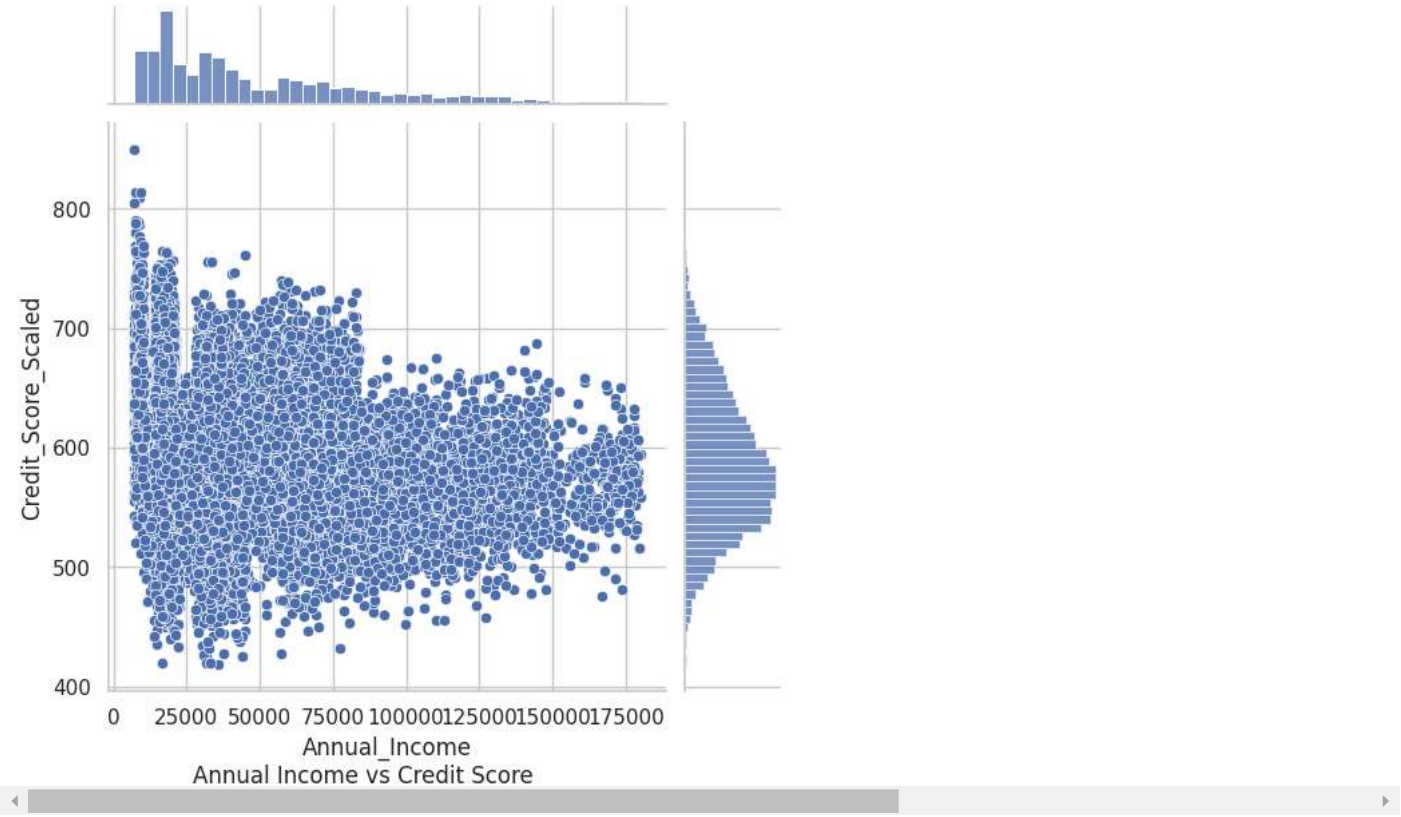
```
print(df_agg[['Customer_ID', 'Final_Credit_Score', 'Final_Credit_Rating']].head())
```

```
   Customer_ID  Final_Credit_Score Final_Credit_Rating
0  CUS_0x1000          634.307229                Fair
1  CUS_0x1009          573.223061                Poor
2  CUS_0x100b          514.542428                Poor
3  CUS_0x1011          516.842511                Poor
4  CUS_0x1013          518.636842                Poor
```

```
df_agg.to_csv('credit_scores.csv', index=False)
```

```
plt.figure(figsize=(14,10))
g= sns.jointplot(x='Annual_Income', y='Credit_Score_Scaled', data=df_agg, kind='scatter')
plt.figtext(0.45, 0.001, 'Annual Income vs Credit Score', ha='center', fontsize=12)
plt.show()
```

```
<Figure size 1400x1000 with 0 Axes>
```

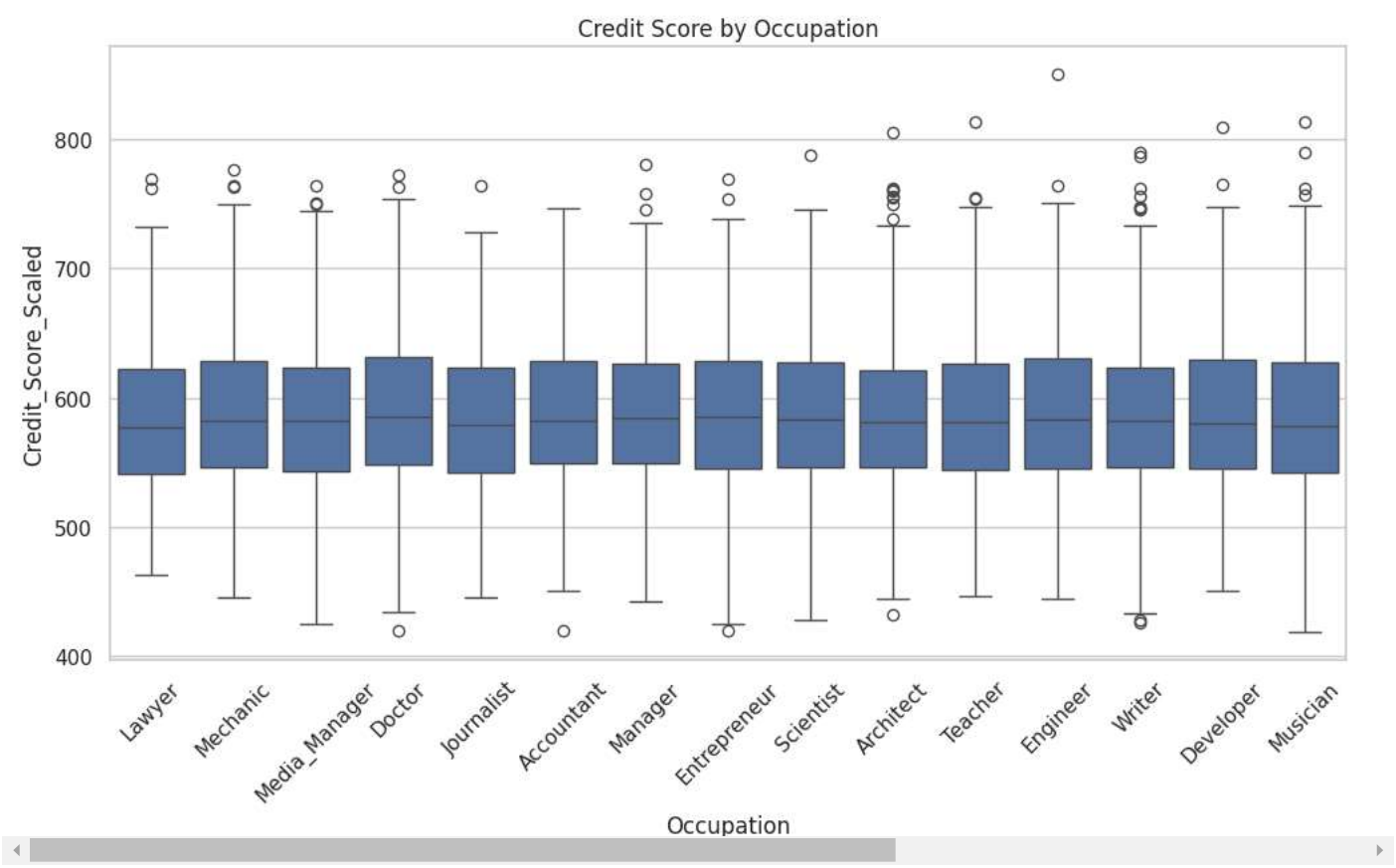


Insights

There doesn't seem to be a strong correlation between annual income and credit score based on this scatter plot. Individuals with both low and high incomes can have varying credit scores, suggesting that income alone may not be a dominant factor in determining credit scores.

However, for incomes above 50,000, the distribution of credit scores seems to stabilize, with fewer extreme low scores (below 500) and a more even spread around the middle.

```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Occupation', y='Credit_Score_Scaled', data=df_agg)
plt.title('Credit Score by Occupation')
plt.xticks(rotation=45)
plt.show()
```

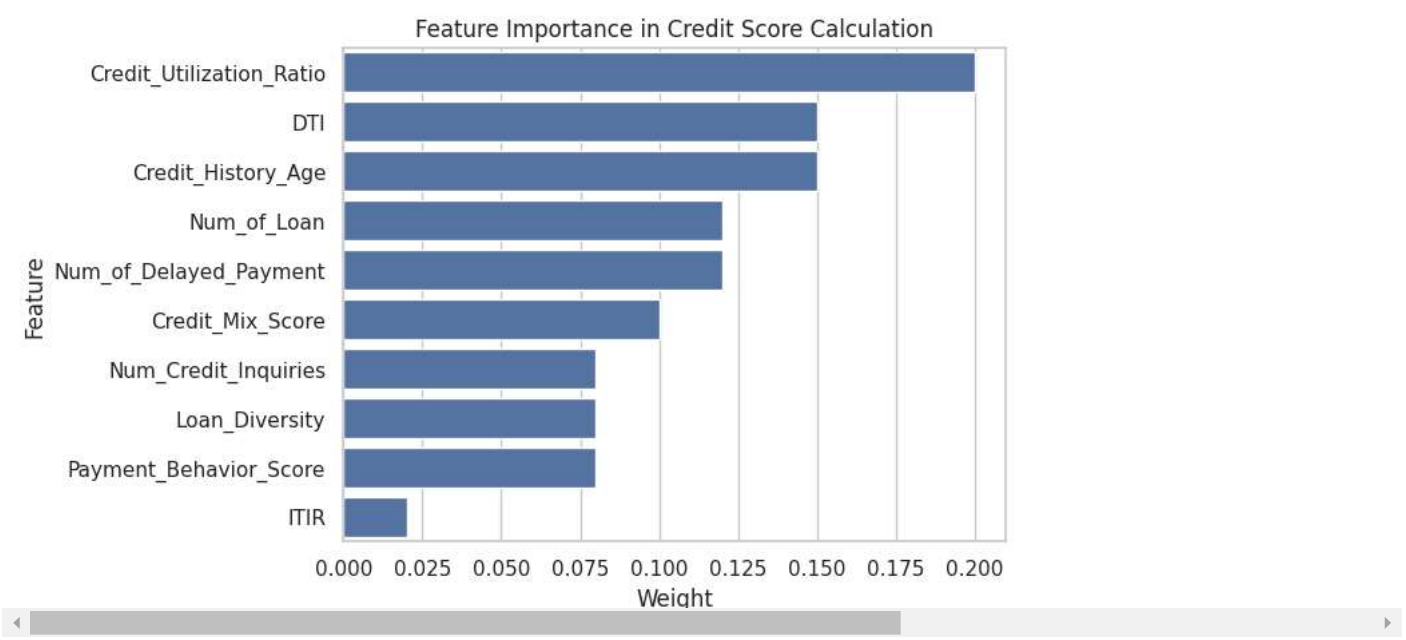


Insights:

The plot reveals significant differences in credit score distributions across various occupations. For instance, doctors tend to have higher median credit scores and a more compact distribution, while musicians exhibit a wider range of scores. This suggests that certain professions may be more conducive to financial stability or have different financial priorities.

The presence of outliers in several occupations, particularly high-scoring outliers in fields like science and architecture, indicates that individual financial performance can deviate significantly from the median.

```
# Feature Importance Visualization
feature_importance = pd.Series(weights).sort_values(ascending=False)
sns.barplot(x=feature_importance.values, y=feature_importance.index)
plt.title('Feature Importance in Credit Score Calculation')
plt.xlabel('Weight')
plt.ylabel('Feature')
plt.show()
```



Insights:

Most important factors:

The longest bars represent the most significant factors in credit score calculation. In this graph, "DTI" (Debt-to-Income Ratio) and "Credit_History_Age" hold the highest weights, signifying their strong influence on the credit score.

Moderately important factors:

Features like "Num_of_Loan" (Number of Loans), "Num_of_Delayed_Payment" (Number of Delayed Payments), and "Credit_Mix_Score" have moderate weights, indicating a substantial but lesser impact on the overall score.

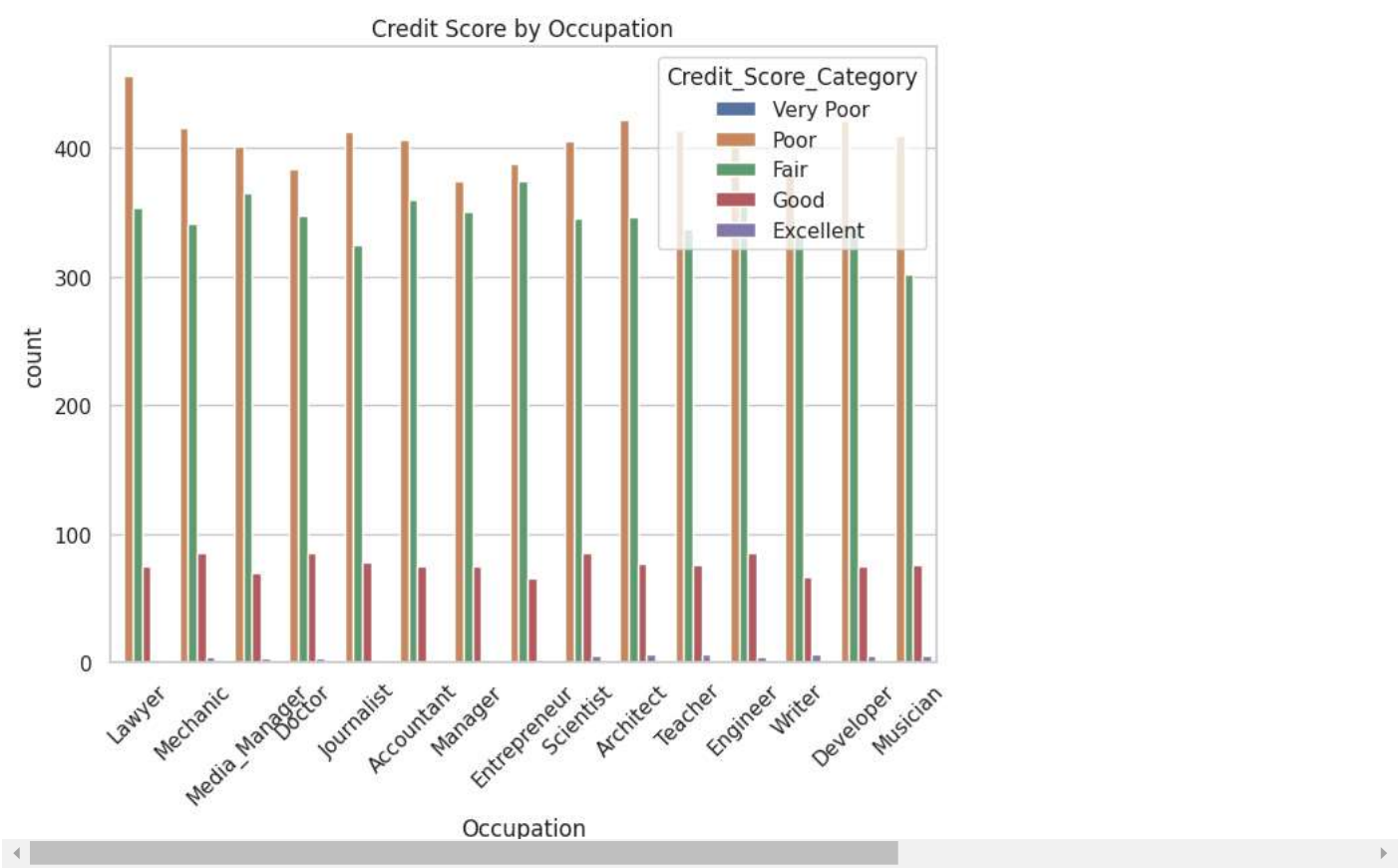
Least important factors:

"ITIR" and "Payment_Behavior_Score" have the shortest bars, implying that they contribute the least to the overall credit score calculation.
Importance:

```
df_agg['Credit_Score_Category'].dtypes
```

```
CategoricalDtype(categories=['Very Poor', 'Poor', 'Fair', 'Good', 'Excellent'], ordered=True, categories_dtype=object)
```

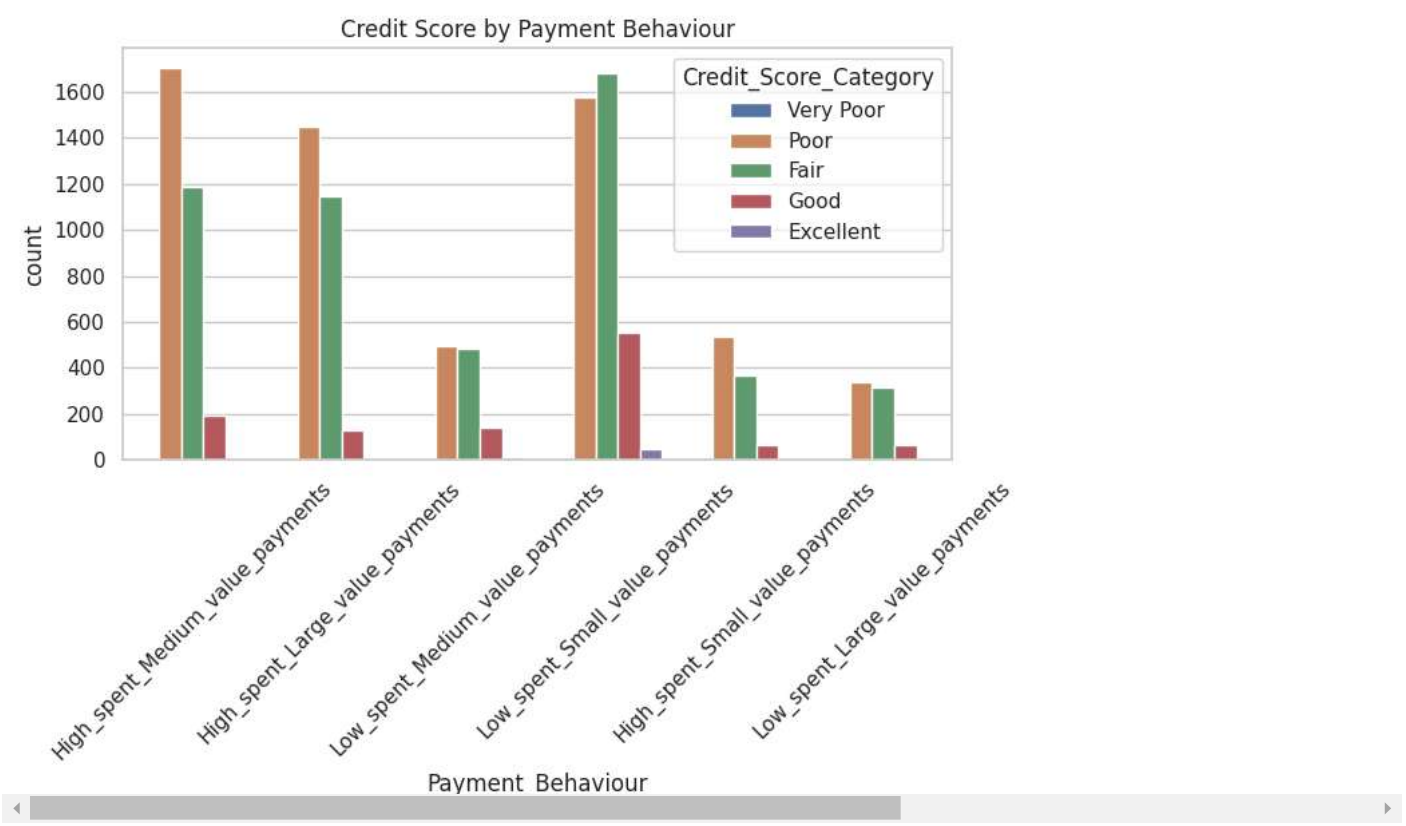
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
plt.figure(figsize=(8, 6))
sns.countplot(x='Occupation', hue='Credit_Score_Category', data=df_agg)
plt.title('Credit Score by Occupation')
plt.xticks(rotation=45)
plt.show()
```




```
df_agg['Payment_Behavior_Score'].unique()
```

```
array([0.5 , 0.25, 0.75, 0.  , 1.  ])
```

```
plt.figure(figsize=(8, 4))
sns.countplot(x='Payment_Behaviour', hue='Credit_Score_Category', data=df_agg)
plt.title('Credit Score by Payment Behaviour')
plt.xticks(rotation=45)
plt.show()
```



Insights:

Most of the people lies in High_spent_Medium_value_payments,High_spent_Large_value_payments,Low_spent_small_value_payments segments

> Hight spending category:

For high spent medium value payments and high spent, large value payments, most individuals have Poor or Fair credit scores. There is a small number of individuals with Good credit scores.

> Low spending category:

For low spent medium value payments, low spent large value payments the pattern is somewhat similar. However, in low spent small value payments stands out, the number of individuals with a Fair score is significantly larger, and we see a rise in Good credit scores higher than rest of the other category. We can only see excellent score in only low spending categories

> General Observation:

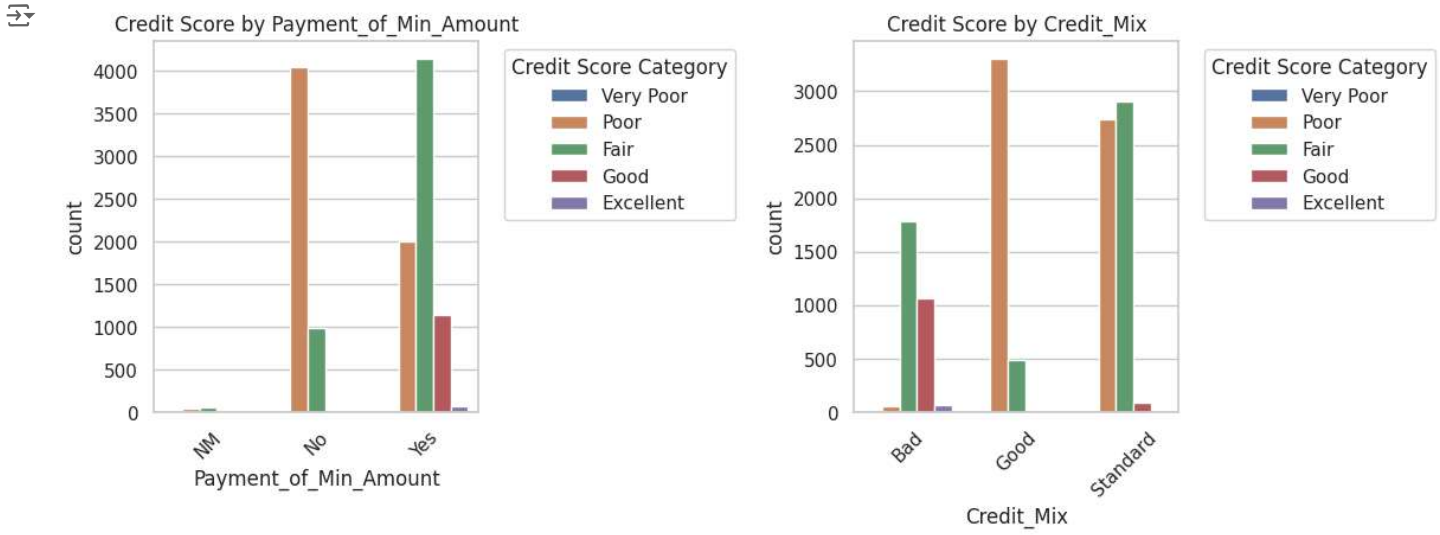
Overall, it appears that Fair and Poor credit scores dominate across most payment behaviors. Good and Excellent scores are relatively rare, while Very Poor scores are minimal in all categories.

```
df_agg['Payment_Behaviour'].unique()
```

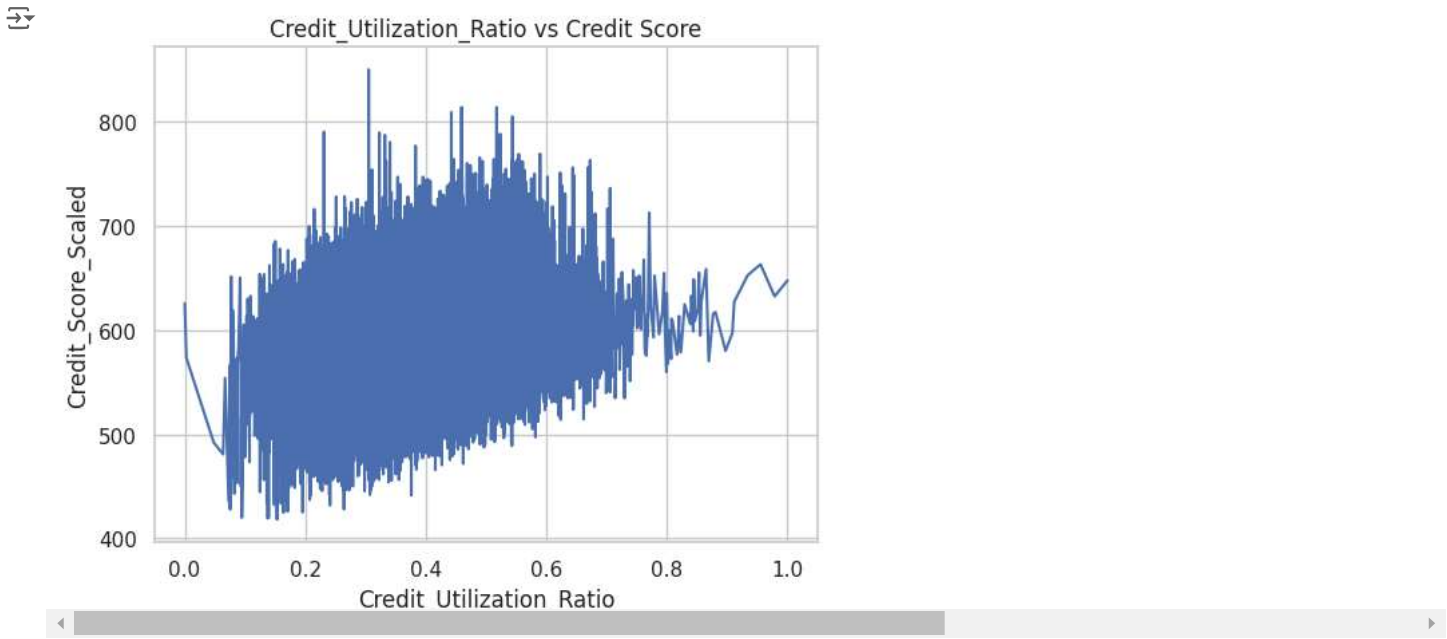
```
array(['High_spent_Medium_value_payments',
       'High_spent_Large_value_payments',
       'Low_spent_Medium_value_payments',
       'Low_spent_Small_value_payments',
       'High_spent_Small_value_payments',
       'Low_spent_Large_value_payments'], dtype=object)
```

```
cat_col = ['Payment_of_Min_Amount', 'Credit_Mix']

for col in cat_col:
    df_agg[col] = df_agg[col].astype('category')

hue_order = df_agg['Credit_Score_Category'].unique()

sns.set(style="whitegrid")

plt.figure(figsize=(12, 8))
```

```
for i, col in enumerate(cat_col):
    plt.subplot(2, 2, i + 1)
    sns.countplot(x=col, hue='Credit_Score_Category', data=df_agg)
    plt.title(f'Credit Score by {col}')
    plt.xticks(rotation=45)
    plt.legend(title='Credit Score Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```




```
sns.lineplot(x='Credit_Utilization_Ratio', y='Credit_Score_Scaled', data=df_agg)
plt.title('Credit_Utilization_Ratio vs Credit Score')
plt.show()
```

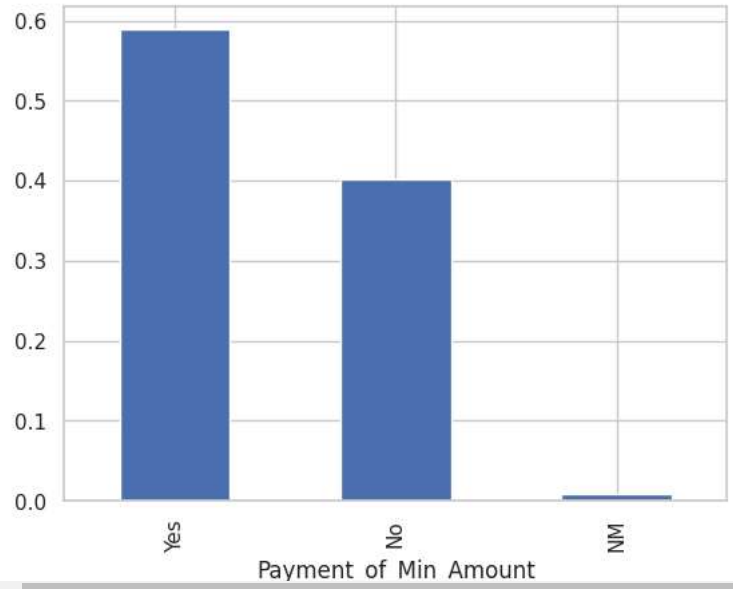


Insights:

There is a negative correlation between credit_utilization_ratio and credit_Score. Which means that as the credit_utilization_ratio increases, the credit_score tends to decrease

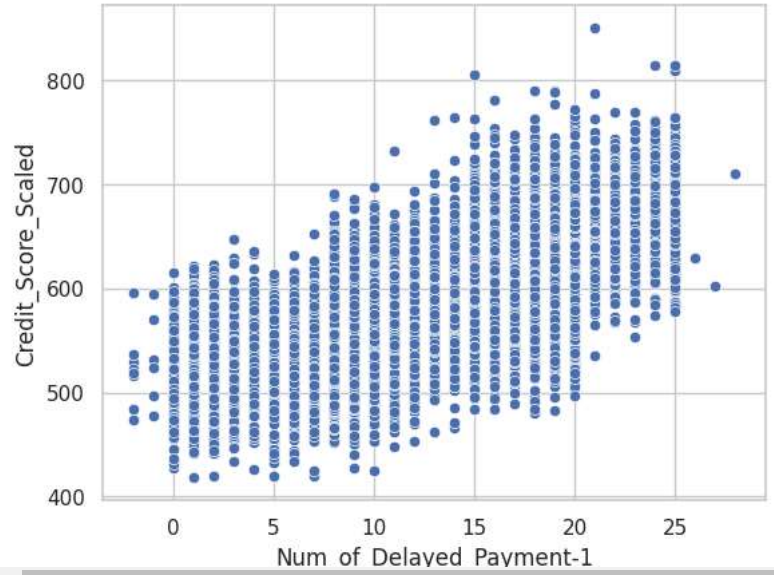
```
df_agg['Payment_of_Min_Amount'].value_counts(normalize=True).plot(kind='bar')
```

```
<Axes: xlabel='Payment_of_Min_Amount'>
```




```
sns.scatterplot(x='Num_of_Delayed_Payment-1', y='Credit_Score_Scaled', data=df_agg)
```

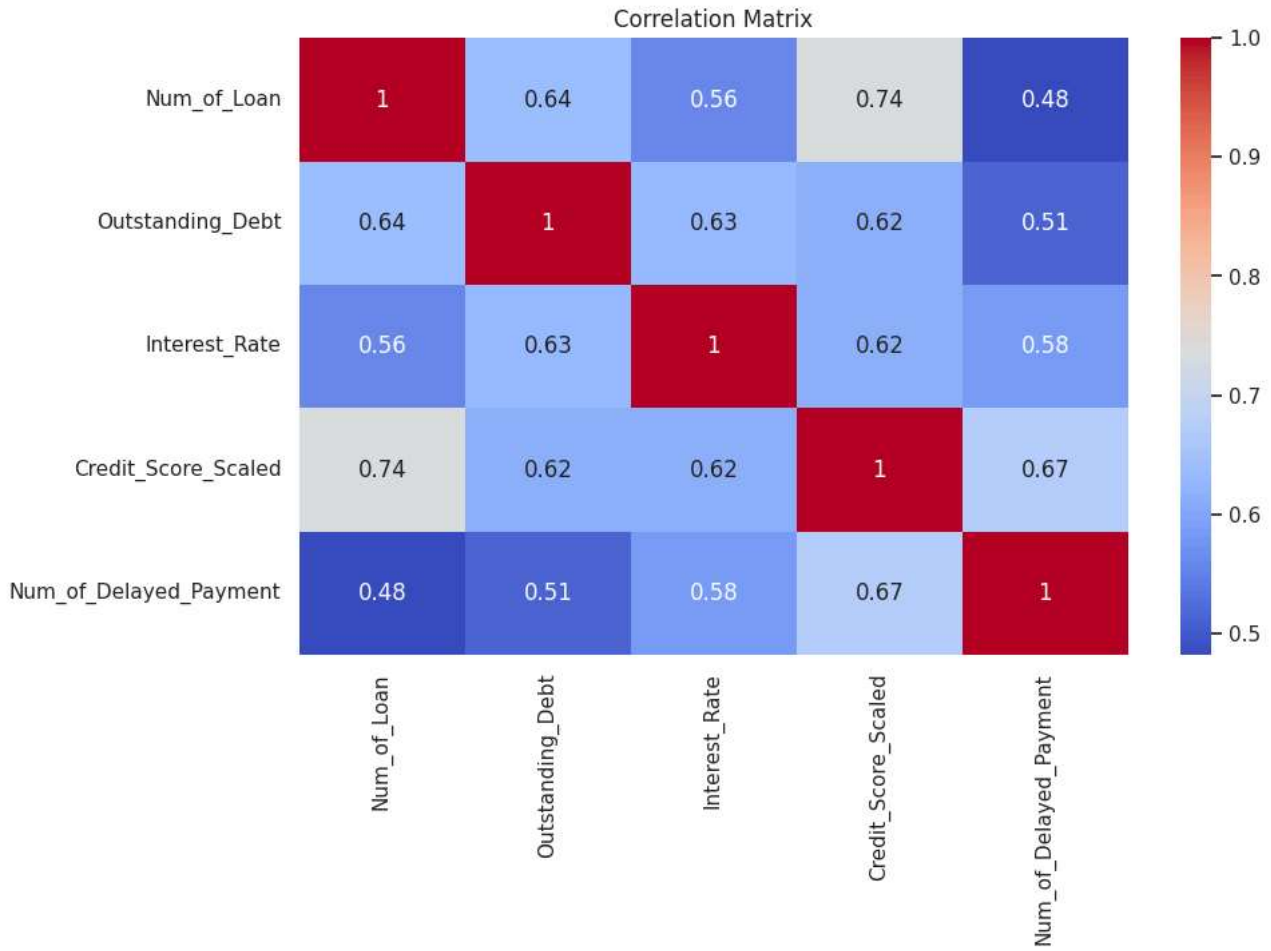
```
<Axes: xlabel='Num_of_Delayed_Payment-1', ylabel='Credit_Score_Scaled'>
```



Insights:

There is a clear positive correlation between the number of delayed payments and the credit score. As the number of delayed payments increases (from 0 to 1), the credit score also increases.

```
col = ['Num_of_Loan', 'Outstanding_Debt','Interest_Rate','Credit_Score_Scaled','Num_of_Delayed_Payment']
corr_matrix = df_agg[col].corr(method = 'pearson')

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Insights:

pThe strongest positive correlation is between "Num_of_Loan" and "Outstanding_Debt" (0.64), suggesting that as the number of loans increases, the outstanding debt also tends to increase.

There's a moderate positive correlation between "Interest_Rate" and "Outstanding_Debt" (0.63), indicating that higher interest rates are associated with higher outstanding debt.

There are other moderate positive correlations between variables, such as "Num_of_Loan" and "Credit_Score_Scaled" (0.74), suggesting that individuals with more loans tend to have higher credit scores.

Credit Score : 3 and 6 Months

Calculating cedit score over different time frames helps capture the most relevant, up-to-date financial information, enhances risk prediction accuracy, and allows for better monitoring of credit behavior.

It ensures that the creditworthiness assessment is dynamic and reflects both short-term financial changes and long-term patterns, leading to more informed decisions for lenders and financial institutions.

```
df_m = df.copy()
```

```
df_m.columns
```

```
Index(['Customer_ID', 'Month', 'Age', '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',
       'Total_Debt_Payments', 'DTI', 'Loan_Diversity', 'ITIR',
```

```
              'Payment_Behavior_Score'],
             dtype='object')


df_m['Credit_Mix_Score'] = df_m['Credit_Mix'].map({
    'Good': 1,
    'Standard': 2,
    'Bad': 3
})

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

features_to_scale = ['Credit_Utilization_Ratio', 'DTI', 'Num_of_Loan', 'Num_Credit_Inquiries', 'Num_of_Delayed_Payment',
                     'Credit_History_Age', 'Loan_Diversity', 'Payment_Behavior_Score', 'ITIR', 'Credit_Mix_Score']

df_m[features_to_scale] = scaler.fit_transform(df_m[features_to_scale])

weights = {
    'Credit_Utilization_Ratio': 0.30,
    'DTI': 0.20,
    'Num_of_Loan': 0.10,
    'Num_Credit_Inquiries': 0.10,
    'Num_of_Delayed_Payment': 0.10,
    'Credit_History_Age': 0.10,
    'Loan_Diversity': 0.05,
    'Payment_Behavior_Score': 0.05,
    'ITIR': 0.05,
    'Credit_Mix_Score': 0.05

}
df_m['Credit_Score'] = (
    df_m['Credit_Utilization_Ratio'] * weights['Credit_Utilization_Ratio'] +
    df_m['DTI'] * weights['DTI'] +
    df_m['Num_of_Loan'] * weights['Num_of_Loan'] +
    df_m['Num_Credit_Inquiries'] * weights['Num_Credit_Inquiries'] +
    df_m['Num_of_Delayed_Payment'] * weights['Num_of_Delayed_Payment'] +
    df_m['Credit_History_Age'] * weights['Credit_History_Age'] +
    df_m['Loan_Diversity'] * weights['Loan_Diversity'] +
    df_m['Payment_Behavior_Score'] * weights['Payment_Behavior_Score'] +
    df_m['ITIR'] * weights['ITIR'] +
    df_m['Credit_Mix_Score'] * weights['Credit_Mix_Score']
)

df_m['Credit_Score_Scaled'] = 300 + (df_m['Credit_Score']/df_m['Credit_Score'].max())*550

bins = [0, 300, 580, 670, 740, 850]
labels = ['Very Poor','Poor','Fair','Good','Excellent']

df_m['Credit_Score_Category'] = pd.cut(df_m['Credit_Score_Scaled'], bins= bins, labels=labels, right = False)


df_m['Credit_Score_3_Months'] = df_m.groupby('Customer_ID')['Credit_Score_Scaled'].transform(lambda x: x.rolling(3).mean())


df_m['Credit_Score_6_Months'] = df_m.groupby('Customer_ID')['Credit_Score_Scaled'].transform(lambda x: x.rolling(6).mean())


df_m[['Customer_ID', 'Credit_Score_Scaled', 'Credit_Score_3_Months','Credit_Score_6_Months']].head(30)
```

	Customer_ID	Credit_Score_Scaled	Credit_Score_3_Months	Credit_Score_6_Months
0	CUS_0xd40	492.983356	NaN	NaN
1	CUS_0xd40	562.846401	NaN	NaN
2	CUS_0xd40	537.671598	531.167118	NaN
3	CUS_0xd40	543.830336	548.116112	NaN
4	CUS_0xd40	478.224873	519.908936	NaN
5	CUS_0xd40	497.303124	506.452778	518.809948
6	CUS_0xd40	486.925765	487.484587	517.800350
7	CUS_0xd40	475.932726	486.720538	503.314737
8	CUS_0x21b1	451.616363	NaN	NaN
9	CUS_0x21b1	544.314357	NaN	NaN
10	CUS_0x21b1	498.638802	498.189841	NaN
11	CUS_0x21b1	569.977335	537.643498	NaN
12	CUS_0x21b1	525.234458	531.283532	NaN
13	CUS_0x21b1	502.959053	532.723616	515.456728
14	CUS_0x21b1	488.495645	505.563052	521.603275
15	CUS_0x21b1	517.858877	503.104525	517.194029
16	CUS_0x2dbc	481.165354	NaN	NaN
17	CUS_0x2dbc	576.441170	NaN	NaN
18	CUS_0x2dbc	463.972628	507.193051	NaN
19	CUS_0x2dbc	587.530717	542.648172	NaN

Customers with recent scores lower than their overall scores indicate a decline in financial stability or increasing risk.

Conversely, customers whose recent scores are higher than their overall scores may be improving their financial habits and becoming lower risk.

Customers who show a downward trend in recent scores may warrant closer scrutiny, as this could indicate rising financial stress or potential risk for lenders.