In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
```

In [3]:
```python
df = pd.read_csv("Credit_score.csv", low_memory = False)
```

In [4]:
```python
data = df.copy()
```

Basic Analyis :-

In [5]:
```python
data.head()
```

Out[5]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Num_Credit
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

In [6]:
```python
data.shape
```

Out[6]: (100000, 27)

In [7]:
```python
data.size
```

Out[7]: 2700000

In [8]:
```python
# Checking count of null values in each columns.
data.isna().sum()
```

Out[8]:
```
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
```

In [9]:
```
data.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
```

```
From this data info we can we see many columns does not correct data types & many null values also. So in
the further analysis we'll try to change the data types of column for better analysis.
```

In [10]:
```
data.head()
```

Out[10]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Num_Credit
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

In [11]:
```
# Code to fill the null values present in Name column:

#sort the dataframe by customer_id & name
data.sort_values(['Customer_ID','Name'], inplace = True)

#forward fill the customer name within each customer id
data['Name'] = data.groupby('Customer_ID')['Name'].fillna(method = 'ffill')

#backward fill the customer name within each customer id
data['Name'] = data.groupby('Customer_ID')['Name'].fillna(method = 'bfill')

#if there are still null values then replacing them with unknown name
data['Customer_ID'].fillna('Unknown Name', inplace = True)
```

In [12]:
```
# Checking is there any null values in Name column left:
data['Name'].isnull().sum()
```

Out[12]: 0

In [13]:
```
# Handeling the absurd values in Age Column:

data['Age'] = data['Age'].str.replace('_', '')
data['Age'] = data['Age'].str.replace('_', '')

data['Age'] = data['Age'].astype('int')

# Replacing age with NA if age is not in the range of 0 to 100
data['Age'] = data['Age'].where((data['Age'] > 0) & (data['Age'] <= 100), pd.NA)

# Filling NA with the mode age of that particular Customer.
data['Age'] = data.groupby('Customer_ID')['Age'].transform(lambda x: x.fillna(x.mode().iloc[0]))

# Replacing min & max age of cutomer with mode age, indirectly handeling outliers.
data['Age'] = data.groupby('Customer_ID')['Age'].transform(lambda x: x.replace(x.max(),x.mode().iloc[0]))
data['Age'] = data.groupby('Customer_ID')['Age'].transform(lambda x: x.replace(x.min(),x.mode().iloc[0]))

# Changing data type to int again
data['Age']=data['Age'].astype(int)
```

In [14]:
```
# Checking if the data of Age column got corrected.
data['Age'].unique()
```

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

In [15]:
```
data['Age'].value_counts()
```

Out[15]:
```
31    3112
28    3096
38    3032
26    3032
25    3016
27    2952
36    2952
35    2944
39    2936
34    2896
32    2888
44    2888
37    2888
22    2880
19    2840
41    2840
20    2816
29    2808
21    2776
```

In [16]:
```
data['Month'].value_counts()
```

Out[16]:
```
January     12500
March       12500
April       12500
May         12500
June        12500
July        12500
August      12500
February    12500
Name: Month, dtype: int64
```

From above value count for 'Months', we can see only 8 months are present in our data.

In [17]:
```
data['SSN'].value_counts().sort_values(ascending = False)
```

Out[17]:
```
#F%$D@*&8      5572
425-36-5909       8
754-40-2219       8
511-20-0282       8
228-53-9703       8
               ...
604-62-6133       4
286-44-9634       4
838-33-4811       4
331-28-1921       4
753-72-2651       4
Name: SSN, Length: 12501, dtype: int64
```

In [18]:
```python
# Code to correct the absurd value in SSN column:

#Function to return valid SSN
def replace_wrong_SSN(group):
    correct_ssn = group.loc[group['SSN'] != '#F%$D@*&8', 'SSN'].iloc[0]
    group_ssn = group.loc[group['SSN'] == '#F%$D@*&8', 'SSN'] = correct_ssn
    return group

data = data.groupby('Customer_ID', group_keys = False).apply(replace_wrong_SSN).reset_index(drop = True)
```

In [19]:
```python
data['SSN'].value_counts().sort_values(ascending =True)
```

Out[19]:
```
913-74-1218    8
066-65-6008    8
230-22-9583    8
238-62-0395    8
793-05-8223    8
              ..
075-63-9119    8
138-97-1797    8
731-85-6329    8
115-56-7254    8
832-88-8320    8
Name: SSN, Length: 12500, dtype: int64
```

In [20]:
```python
data['Occupation'].value_counts()
```

Out[20]:
```
_______          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
Name: Occupation, dtype: int64
```

In [21]:
```python
# Code to handle "______" in Occupation column:

def replace_underscore(group):
    mode_occupation = group['Occupation'].mode().iloc[0]
    if mode_occupation != "_______":
        group['Occupation'] = group['Occupation'].replace("_______", mode_occupation)

    else:
        underscore_modes = group['Occupation'][group['Occupation'] != "_______"].mode().iloc[0]
        group['Occupation'] = group['Occupation'].replace("_______", underscore_modes)
    return group


data = data.groupby("Customer_ID", group_keys = False).apply(replace_underscore).reset_index(drop = True)
```

In [22]:
```python
data['Occupation'].value_counts()
```

Out[22]:
```
Lawyer           7096
Engineer         6864
Architect        6824
Mechanic         6776
Accountant       6744
Scientist        6744
Media_Manager    6720
Developer        6720
Teacher          6672
Entrepreneur     6648
Doctor           6568
Journalist       6536
Manager          6432
Musician         6352
Writer           6304
Name: Occupation, dtype: int64
```

In [23]:
```python
# Cleaning in Annual Income Column:

data['Annual_Income'] = data['Annual_Income'].str.replace('_','')
data['Annual_Income'] = data['Annual_Income'].astype(float)
```

In [24]:
```
nan_count_by_customer = data.groupby('Customer_ID')['Monthly_Inhand_Salary'].apply(lambda x: x.isna().sum()
nan_count_by_customer.value_counts()
```

Out[24]:
```
1    4862
0    3401
2    2904
3    1048
4     240
5      42
6       3
Name: Monthly_Inhand_Salary, dtype: int64
```

In [25]:
```
data.sort_values(by=['Customer_ID', 'Month'], inplace=True)
data['Monthly_Inhand_Salary'] = data.groupby('Customer_ID')['Monthly_Inhand_Salary'].fillna(method='ffill')
data['Monthly_Inhand_Salary'] = data.groupby('Customer_ID')['Monthly_Inhand_Salary'].fillna(method='bfill')
```

In [26]:
```
data['Monthly_Inhand_Salary'].isna().sum()
```

Out[26]:
```
0
```

In [27]:
```
# Checking for any data anomality in Num_Bank_Accounts Column.
data['Num_Bank_Accounts'].value_counts().sort_values(ascending = False).head(60)
```

Out[27]:
```
 6       13001
 7       12823
 8       12765
 4       12186
 5       12118
 3       11950
 9        5443
 10       5247
 1        4490
 0        4328
 2        4304
-1          21
 11          9
 803         7
 1668        5
 105         5
 791         5
 1257        4
 312         4
```

As we can see their are many annomality in this column, so in further steps we'll try to correct it.

In [28]:
```
# Replacing the outliers value with the mode or 0 whichever is high, for each group.

data['Num_Bank_Accounts'] = data.groupby('Customer_ID')['Num_Bank_Accounts'].transform(lambda x: max(0, x.m
```

In [29]:
```
# Checking if the data cleaning is done properly or not.

data['Num_Bank_Accounts'].value_counts().sort_values(ascending = False).head(60)
```

Out[29]:
```
6     13184
7     12976
8     12936
4     12392
5     12272
3     12096
9      5512
10     5328
1      4552
0      4400
2      4352
Name: Num_Bank_Accounts, dtype: int64
```

In [30]:
```
# Checking for anaomality in 'Num_Credit_card' column.

data['Num_Credit_Card'].value_counts().sort_values(ascending = True).head(50)
```

Out[30]:
```
1108    1
592     1
1198    1
1376    1
475     1
1103    1
1219    1
601     1
1435    1
1035    1
1331    1
168     1
518     1
1494    1
432     1
1028    1
499     1
591     1
1199    1
445     1
313     1
606     1
1332    1
1270    1
718     1
1256    1
122     1
1412    1
1050    1
525     1
1464    1
752     1
1337    1
527     1
916     1
1225    1
1133    1
727     1
960     1
600     1
1450    1
1195    1
702     1
1148    1
759     1
1430    1
930     1
1098    1
818     1
1319    1
Name: Num_Credit_Card, dtype: int64
```

As we can see few values are very high , so let's make take appropriate steps to correct it.

In [31]:
```
# Group by customer id & replace the Num_Credit_Card value with mode of that particular group.

data['Num_Credit_Card'] = data.groupby('Customer_ID')['Num_Credit_Card'].transform(lambda x: x.mode().iloc[(
```

In [32]:
```
# Checking if data cleaing is dome properly for 'Num_Credit_Card' column or not.

data['Num_Credit_Card'].value_counts().sort_values(ascending = True).head(50)
```

Out[32]:
```
0        16
11       40
1      2184
2      2208
9      4736
10     4960
8      5096
3     13576
4     14336
6     16960
7     16984
5     18904
Name: Num_Credit_Card, dtype: int64
```

In [33]:
```
# Data cleaning for 'Interest_Rate' column:

data['Interest_Rate'].value_counts().sort_values(ascending = False).head(60)
```

Out[33]:
```
8       5012
5       4979
6       4721
12      4540
10      4540
7       4494
9       4494
11      4428
18      4102
15      3992
20      3929
17      3813
16      3730
19      3630
3       2765
1       2683
4       2589
2       2465
13      2384
```

In [34]:
```
# Function to replace values greater than 40 with the mode within each customer group

def replace_high_interest_rate(group):
    mode_group = group.mode().iloc[0]
    group[group > 40] = mode_group
    return group

data['Interest_Rate'] = data.groupby('Customer_ID')['Interest_Rate'].transform(replace_high_interest_rate)
```

In [35]:
```
# Checking the results:

data['Interest_Rate'].value_counts().sort_values(ascending = False).head(60)
```

Out[35]:
```
8     5104
5     5096
6     4832
12    4648
10    4616
7     4584
9     4576
11    4512
18    4192
15    4072
20    4008
17    3888
16    3800
19    3704
3     2824
1     2744
4     2640
2     2520
13    2432
```

In [36]:
```
data['Num_of_Loan'].value_counts()
```

Out[36]:
```
3       14386
2       14250
4       14016
0       10380
1       10083
        ...
449         1
1135        1
147         1
515         1
472         1
Name: Num_of_Loan, Length: 434, dtype: int64
```

In [37]:
```
# Replacing the '_' in 'Num_of_Loan' column.
data['Num_of_Loan'] = data['Num_of_Loan'].str.replace('_','')

# Changing the dtype to int
data['Num_of_Loan'] = data['Num_of_Loan'].astype('int')

# Replacing the outliers with mode for each customer
data['Num_of_Loan'] = data.groupby('Customer_ID')['Num_of_Loan'].transform(lambda x: x.mode().iloc[0])
```

In [38]:
```
# Checking the results:

data['Num_of_Loan'].value_counts()
```

Out[38]:
```
3    15752
2    15712
4    15456
0    11408
1    11128
6     8144
7     7680
5     7528
9     3856
8     3336
Name: Num_of_Loan, dtype: int64
```

In [39]:
```
# Replacing the null values in 'Type_of_Loan' with 'Not Specified'.

data['Type_of_Loan'] = data['Type_of_Loan'].fillna('Not Specified')
```

In [40]:
```
# Cleaning in 'Num_of_Delayed_Payment'.

data['Num_of_Delayed_Payment'].value_counts()
```

Out[40]:
```
19      5327
17      5261
16      5173
10      5153
18      5083
        ...
3037       1
848_       1
813        1
2413       1
1087       1
Name: Num_of_Delayed_Payment, Length: 749, dtype: int64
```

In [41]:
```
# Replacing the '_' in 'Num_of_Delayed_Payment' column.

data['Num_of_Delayed_Payment'] = data['Num_of_Delayed_Payment'].str.replace('_','')
data['Num_of_Delayed_Payment'] = data['Num_of_Delayed_Payment'].str.replace('-','')
data['Num_of_Delayed_Payment'] = data['Num_of_Delayed_Payment'].fillna(0)

# Changing the dtype to int
data['Num_of_Delayed_Payment'] = data['Num_of_Delayed_Payment'].astype('int')

# Replacing the outliers with mode for each customer
data['Num_of_Delayed_Payment'] = data.groupby('Customer_ID')['Num_of_Delayed_Payment'].transform(lambda x: 
```

In [42]:
```
# Checking the results:

data['Num_of_Delayed_Payment'].value_counts().sort_values(ascending = False).head(60)
```

Out[42]:
```
19    6264
20    6096
10    6088
16    6008
15    5880
8     5816
17    5656
18    5632
9     5624
12    5504
11    5208
0     4192
14    3920
13    3712
25    2160
5     2128
6     2120
1     2112
21    2104
```

In [43]:
```
data['Num_Credit_Inquiries'].value_counts()
```

Out[43]:
```
4.0       11271
3.0        8890
6.0        8111
7.0        8058
2.0        8028
          ...
253.0         1
2352.0        1
2261.0        1
519.0         1
1801.0        1
Name: Num_Credit_Inquiries, Length: 1223, dtype: int64
```

In [44]:
```
Replacing the extreme values with mode value of each customer data
ta['Num_Credit_Inquiries'] = data.groupby('Customer_ID')['Num_Credit_Inquiries'].transform(lambda x: x.mode(

Changing the data type from float to int
ta['Num_Credit_Inquiries'] = data['Num_Credit_Inquiries'].astype('int')
```

In [45]:
```
data['Num_Credit_Inquiries'].value_counts()
```

Out[45]:
```
4     11936
3      9416
2      8568
7      8416
6      8264
8      8152
1      8104
0      7504
5      5728
9      5304
11     5280
10     5016
12     4592
13     1344
14      960
15      728
16      416
17      272
Name: Num_Credit_Inquiries, dtype: int64
```

In [46]:
```
data['Credit_Mix'].value_counts()
```

Out[46]:
```
Standard    36479
Good        24337
_           20195
Bad         18989
Name: Credit_Mix, dtype: int64
```

In [47]:
```
# Cleaning the Credit Mix column by giving same credit mix to each customer.

# Convert the column to string type
data['Credit_Mix'] = data['Credit_Mix'].astype(str)

# Replace underscores with NaN
data['Credit_Mix'] = data['Credit_Mix'].replace('_', np.nan)

# Fill NaN values with the mode within each customer group
data['Credit_Mix'] = data.groupby('Customer_ID')['Credit_Mix'].transform(lambda x: x.fillna(x.mode().iloc[0
```

In [48]:
```
# Changing the data type of Outstanding_Debt column:
data['Outstanding_Debt'] = data['Outstanding_Debt'].str.replace('_','')

data['Outstanding_Debt'] = data['Outstanding_Debt'].astype('float')
```

In [49]:
```
# Replacing the values in Credit_History_Age with mode of that particular customer.

data['Credit_History_Age'] = data.groupby('Customer_ID')['Credit_History_Age'].transform(lambda x: x.mode()
```

In [50]:
```
data['Payment_of_Min_Amount'].value_counts()
```

Out[50]:
```
Yes    52326
No     35667
NM     12007
Name: Payment_of_Min_Amount, dtype: int64
```

In [51]:
```
data['Amount_invested_monthly'] = data['Amount_invested_monthly'].str.replace('_','')

data['Amount_invested_monthly'] = data['Amount_invested_monthly'].astype('float')

data['Amount_invested_monthly'] = data.groupby('Customer_ID')['Amount_invested_monthly'].transform(lambda x
```

In [52]:
```

def replace_payment_behaviour(group):
    mode_value = group.mode().iloc[0]
    group[group == '!@9#%8'] = mode_value
    return group


data['Payment_Behaviour'] = data.groupby('Customer_ID')['Payment_Behaviour'].transform(replace_payment_beha
data['Payment_Behaviour'] = data['Payment_Behaviour'].str.replace('!@9#%8', 'Not mentioned')

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

Out[52]:
```
Low_spent_Small_value_payments      27489
High_spent_Medium_value_payments    18911
High_spent_Large_value_payments     14911
Low_spent_Medium_value_payments     14414
High_spent_Small_value_payments     11771
Low_spent_Large_value_payments      10768
Not mentioned                        1736
Name: Payment_Behaviour, dtype: int64
```

In [53]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 2 to 99996
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                      100000 non-null  object 
 4   Age                       100000 non-null  int64  
 5   SSN                       100000 non-null  object 
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  float64
 8   Monthly_Inhand_Salary     100000 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  int64  
 13  Type_of_Loan              100000 non-null  object 
 14  Delay_from_due_date       100000 non-null  int64  
 15  Num_of_Delayed_Payment    100000 non-null  int64  
 16  Changed_Credit_Limit      100000 non-null  object 
 17  Num_Credit_Inquiries      100000 non-null  int64  
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  float64
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        100000 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   100000 non-null  float64
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           98800 non-null   object 
dtypes: float64(6), int64(8), object(13)
memory usage: 21.4+ MB
```

In [54]:
```
data['Monthly_Balance'].value_counts()
```

Out[54]:
```
__-333333333333333333333333333__    9
350.0148691                         2
695.0571561                         2
419.7651674                         1
615.6677195                         1
                                   ..
259.3760946                         1
343.7619864                         1
288.6680278                         1
468.4784226                         1
337.380877                          1
Name: Monthly_Balance, Length: 98790, dtype: int64
```

In [55]:
```
# Cleaning process of Monthly Balance Column:

data['Monthly_Balance'] = data['Monthly_Balance'].str.replace('__', '')
data['Monthly_Balance'] = data['Monthly_Balance'].str.replace('-', '')

# Changing the data type to float
data['Monthly_Balance'] = data['Monthly_Balance'].astype('float')

# Handelling the null values by replacing them with mean Monthly_Balance of that group.

# Step 1: Calculating the mean monthly baalance for each customer id.
mean_monthly_balance = data.groupby('Customer_ID')['Monthly_Balance'].transform('mean')

# Step 2: Replacing the null by mean.
data['Monthly_Balance'] = data['Monthly_Balance'].fillna(mean_monthly_balance)
```

In [56]:
```
data['Monthly_Balance'].isnull().sum()
```

Out[56]:
```
0
```

In [57]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 2 to 99996
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                      100000 non-null  object 
 4   Age                       100000 non-null  int64  
 5   SSN                       100000 non-null  object 
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  float64
 8   Monthly_Inhand_Salary     100000 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  int64  
 13  Type_of_Loan              100000 non-null  object 
 14  Delay_from_due_date       100000 non-null  int64  
 15  Num_of_Delayed_Payment    100000 non-null  int64  
 16  Changed_Credit_Limit      100000 non-null  object 
 17  Num_Credit_Inquiries      100000 non-null  int64  
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  float64
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        100000 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   100000 non-null  float64
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           100000 non-null  float64
dtypes: float64(7), int64(8), object(12)
memory usage: 21.4+ MB
```

```
So till now we have done all the kind of basic data cleaning stuff, handeling the null values and changing
the data types of the columns if required, etc.
We can see there are two columns which if are of no use for our visualization, i.e SSN, ID. LEt's drop
them also.
```

In [58]:
```
# Deleting the not reqired columns.

data.drop(columns = ['SSN', 'ID'], inplace = True)
```

In [59]:
```
data.describe().T
```

Out[59]:

	count	mean	std	min	25%	50%	75%	max
Age	100000.0	3.327456e+01	1.076444e+01	14.000000	24.000000	33.000000	42.000000	5.600000e+01
Annual_Income	100000.0	1.764157e+05	1.429618e+06	7005.930000	19457.500000	37578.610000	72790.920000	2.419806e+07
Monthly_Inhand_Salary	100000.0	4.198262e+03	3.187363e+03	303.645417	1626.594167	3096.066250	5957.715000	1.520463e+04
Num_Bank_Accounts	100000.0	5.367840e+00	2.592597e+00	0.000000	3.000000	5.000000	7.000000	1.000000e+01
Num_Credit_Card	100000.0	5.532720e+00	2.067504e+00	0.000000	4.000000	5.000000	7.000000	1.100000e+01
Interest_Rate	100000.0	1.453208e+01	8.741330e+00	1.000000	7.000000	13.000000	20.000000	3.400000e+01
Num_of_Loan	100000.0	3.532880e+00	2.446356e+00	0.000000	2.000000	3.000000	5.000000	9.000000e+00
Delay_from_due_date	100000.0	2.106878e+01	1.486010e+01	-5.000000	10.000000	18.000000	28.000000	6.700000e+01
Num_of_Delayed_Payment	100000.0	1.300576e+01	6.416920e+00	0.000000	9.000000	13.000000	18.000000	2.800000e+01
Num_Credit_Inquiries	100000.0	5.677760e+00	3.827248e+00	0.000000	3.000000	5.000000	8.000000	1.700000e+01
Outstanding_Debt	100000.0	1.426220e+03	1.155129e+03	0.230000	566.072500	1166.155000	1945.962500	4.998070e+03
Credit_Utilization_Ratio	100000.0	3.228517e+01	5.116875e+00	20.000000	28.052567	32.305784	36.496663	5.000000e+01
Total_EMI_per_month	100000.0	1.403118e+03	8.306041e+03	0.000000	30.306660	69.249473	161.224249	8.233100e+04
Amount_invested_monthly	100000.0	1.823009e+02	1.682554e+02	10.659493	91.356856	139.054802	227.239727	1.000000e+04
Monthly_Balance	100000.0	3.000000e+22	3.162151e+24	0.007760	270.190370	337.126271	471.628361	3.333333e+26

In [60]:
```
data.describe(exclude = np.number).T
```

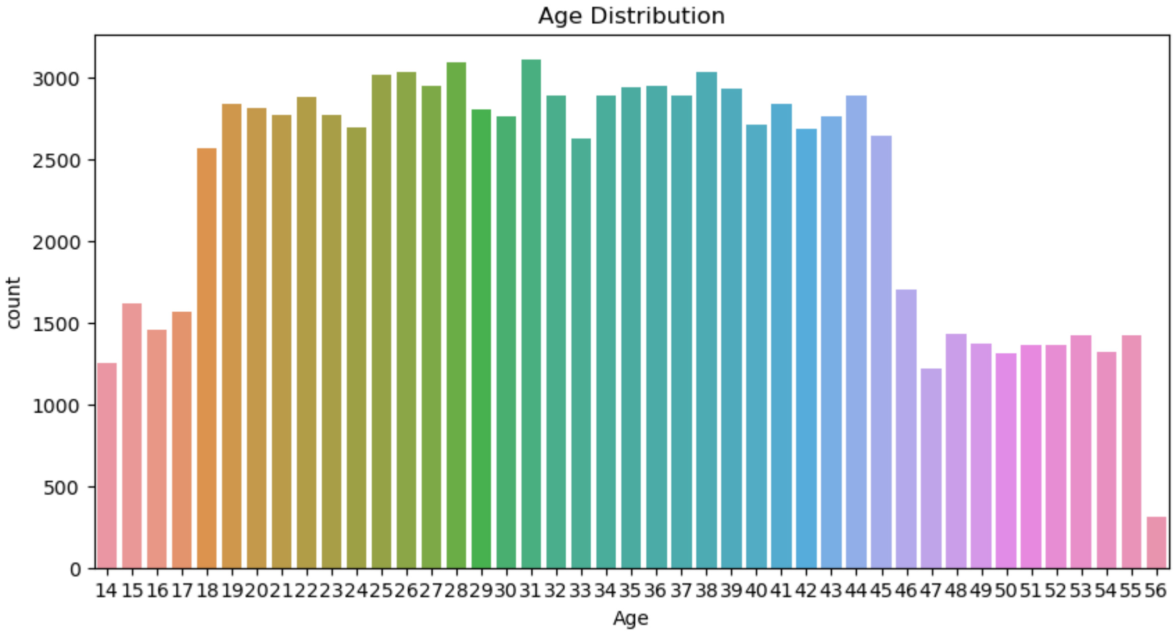
Out[60]:

	count	unique	top	freq
Customer_ID	100000	12500	CUS_0x1000	8
Month	100000	8	April	12500
Name	100000	10139	Jessicad	48
Occupation	100000	15	Lawyer	7096
Type_of_Loan	100000	6260	Not Specified	12816
Changed_Credit_Limit	100000	3635	_	2091
Credit_Mix	100000	3	Standard	45848
Credit_History_Age	100000	249	15 Years and 10 Months	3488
Payment_of_Min_Amount	100000	3	Yes	52326
Payment_Behaviour	100000	7	Low_spent_Small_value_payments	27489

```
Some insights:-
1) Totla 12500 unique customer data is present.
2) Total 8 month data is available, with april occuring the most.
3) Unique 15 occupation is present with Lawyer being the most frequent.
```

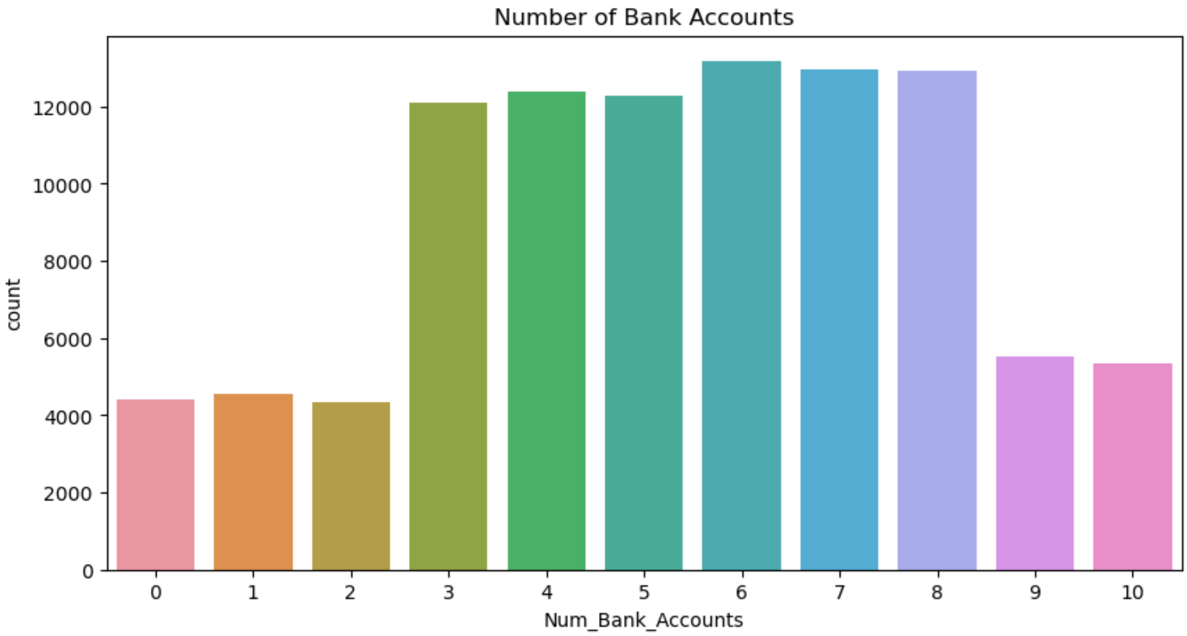
Univariete Analysis:

In [61]:
```
# Visualization to see the Age distribution

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x='Age')
plt.title("Age Distribution")
plt.show()
```



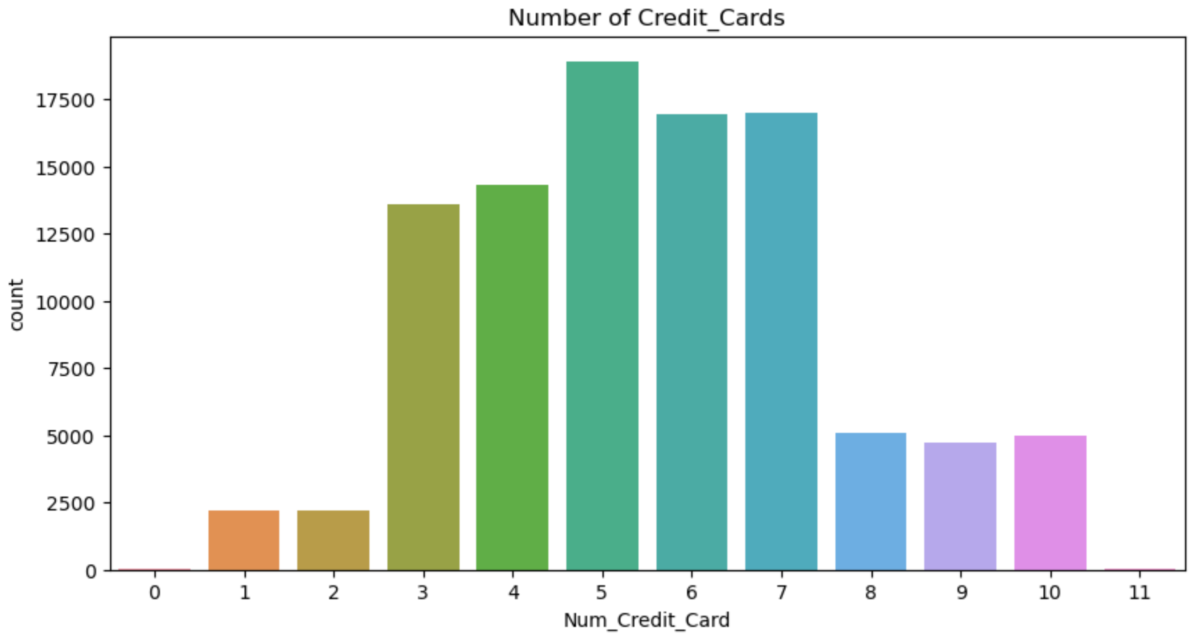
From the above graph we can say maximum people has age between 18 to 45 in our data set.

In [62]:
```
# Visualization to see the Number of bank accounts distribution

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Num_Bank_Accounts')
plt.title("Number of Bank Accounts")
plt.show()
```



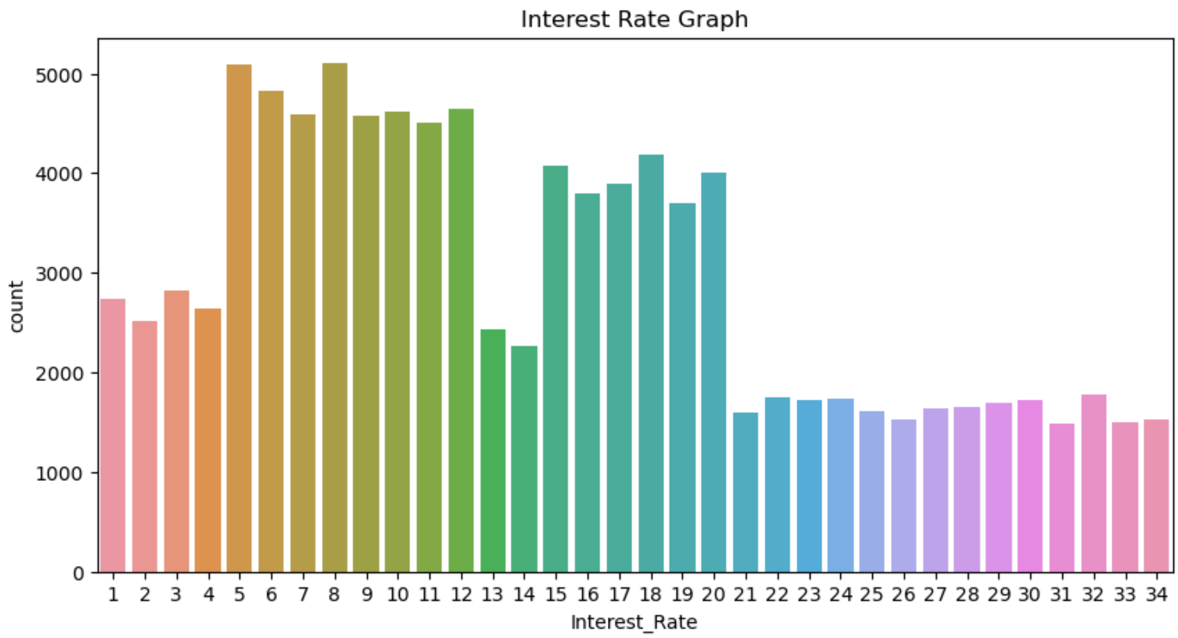
From above graph we can see maximum customer has 3 to 8 Bank Accounts.

In [63]:
```
# Visualization to see the Number of Credit Card distribution

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Num_Credit_Card')
plt.title("Number of Credit_Cards")
plt.show()
```



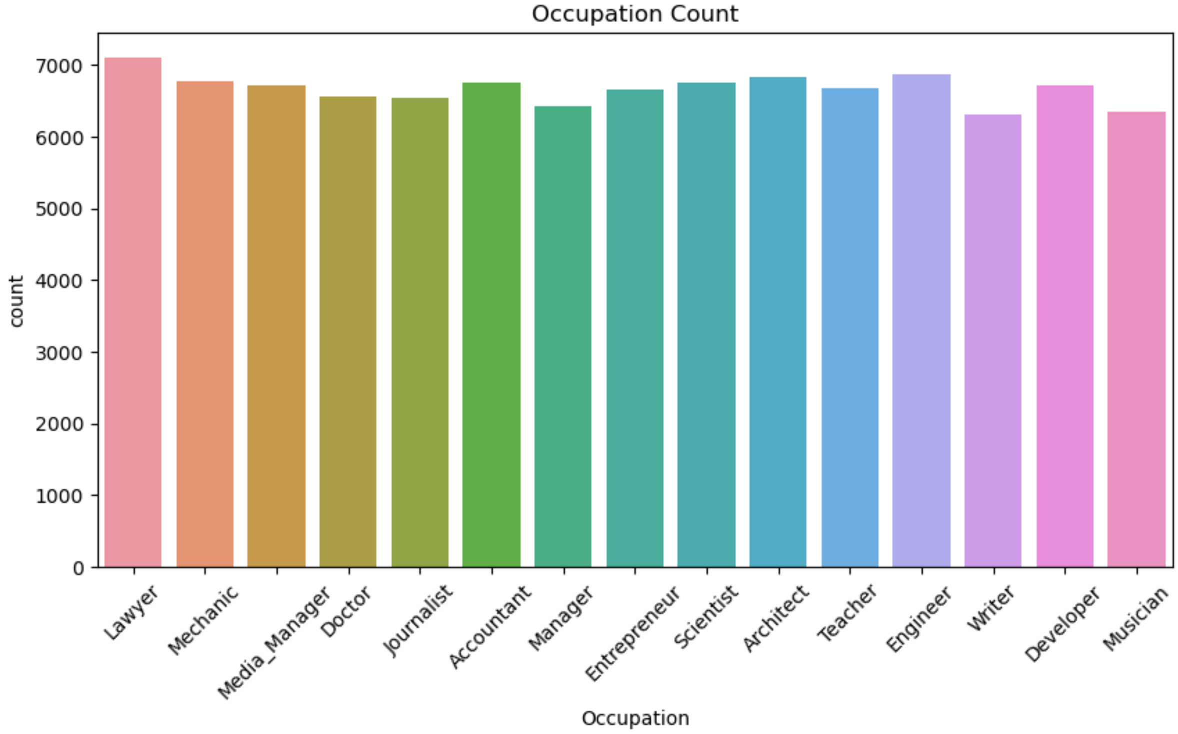
From above graph we can see maximum customer has 3 to 7 Credit Cards

In [64]:
```
# Visualization to see the Interest Rate distribution

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Interest_Rate')
plt.title("Interest Rate Graph")
plt.show()
```



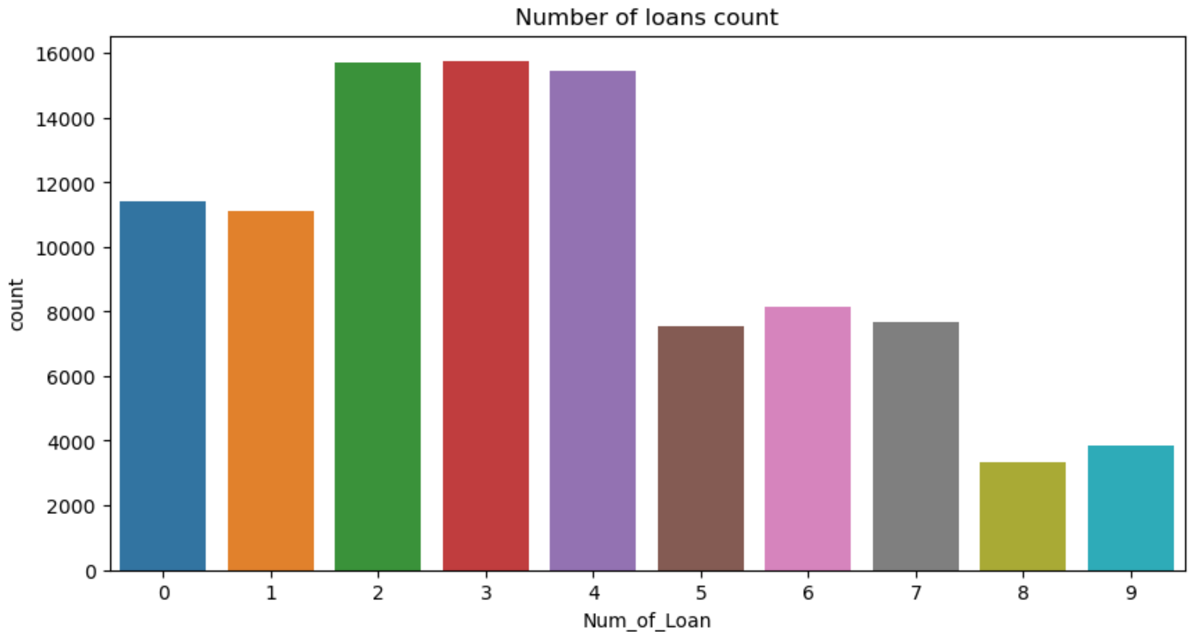
Maximum interest rate are between 5 to 12 or 15 to 20. Overall the range is 1 to 34.

In [65]:
```
# Visualization to see the Occupation distribution

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Occupation')
plt.xticks(rotation = 45)
plt.title("Occupation Count")
plt.show()
```



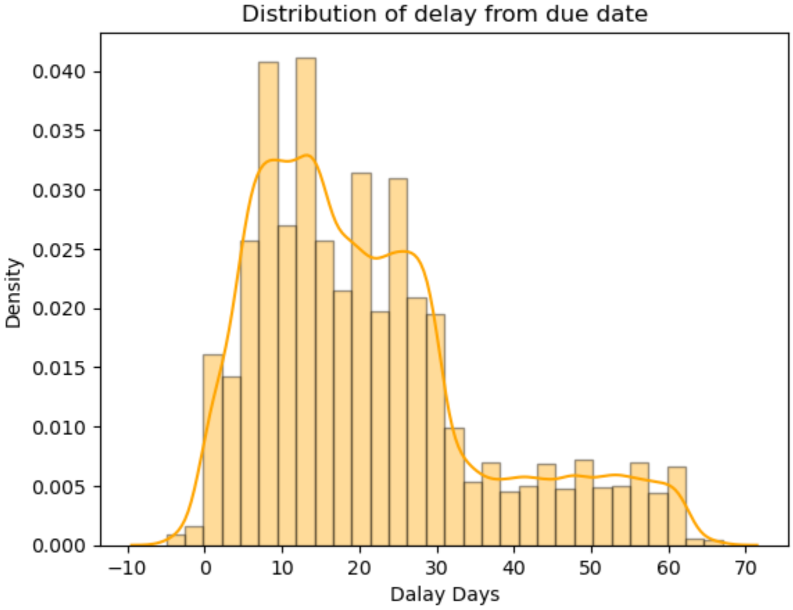

```
From above graph we can say all Occupation are near to equally distributed.
```

In [66]:
```
# Visualization to see the Number of Loans distribution

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Num_of_Loan')
plt.title("Number of loans count")
plt.show()
```



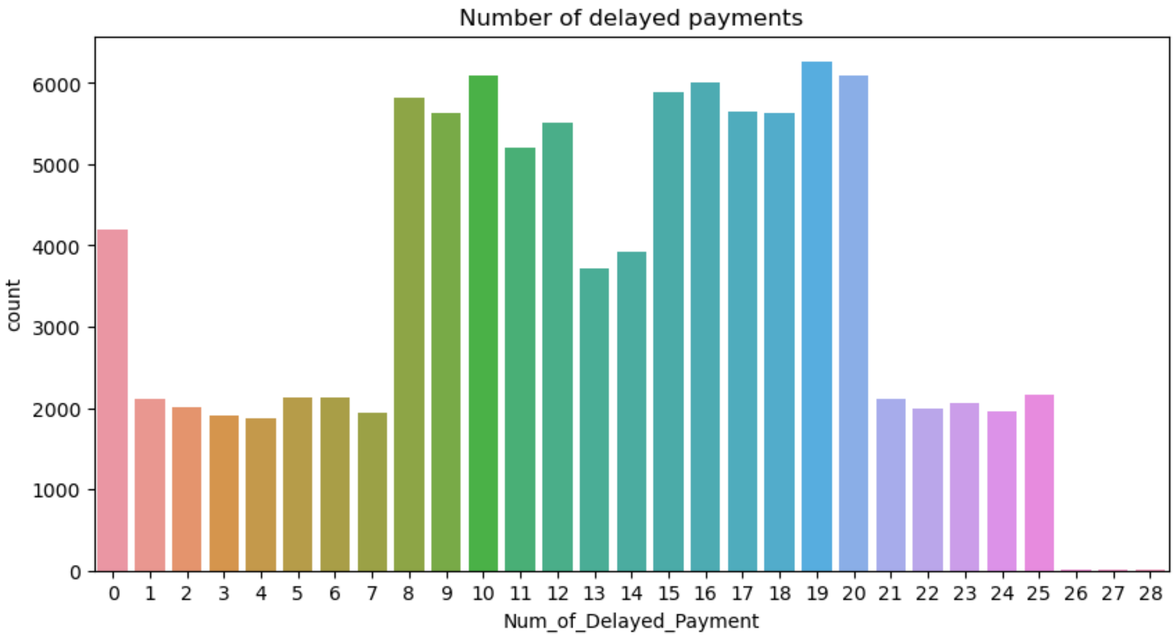

```
From aboe graph it's clear maximum customer has 2 to 4 loan.
```

In [67]:
```
# Distribution of Delay from due date column

sns.distplot(data['Delay_from_due_date'], bins=30, color='orange', hist_kws={'edgecolor': 'black'})
plt.xlabel('Dalay Days')
plt.ylabel('Density')
plt.title('Distribution of delay from due date')
plt.show()
```

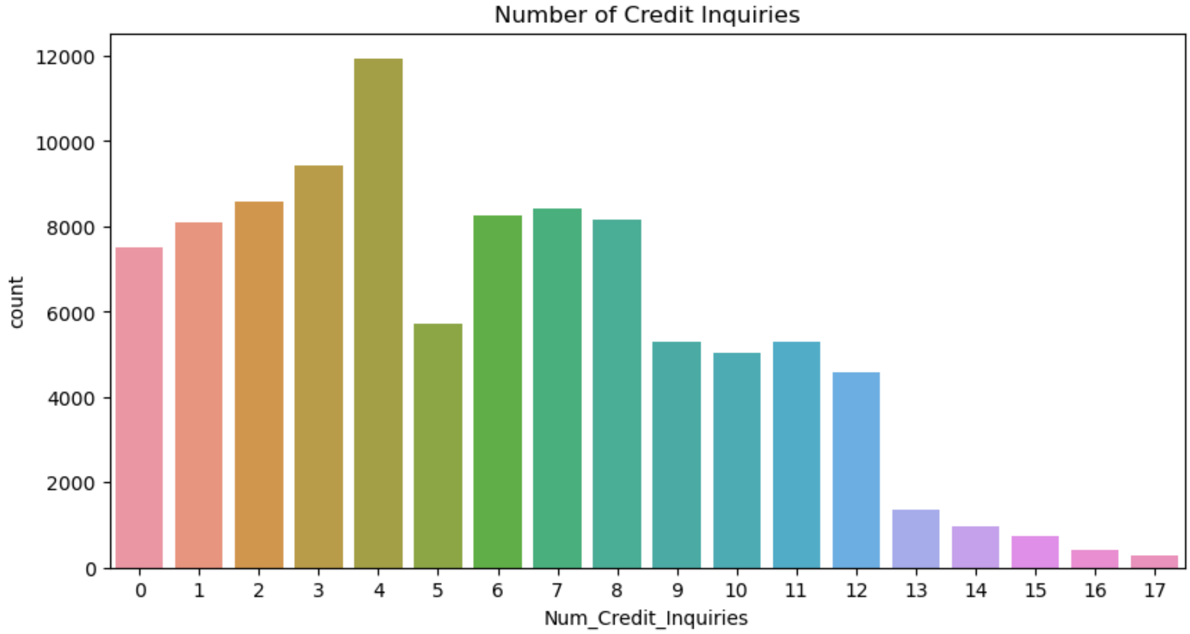


From above graph we can say majorly the delay from due date ranges from 0 to 30 days, out of that also 8 to 12 days are most common.

In [68]:
```
# Visualization to see the Number of delayed payment.

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Num_of_Delayed_Payment')
plt.title("Number of delayed payments")
plt.show()
```



In [69]:
```
# Visualization to see the Number of credit inquiries.

plt.figure(figsize=(10, 5))
sns.countplot(data=data, x = 'Num_Credit_Inquiries')
plt.title("Number of Credit Inquiries")
plt.show()
```

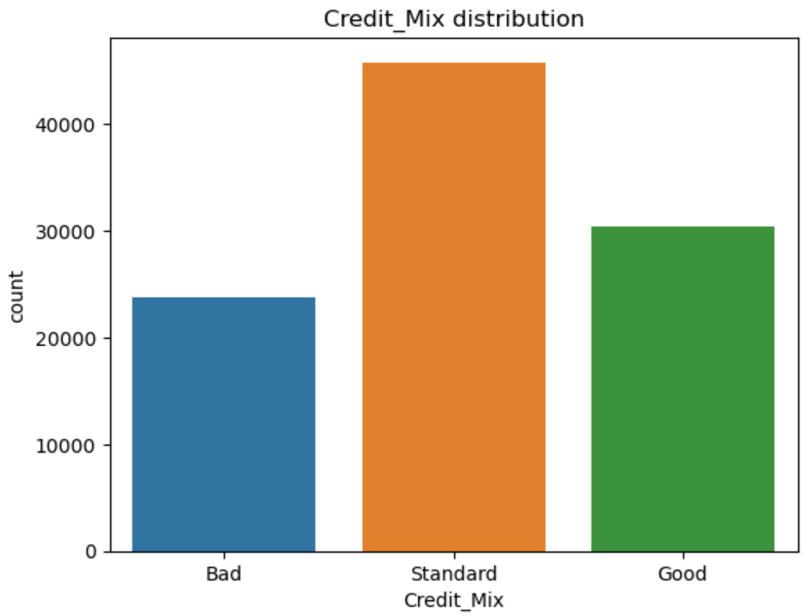



```
Credit enqiry ranges from 0 to 17 counts, out of which also 0 to 4 count is most. 4 being the most occured value.
```

In [70]:
```
data['Credit_Mix'].value_counts()
```

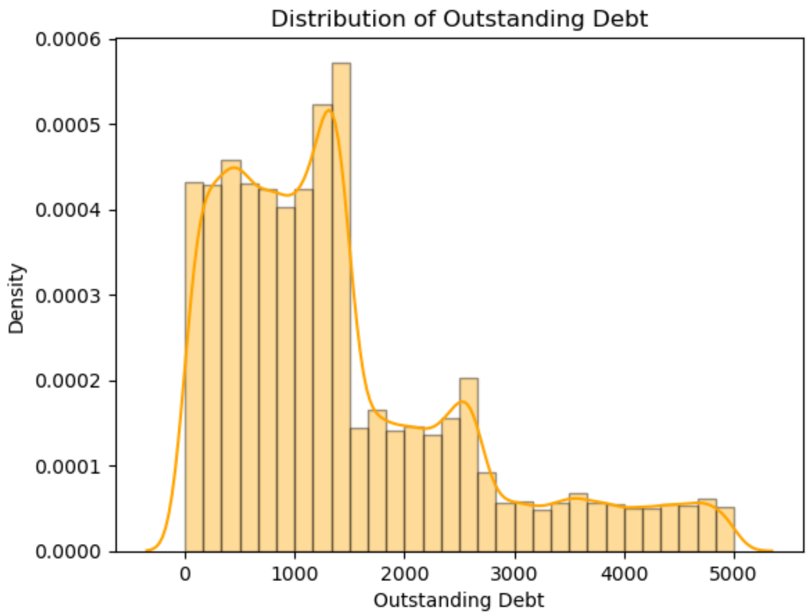
Out[70]:
```
Standard    45848
Good        30384
Bad         23768
Name: Credit_Mix, dtype: int64
```

In [71]:
```
# Visualization to see the Credit Mix Distribution.

sns.countplot(data=data, x = 'Credit_Mix')
plt.title("Credit_Mix distribution")
plt.show()
```

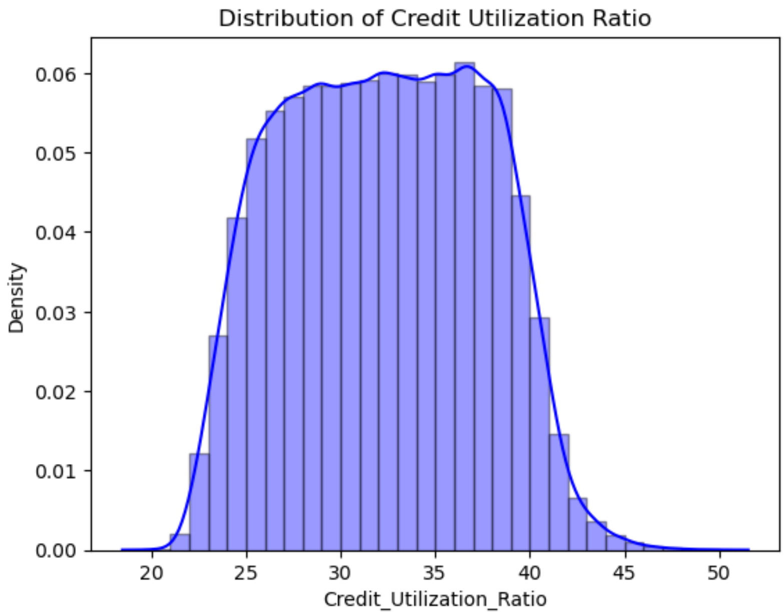



```
From the above graph it's clear maximum Customer has Standard Credit Mix.
```

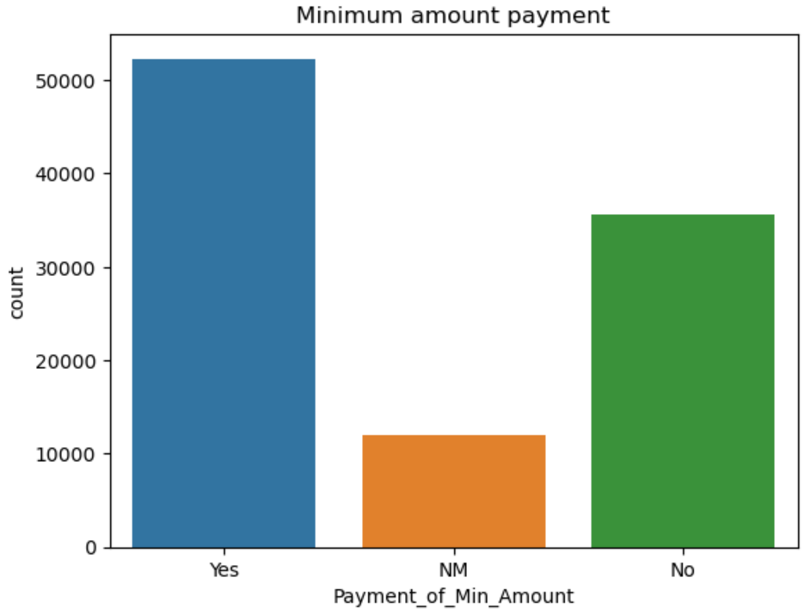
In [72]:
```
# Distribution of Outstanding Debt.

sns.distplot(data['Outstanding_Debt'], bins=30, color='orange', hist_kws={'edgecolor': 'black'})
plt.xlabel('Outstanding Debt')
plt.ylabel('Density')
plt.title('Distribution of Outstanding Debt')
plt.show()
```



In [73]:
```
# Distribution of Credit Utilization Ratio.

sns.distplot(data['Credit_Utilization_Ratio'], bins=30, color='blue', hist_kws={'edgecolor': 'black'})
plt.xlabel('Credit_Utilization_Ratio')
plt.ylabel('Density')
plt.title('Distribution of Credit Utilization Ratio')
plt.show()
```

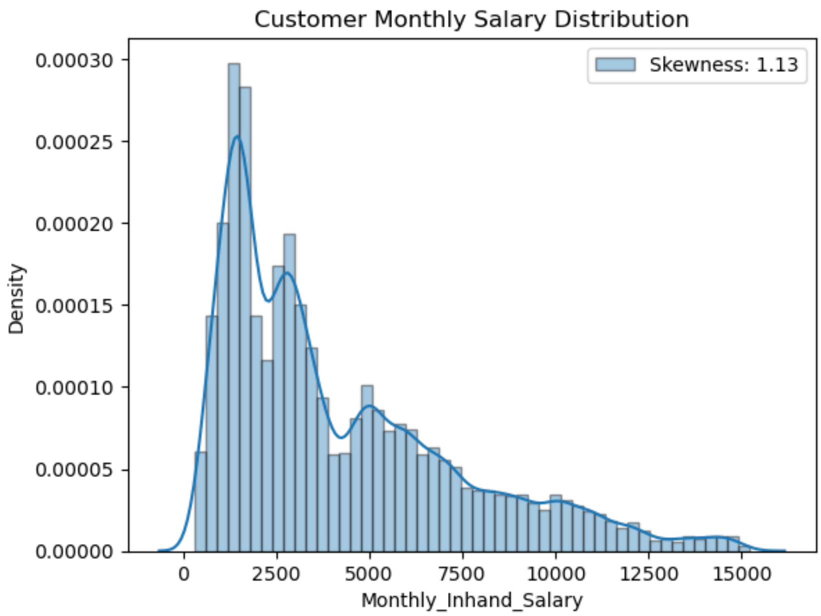


In [74]:
```
# Visualization to see the Payment of min amount Distribution.

sns.countplot(data=data, x = 'Payment_of_Min_Amount')
plt.title("Minimum amount payment")
plt.show()
```



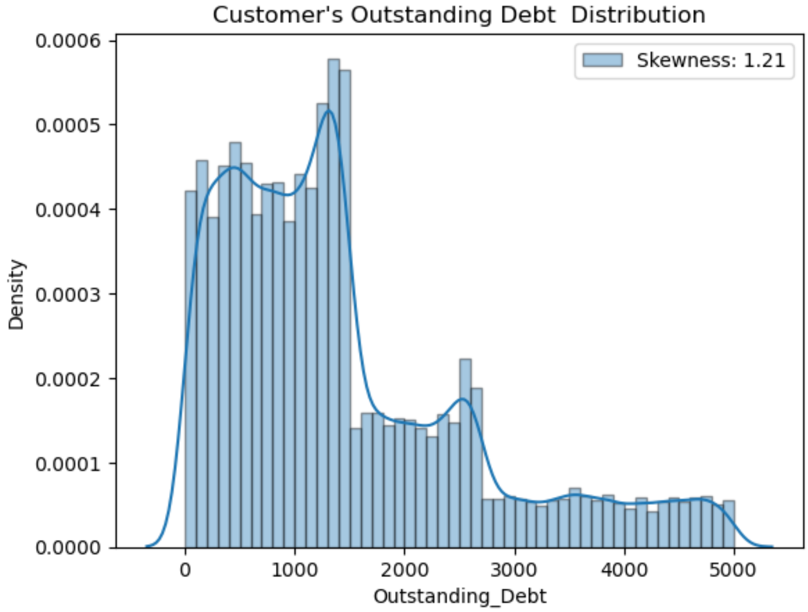
From the above graph we can say maximum people has paid the minimum amount payable but their is significant number of people who has not paid.

In [75]:
```
# Customer Monthly Salary Distribution

sns.distplot(data['Monthly_Inhand_Salary'],hist_kws={'edgecolor': 'black'}, label = 'Skewness: %.2f'%(data[
plt.legend(loc = 'best')
plt.title('Customer Monthly Salary Distribution')
plt.show()
```



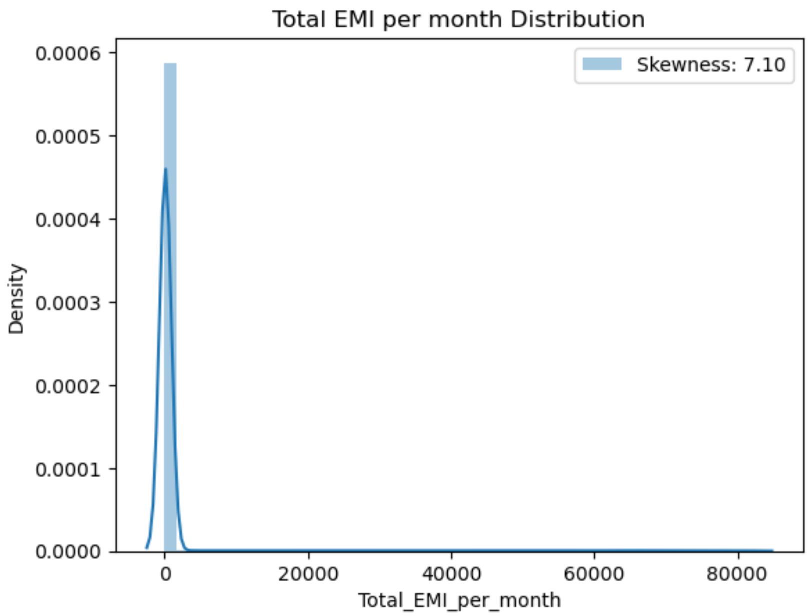
From the above graph we can say it is right skewed & maximum customer has salary on lower side.

In [76]:
```
# Customer's Outstanding Debt  Distribution

sns.distplot(data['Outstanding_Debt'], hist_kws={'edgecolor': 'black'}, label = 'Skewness: %.2f'%(data['Out
plt.legend(loc = 'best')
plt.title("Customer's Outstanding Debt  Distribution")
plt.show()
```

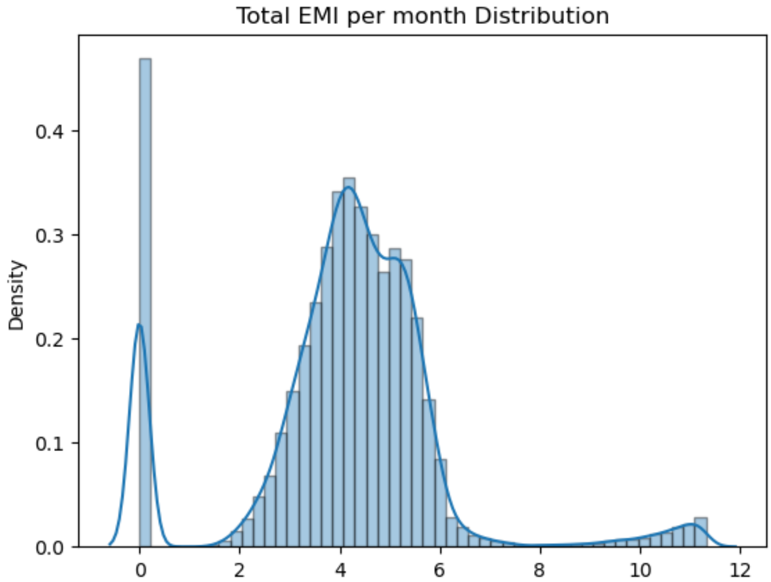


From above graph we can say maximum customer has debt from 0 to 10,000.

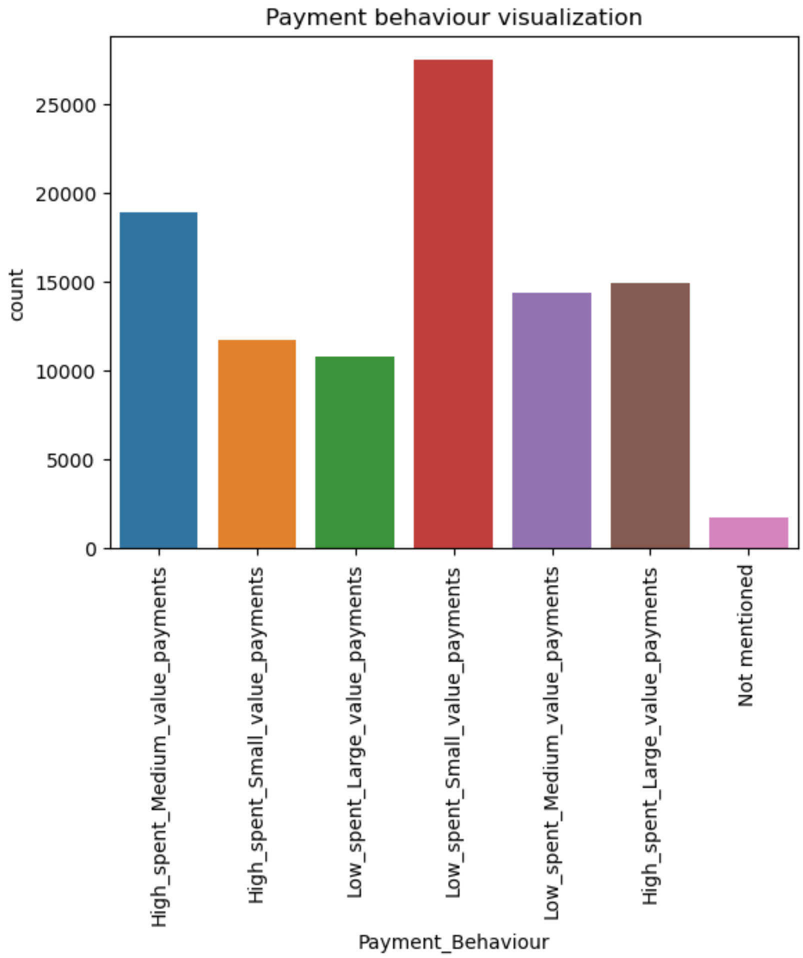
In [77]:
```
# Understanding the distribution of the column - Total_EMI_per_month

sns.distplot(data['Total_EMI_per_month'], label = 'Skewness: %.2f'%(data['Total_EMI_per_month'].skew()))
plt.legend(loc = 'best')
plt.title('Total EMI per month Distribution')
plt.show()
```



In [78]:
```
### Understanding the distribution of the data log(Total_EMI_per_month)

modified_emi = [np.log(emi) if emi > 0 else 0 for emi in data['Total_EMI_per_month']]

sns.distplot(modified_emi, hist_kws={'edgecolor': 'black'})
plt.title('Total EMI per month Distribution')
plt.show()
```



In [79]:
```
# Visualization to see the Payment Behaviour Distribution.

sns.countplot(data=data, x = 'Payment_Behaviour')
plt.title("Payment behaviour visualization")
plt.xticks(rotation = 90)
plt.show()
```



From above graph we can say , maximum we have Low Spend Small value payments.

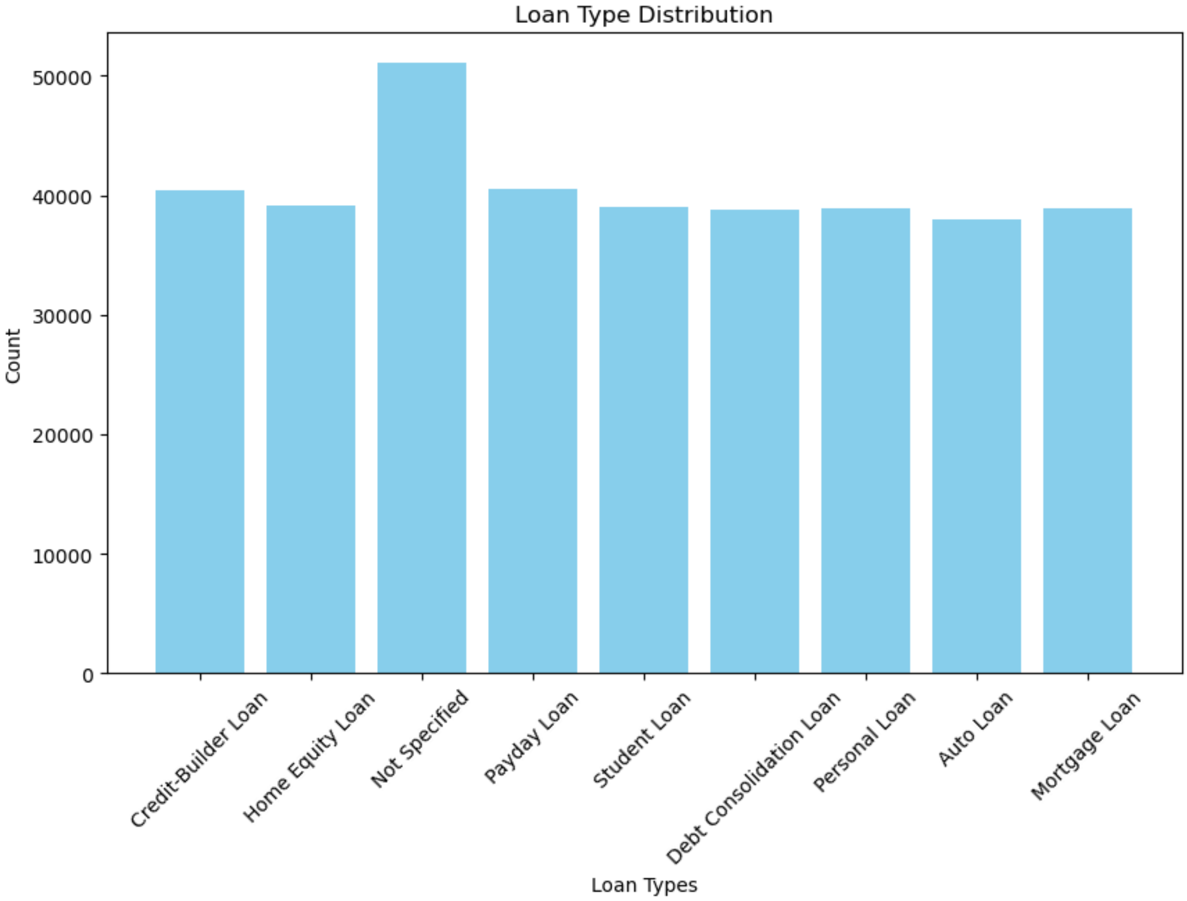
In [80]:
```python
# Analysing the type of Loan Column :

# Fetching the not null data of the column - Type of Data

index_values = ~data['Type_of_Loan'].isnull().values

# making a list of all the kind of loan type
loan_type_data = list(data['Type_of_Loan'][index_values])
loan_type_data
```

Out[80]:
```
['Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Credit-Builder Loan, and Home Equity Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified, Home Equity Loan, Credit-Builder Loan, and Payday Loan',
 'Not Specified',
 'Not Specified',
 'Not Specified',
 'Not Specified',
```

In [81]:
```python
# Creating a dictionary to store the counts of all the various loan types

loan_type_dict = dict()
for value in loan_type_data:
    values = value.split(',')
    for each_value in values:
        loan_type = each_value.strip(' ')
        if 'and' in loan_type:
            loan_type = loan_type[4 : ]
        if loan_type in loan_type_dict:
            loan_type_dict[loan_type] += 1
        else:
            loan_type_dict[loan_type] = 1

loan_type_dict
```

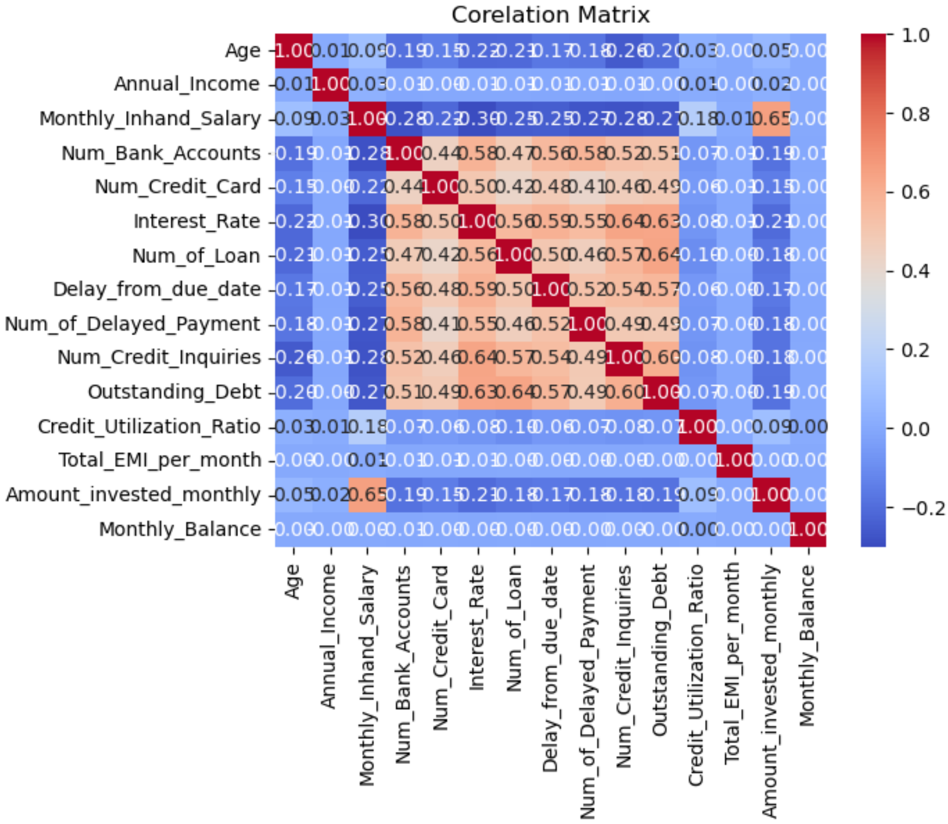
Out[81]:
```
{'Credit-Builder Loan': 40440,
 'Home Equity Loan': 39104,
 'Not Specified': 51024,
 'Payday Loan': 40568,
 'Student Loan': 38968,
 'Debt Consolidation Loan': 38776,
 'Personal Loan': 38888,
 'Auto Loan': 37992,
 'Mortgage Loan': 38936}
```

In [82]:
```
# Extract loan types and counts from the dictionary
loan_types = list(loan_type_dict.keys())
loan_counts = list(loan_type_dict.values())

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(loan_types, loan_counts, color='skyblue')
plt.xlabel('Loan Types')
plt.ylabel('Count')
plt.title('Loan Type Distribution')
plt.xticks(rotation = 45)
plt.show()
```



From above graph we can say Payday Loan & Credit-Builder Loan is most famous.

In [83]:
```
# Corelation analysis:

corelation = data.corr()
sns.heatmap(corelation, annot = True, cmap = 'coolwarm', fmt = '.2f')
plt.title("Corelation Matrix")
plt.show()
```




```
From this above corelation analysis we can find many hidden corelations in our data.
```

In [84]:
```
data.columns
```

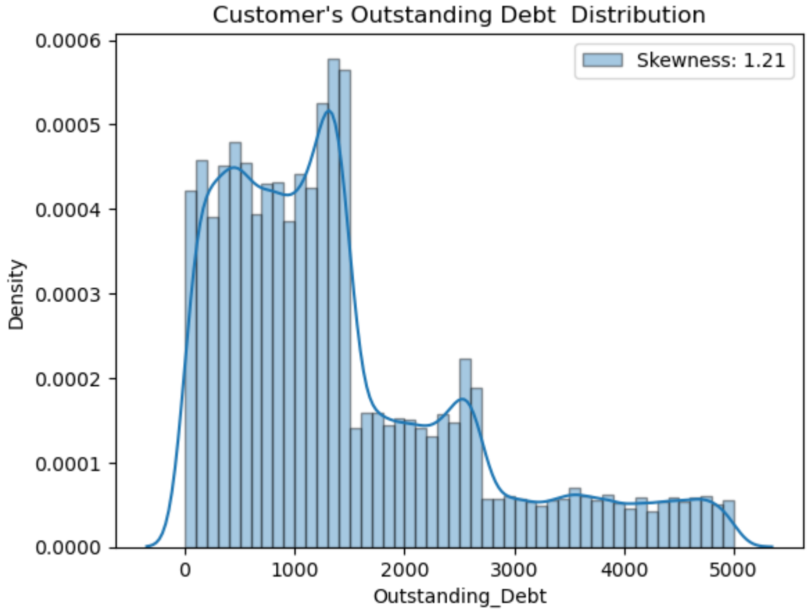
Out[84]:
```
Index(['Customer_ID', 'Month', 'Name', 'Age', 'Occupation', 'Annual_Income',
       'Monthly_Inhand_Salary', 'Num_Bank_Accounts', 'Num_Credit_Card',
       'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan', 'Delay_from_due_date',
       'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       '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')
```

In [85]:
```
# Converting the Credit history in months

def convert_month(value):
    if pd.notnull(value):
        years = int(value.split(' ')[0])
        months = int(value.split(' ')[3])
        return (years*12)+months
    else:
        return value

data['Credit_History_Months'] = data['Credit_History_Age'].apply(lambda x: convert_month(x)).astype(int)
```

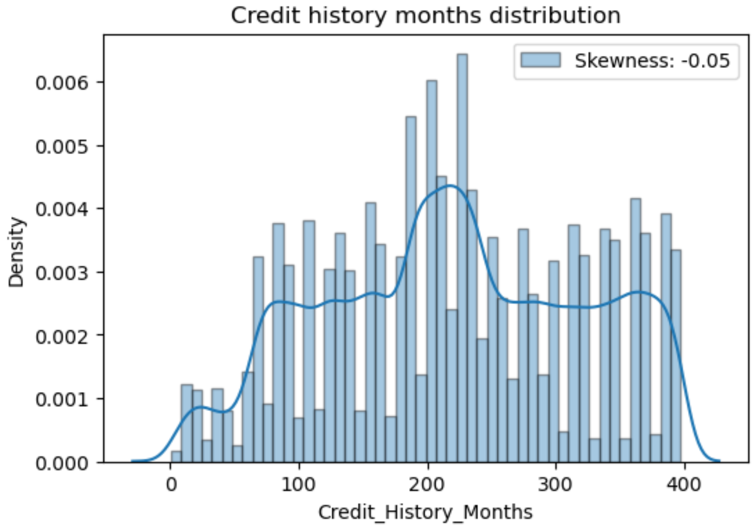
In [86]:
```
data['Credit_History_Months'].value_counts()
```

Out[86]:
```
190    3488
214    3480
226    3264
238    3224
202    3088
       ...
297       8
388       8
208       8
225       8
352       8
Name: Credit_History_Months, Length: 249, dtype: int64
```

In [87]:
```
Customer's Outstanding Debt  Distribution

ıs.distplot(data['Outstanding_Debt'], hist_kws={'edgecolor': 'black'}, label = 'Skewness: %.2f'%(data['Outsta
.t.legend(loc = 'best')
.t.title("Customer's Outstanding Debt  Distribution")
.t.show()
```



In [88]:
```
plt.figure(figsize = (6,4))
sns.distplot(data['Credit_History_Months'], hist_kws = {'edgecolor': 'black'}, label = 'Skewness: %.2f'%(da
plt.legend(loc = 'best')
plt.title("Credit history months distribution")
plt.show()
```



In [89]:
```
# Atlast checking again if there is any null value left behind in the data.

data.isnull().sum()
```

Out[89]:
```
Customer_ID                 0
Month                       0
Name                        0
Age                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Type_of_Loan                0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_History_Months       0
dtype: int64
```

Label Encoding to features

From now on we will be working on to give credit score for each customer.

In [90]:
```
data['Payment_of_Min_Amount'].value_counts()
```

Out[90]:
```
Yes    52326
No     35667
NM     12007
Name: Payment_of_Min_Amount, dtype: int64
```

In [91]:
```
data["Payment_of_Min_Amount"] = data["Payment_of_Min_Amount"].replace({"Yes": 1, "No": 0, "NM": 0})
```

In [92]:
```
data["Credit_Mix"].value_counts()
```

Out[92]:
```
Standard    45848
Good        30384
Bad         23768
Name: Credit_Mix, dtype: int64
```

In [93]:
```
data["Credit_Mix"] = data["Credit_Mix"].replace({"Standard": 1, "Good": 2, "Bad": 0})
```

In [94]:
```
data["Payment_Behaviour"] = data["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,
      "Not mentioned": 0
  })
```

In [95]:
```
data['Payment_Behaviour'] = data['Payment_Behaviour'].astype('int')
```

In [96]:
```
data['Payment_Behaviour'].value_counts()
```

Out[96]:
```
1    27489
2    18911
4    14911
3    14414
5    11771
6    10768
0     1736
Name: Payment_Behaviour, dtype: int64
```

Debt to income ratio calculation

In [97]:
```
# Calculating Debt to Income ratio

data['Monthly_Debt_to_Income_Ratio'] = data['Outstanding_Debt'] / data['Monthly_Inhand_Salary']
```

In [98]:
```
### Monthly debt repayment capability

data['Monthly_Debt_Repayment_Capacity'] = data['Monthly_Inhand_Salary'] - data['Total_EMI_per_month']
```

Credit Score Calculation

In [99]:
```
def calculate_credit_score(data):

  # Group by Customer ID, handling month-level data and calculating scores
  grouped_data = data.groupby("Customer_ID").agg(
      Credit_History_Months = ("Credit_History_Months", "max"),  # Use maximum history age,
      Outstanding_Debt = ("Outstanding_Debt", "mean"),
      Credit_Mix = ("Credit_Mix","mean"),
      Monthly_Debt_to_Income_Ratio=("Monthly_Debt_to_Income_Ratio", "mean"),
      Credit_Utilization_Ratio=("Credit_Utilization_Ratio", "mean"),
      Monthly_Debt_Repayment_Capacity=("Monthly_Debt_Repayment_Capacity",'mean'),
      Payment_Behaviour=("Payment_Behaviour","mean"), # Use average payment behaviour encoding
  )

  # Standardize values for numerical features
  grouped_data = (grouped_data - grouped_data.mean()) / grouped_data.std()

  # Calculate weighted scores
  grouped_data["credit_score"] = (
     0.20 * grouped_data["Credit_History_Months"]
      + 0.15 * (1-grouped_data["Monthly_Debt_to_Income_Ratio"]) # Inverse relation as lower the value bette
      + 0.10 * (1-grouped_data["Credit_Utilization_Ratio"]) # Inverse relation lower the better
      + 0.10 * grouped_data["Monthly_Debt_Repayment_Capacity"]
      + 0.25 * grouped_data["Outstanding_Debt"]
      + 0.10 * grouped_data['Credit_Mix']
      + 0.10 * grouped_data["Payment_Behaviour"]
  )

  # Normalize scores to a range of 0 to 100
  grouped_data["credit_score"] = (grouped_data["credit_score"] - grouped_data["credit_score"].min()) / (grou

  return grouped_data.reset_index()


# Calculate scores for all customers
credit_scores_data = calculate_credit_score(data)
credit_scores_data[["Customer_ID","credit_score"]]
```

Out[99]:

	Customer_ID	credit_score
0	CUS_0x1000	54.839167
1	CUS_0x1009	72.220024
2	CUS_0x100b	72.063897
3	CUS_0x1011	64.958104
4	CUS_0x1013	67.549429
...	...	...
12495	CUS_0xff3	58.909807
12496	CUS_0xff4	58.218064
12497	CUS_0xff6	75.324737
12498	CUS_0xffc	55.225046
12499	CUS_0xffd	68.622082

12500 rows × 2 columns

In [100]:
```
# Maximum credit score

credit_scores_data['credit_score'].max()
```

Out[100]: 100.0

In [101]:
```
# Minimum credit score

credit_scores_data['credit_score'].min()
```

Out[101]: 0.0

Insights:-

1. 12500 unique customer data is present.
2. Data contains months from January to August.
3. Total 27 columns were present , out of which we deleted 2 columns named SSN, ID . As they were of no use for our EDA
4. Total 8 types of loans are present in the data.
5. Maximum customer has 3 to 8 bank accounts.
6. Maximum Customer has 3 to 7 credit cards, 5 being the top.
7. Age is distributed from 14 to 56, 18 to 46 being the most customers belongs to.
8. Interest Rates ranging from 1 to 34, maximum custoomer belong from 5 to 12% slab.
9. All occupation are near to equally distributed. LAwyer being the most occured.
10. Maximum customer has 2 to 4 loans.
11. We can say majorly the delay from due date ranges from 0 to 30 days, out of that also 8 to 12 days are most common.
12. Maximum customer belongs from 8 to 20 counts for delayed payemtns.
13. Number of Credit inquiry ranges from 1 to 17, 4 being the top , followed by 3.
14. Maximum customer are having standard credit mix type.
15. Maximum customer are having debt from 0 to 13,000.
16. Monthly salary is also right skewed , i.e. customer has less salary.
17. "Low spend small value paymetn" most customer belong from this category.

For Credit score we have utilized following features.

1. Credit history months 20%
2. Monthly debt to income ratio 15%
3. Credit utilization ratio 10%
4. Monthly_Debt_Repayment_Capacity 10%
5. Outstanding_Debt 25%
6. Credit_Mix 10%
7. Payment_Behaviour 10%

Recommendations

1. We can completely automate this credit scoring process of customers using Machine learning models. Which can be more effecient and can provide more optimized scores.
2. We have given different weighting for our credit score, different banks etc can have different conditions and weightage. Results may vary according to that.