Credit EDA Case Study

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import re
from sklearn.preprocessing import LabelEncoder
import warnings
import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
# Suppress only Runtimewarnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

%matplotlib inline
```

Reading Data

In [2]:
```
df = pd.read_csv("Credit_score.csv")
df.head(10)
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\3791055799.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  df = pd.read_csv("Credit_score.csv")
```

Out[2]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts
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	0x1607	CUS_0xd40	June	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3
6	0x1608	CUS_0xd40	July	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts
7	0x1609	CUS_0xd40	August	NaN	23	#F%$D@*&8	Scientist	19114.12	1824.843333	3
8	0x160e	CUS_0x21b1	January	Rick Rothackerj	28_	004-07-5839	_______	34847.84	3037.986667	2
9	0x160f	CUS_0x21b1	February	Rick Rothackerj	28	004-07-5839	Teacher	34847.84	3037.986667	2

Exploratory Data Analysis

In [3]:
```
df.shape
```

Out[3]:
```
(100000, 27)
```

In [4]:
```
df.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
```

In [5]:
```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Monthly_Inhand_Salary	84998.0	4194.170850	3183.686167	303.645417	1625.568229	3093.745000	5957.448333	15204.63333
Num_Bank_Accounts	100000.0	17.091280	117.404834	-1.000000	3.000000	6.000000	7.000000	1798.00000
Num_Credit_Card	100000.0	22.474430	129.057410	0.000000	4.000000	5.000000	7.000000	1499.00000
Interest_Rate	100000.0	72.466040	466.422621	1.000000	8.000000	13.000000	20.000000	5797.00000
Delay_from_due_date	100000.0	21.068780	14.860104	-5.000000	10.000000	18.000000	28.000000	67.00000
Num_Credit_Inquiries	98035.0	27.754251	193.177339	0.000000	3.000000	6.000000	9.000000	2597.00000
Credit_Utilization_Ratio	100000.0	32.285173	5.116875	20.000000	28.052567	32.305784	36.496663	50.00000
Total_EMI_per_month	100000.0	1403.118217	8306.041270	0.000000	30.306660	69.249473	161.224249	82331.00000

In [6]:

```
df.describe(exclude=np.number).T
```

Out[6]:

	count	unique	top	freq
ID	100000	100000	0x1602	1
Customer_ID	100000	12500	CUS_0xd40	8
Month	100000	8	January	12500
Name	90015	10139	Langep	44
Age	100000	1788	38	2833
SSN	100000	12501	#F%$D@*&8	5572
Occupation	100000	16	_______	7062
Annual_Income	100000	18940	36585.12	16
Num_of_Loan	100000	434	3	14386
Type_of_Loan	88592	6260	Not Specified	1408
Num_of_Delayed_Payment	92998	749	19	5327
Changed_Credit_Limit	100000	3635	_	2091
Credit_Mix	100000	4	Standard	36479
Outstanding_Debt	100000	13178	1360.45	24
Credit_History_Age	90970	404	15 Years and 11 Months	446
Payment_of_Min_Amount	100000	3	Yes	52326
Amount_invested_monthly	95521	91049	__10000__	4305
Payment_Behaviour	100000	7	Low_spent_Small_value_payments	25513
Monthly_Balance	98800	98790	__-333333333333333333333333333__	9

Observations:

- **Customer ID:** Has 12500 unique values, indicating data for 12500 customers.

- **Month:** Only has 8 unique values. Further analysis is needed to determine which months are present.
- **Age:** Has 1788 unique values, which seems unusual as the general age range is typically 0-100.
- **SSN:** Has 12501 unique values, while Customer ID has only 12500 unique values. This suggests a potential issue with incorrect SSN entry for one customer, as the same person cannot have multiple SSNs.
- **SSN Garbage Value:** The most frequently occurring SSN appears to be a garbage value.
- **Data Cleaning:** The dataset requires data cleaning due to the presence of underscores in a few columns.

In [7]:
```
# Checking number of empty values in each column
df.isna().sum()/len(df)*100
```

Out[7]:
```
ID                           0.000
Customer_ID                  0.000
Month                        0.000
Name                         9.985
Age                          0.000
SSN                          0.000
Occupation                   0.000
Annual_Income                0.000
Monthly_Inhand_Salary       15.002
Num_Bank_Accounts            0.000
Num_Credit_Card              0.000
Interest_Rate                0.000
Num_of_Loan                  0.000
Type_of_Loan                11.408
Delay_from_due_date          0.000
Num_of_Delayed_Payment       7.002
Changed_Credit_Limit         0.000
Num_Credit_Inquiries         1.965
Credit_Mix                   0.000
Outstanding_Debt             0.000
Credit_Utilization_Ratio     0.000
Credit_History_Age           9.030
Payment_of_Min_Amount        0.000
Total_EMI_per_month          0.000
Amount_invested_monthly      4.479
Payment_Behaviour            0.000
Monthly_Balance              1.200
dtype: float64
```

Helper Functions

In [8]:
```
def get_column_details(df, column):
    print("Details of", column, "column")

    # DataType of column
    print("\nDataType: ", df[column].dtype)

    # Check if null values are present
    count_null = df[column].isnull().sum()
    if count_null == 0:
        print("\nThere are no null values")
    elif count_null > 0:
        print("\nThere are ", count_null, " null values")

    # Get Number of Unique Values
    print("\nNumber of Unique Values: ", df[column].nunique())

    # Get Distribution of Column
    print("\nDistribution of column:\n")
    print(df[column].value_counts())
```

In [9]:
```
def fill_missing_with_group_mode(df, groupby, column):
    print("\nNo. of missing values before filling with group mode:", df[column].isnull().sum())

    # Assign None to np.NaN
    if df[column].isin([None]).sum():
        df[column][df[column].isin([None])] = np.NaN

    # Fill with local mode
    mode_per_group = df.groupby(groupby)[column].transform(lambda x: x.mode().iat[0])
    df[column] = df[column].fillna(mode_per_group)

    print("\nNo. of missing values after filling with group mode:", df[column].isnull().sum())
```

In [10]:
```
def clean_categorical_field(df, groupby, column, replace_value=None):
    print("\n")
    print("\nCleaning steps ")
```

```python
    # Replace with np.nan
    if replace_value != None:
        df[column] = df[column].replace(replace_value, np.nan)
        print(f"\nGarbage value ({replace_value}) is replaced with np.nan")

    # For each Customer ID, assign same value for the column
    fill_missing_with_group_mode(df, groupby, column)
```

Objective Variables

In [11]:
```python
get_column_details(df, 'ID')
```

```
Details of ID column

DataType:  object

There are no null values

Number of Unique Values:  100000

Distribution of column:

ID
0x1602     1
0x19c88    1
0x19caa    1
0x19ca5    1
0x19ca4    1
          ..
0xd94d     1
0xd94c     1
0xd94b     1
0xd94a     1
0x25fed    1
Name: count, Length: 100000, dtype: int64
```

In [12]:
```python
# customer-id
```

```python
#Get Details
get_column_details(df, 'Customer_ID')
```

```
Details of Customer_ID column

DataType:  object

There are no null values

Number of Unique Values:  12500

Distribution of column:

Customer_ID
CUS_0xd40     8
CUS_0x9bf4    8
CUS_0x5ae3    8
CUS_0xbe9a    8
CUS_0x4874    8
             ..
CUS_0x2eb4    8
CUS_0x7863    8
CUS_0x9d89    8
CUS_0xc045    8
CUS_0x942c    8
Name: count, Length: 12500, dtype: int64
```

In [13]:

```python
# month

#Get Details
get_column_details(df, 'Month')
```

```
Details of Month column

DataType:  object

There are no null values

Number of Unique Values:  8

Distribution of column:

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

In [14]:

```python
get_column_details(df, 'Name')
```

```
Details of Name column

DataType:  object

There are  9985  null values

Number of Unique Values:  10139

Distribution of column:

Name
Langep           44
Stevex           44
Vaughanl         39
Jessicad         39
Raymondr         38
                 ..
Alina Selyukhg    4
Habboushg         4
Mortimerq         4
Ronaldf           4
Timothyl          3
Name: count, Length: 10139, dtype: int64
```

In [15]:

```
#Cleaning
column_name = 'Name'
group_by = 'Customer_ID'

clean_categorical_field(df,group_by, column_name)
```

```
Cleaning steps

No. of missing values before filling with group mode: 9985

No. of missing values after filling with group mode: 0
```

In [16]:

```
#Get Details
get_column_details(df, 'SSN')
```

```
Details of SSN column

DataType:  object

There are no null values

Number of Unique Values:  12501

Distribution of column:

SSN
#F%$D@*&8      5572
078-73-5990       8
486-78-3816       8
750-67-7525       8
903-50-0305       8
               ... 
856-06-6147       4
753-72-2651       4
331-28-1921       4
604-62-6133       4
286-44-9634       4
Name: count, Length: 12501, dtype: int64
```

In [17]:

```python
column_name = 'SSN'
group_by = 'Customer_ID'
garbage_value = '#F%$D@*&8'

#Cleaning
clean_categorical_field(df,group_by,column_name,garbage_value)
```

```
Cleaning steps

Garbage value (#F%$D@*&8) is replaced with np.nan

No. of missing values before filling with group mode: 5572

No. of missing values after filling with group mode: 0
```

In [18]:
```python
# occupation

get_column_details(df, 'Occupation')
```

```
Details of Occupation column

DataType:  object

There are no null values

Number of Unique Values:  16

Distribution of column:

Occupation
_______          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: count, dtype: int64
```

In [19]:
```python
column_name = 'Occupation'
group_by = 'Customer_ID'
garbage_value = '_______'

#Cleaning
clean_categorical_field(df,group_by,column_name,garbage_value)
```

```
Cleaning steps

Garbage value (_______) is replaced with np.nan

No. of missing values before filling with group mode: 7062

No. of missing values after filling with group mode: 0
```

In [20]:
```
#Get Details
get_column_details(df,'Type_of_Loan')
```

```
Details of Type_of_Loan column

DataType:  object

There are  11408  null values

Number of Unique Values:  6260

Distribution of column:

Type_of_Loan
Not Specified
1408
Credit-Builder Loan
1280
Personal Loan
1272
Debt Consolidation Loan
1264
Student Loan
1240

...
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 Lo
an       8
Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan
8
Name: count, Length: 6260, dtype: int64
```

In [21]:
```
df['Type_of_Loan'].replace([np.NaN], 'Not Specified', inplace=True)
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\859618787.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['Type_of_Loan'].replace([np.NaN], 'Not Specified', inplace=True)
```

In [22]:
```
df['Type_of_Loan'] = df['Type_of_Loan'].apply(lambda x: x.lower().replace('and ', '').replace(', ', ',').strip() if pd.notna(x)
```

In [23]:
```
get_column_details(df,'Type_of_Loan')
```

```
Details of Type_of_Loan column

DataType:  object

There are no null values

Number of Unique Values:  6260

Distribution of column:

Type_of_Loan
not specified                                                                                                1281
6
credit-builder loan                                                                                           128
0
personal loan                                                                                                 127
2
debt consolidation loan                                                                                       126
4
student loan                                                                                                  124
0
                                                                                                              ...
not specified,mortgage loan,auto loan,payday loan
8
payday loan,mortgage loan,debt consolidation loan,student loan
8
debt consolidation loan,auto loan,personal loan,debt consolidation loan,student loan,credit-builder loan
8
student loan,auto loan,student loan,credit-builder loan,home equity loan,debt consolidation loan,debt consolidation loan
8
personal loan,auto loan,mortgage loan,student loan,student loan
8
Name: count, Length: 6260, dtype: int64
```

In [24]:

```python
# 3. Split the column values by commas to separate multiple loan types
df['Type_of_Loan_Split'] = df['Type_of_Loan'].str.split(',')

# 4. Explode the list of loan types into separate rows
exploded_loans = df.explode('Type_of_Loan_Split')

# 5. Get the unique loan types and map to numeric IDs
```

```python
unique_loan_types = exploded_loans['Type_of_Loan_Split'].unique()

# Create a dictionary mapping each unique loan type to a numeric ID
loan_type_dict = {idx: loan_type for idx, loan_type in enumerate(sorted(unique_loan_types))}

# Display the result
loan_type_dict
```

Out[24]:

```
{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'}
```

In [25]:

```python
exploded_loans
```

Out[25]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Nu
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	
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	
...	...	...	...	...	...	...	...	...	...	...	
99997	0x25feb	CUS_0x942c	June	Nicks	25	078-73-5990	Mechanic	39628.99	3359.415833	4	
99998	0x25fec	CUS_0x942c	July	Nicks	25	078-73-5990	Mechanic	39628.99	3359.415833	4	

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Nu
99998	0x25fec	CUS_0x942c	July	Nicks	25	078-73-5990	Mechanic	39628.99	3359.415833	4	
99999	0x25fed	CUS_0x942c	August	Nicks	25	078-73-5990	Mechanic	39628.99_	3359.415833	4	
99999	0x25fed	CUS_0x942c	August	Nicks	25	078-73-5990	Mechanic	39628.99_	3359.415833	4	

364696 rows × 28 columns

Summary:

- **Type of Loan:** There are 6260 unique values and null values are present.
- **Null Value Handling:** All null values in the `Type_of_Loan` column were replaced with "Not Specified."
- **Text Cleaning:** The text was converted to lowercase, "and" was removed, comma spacing was adjusted, and leading/trailing whitespace was stripped.
- **Loan Types:** There are 9 distinct loan types:
  1. auto loan
  2. credit-builder loan
  3. debt consolidation loan
  4. home equity loan
  5. mortgage loan
  6. not specified
  7. payday loan
  8. personal loan
  9. student loan

In [26]:

```
#credit mix
get_column_details(df, 'Credit_Mix')
```

```
Details of Credit_Mix column

DataType:  object

There are no null values

Number of Unique Values:  4

Distribution of column:

Credit_Mix
Standard    36479
Good        24337
_           20195
Bad         18989
Name: count, dtype: int64
```

In [27]:
```
column_name = 'Credit_Mix'
group_by = 'Customer_ID'
garbage_value = '_'

#Cleaning
clean_categorical_field(df,group_by,column_name, garbage_value)
```

```
Cleaning steps

Garbage value (_) is replaced with np.nan

No. of missing values before filling with group mode: 20195

No. of missing values after filling with group mode: 0
```

In [28]:
```
# payment of min amount

get_column_details(df, 'Payment_of_Min_Amount')
```

```
Details of Payment_of_Min_Amount column

DataType:  object

There are no null values

Number of Unique Values:  3

Distribution of column:

Payment_of_Min_Amount
Yes    52326
No     35667
NM     12007
Name: count, dtype: int64
```

In [29]:

```python
# payment behaviour

get_column_details(df, 'Payment_Behaviour')
```

```
Details of Payment_Behaviour column

DataType:  object

There are no null values

Number of Unique Values:  7

Distribution of column:

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
```

In [30]:
```python
column_name = 'Payment_Behaviour'
group_by = 'Customer_ID'
garbage_value = '!@9#%8'

#Cleaning
clean_categorical_field(df, group_by, column_name, garbage_value)
```

```
Cleaning steps

Garbage value (!@9#%8) is replaced with np.nan

No. of missing values before filling with group mode: 7600

No. of missing values after filling with group mode: 0
```

In [31]:
```python
def get_group_min_max(df, groupby, column):
    """
    This function calculates the minimum and maximum of the mode values within each group for a specific column.
    It groups the DataFrame by a specified column and returns the minimum and maximum values of the most frequent values.

    Parameters:
        df (DataFrame): The DataFrame containing the data.
        groupby (str): The column name to group the data by.
        column (str): The column name for which we are calculating the min/max of the mode.

    Returns:
        tuple: Minimum and maximum mode values from the grouped data.
    """

    # Drop NaN values in the specified column
    df_clean = df.dropna(subset=[column])

    # Compute the mode for each group
    group_modes = df_clean.groupby(groupby)[column].agg(
        lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan
    )

    # Find the minimum and maximum of these modes
    min_mode = group_modes.min()
```

```
    max_mode = group_modes.max()

    return min_mode, max_mode
```

In [32]:
```
def fix_inconsistent_values(df, groupby, column):
    """
    This function handles outliers and fixes inconsistent values in the DataFrame using group-wise median and MAD.
    It ensures that values in a specified column remain within reasonable bounds and fills NaN or inconsistent values
    with the median of their respective group or the global median.

    Parameters:
        df (DataFrame): The DataFrame containing the data.
        groupby (str): The column name to group the data by.
        column (str): The column for which we want to handle outliers and inconsistent values.

    Returns:
        None: The function modifies the DataFrame in place.
    """

    # Step 1: Print existing minimum and maximum values in the entire column before cleaning
    print("\nExisting Min, Max Values:")
    print(df[column].min())
    print(df[column].max())

    # Step 2: Compute group-wise median and MAD using transform for efficiency
    group_median = df.groupby(groupby)[column].transform('median')
    group_mad = df.groupby(groupby)[column].transform(
        lambda x: np.median(np.abs(x - np.median(x)))
    )
    # Handle cases where MAD is zero (e.g., all values are the same in the group)
    group_mad.replace(0, np.nan, inplace=True)

    # Step 3: Define acceptable range as median ± 3 MAD
    # Since MAD is a measure of dispersion similar to standard deviation but more robust
    # Multiplying by 1.4826 scales MAD to be comparable to standard deviation for normal distribution
    factor = 1.4826  # Scaling factor to make MAD comparable to standard deviation
    lower_bound = group_median - (3 * factor * group_mad)
    upper_bound = group_median + (3 * factor * group_mad)

    # If MAD is NaN (group_mad is NaN), set lower and upper bounds to median
    lower_bound.fillna(group_median, inplace=True)
```

```python
    upper_bound.fillna(group_median, inplace=True)

    # Replace outliers and negative values with NaN
    mask_outliers = (df[column] < lower_bound) | (df[column] > upper_bound) | (df[column] < 0)
    df.loc[mask_outliers, column] = np.nan

    # Step 4: Fill NaN values with the median of the respective group
    df[column] = df.groupby(groupby)[column].transform(
    lambda x: x.fillna(x.median())
    )

    # Step 5: If there are still NaN values, fill them with the global median of the column
    if df[column].isnull().any():
        global_median = df[column].median()
        df[column].fillna(global_median, inplace=True)

    # Step 6: Print the new minimum and maximum values after cleaning
    print("\nAfter Cleaning Min, Max Values:")
    print(df[column].min())
    print(df[column].max())

    # Step 7: Print the number of unique values and null values after cleaning
    print("\nNumber of unique values after cleaning:", df[column].nunique())
    print("Number of null values after cleaning:", df[column].isnull().sum())
```

In [33]:
```python
def clean_numerical_field(df, groupby, column, strip=None, datatype=None, replace_value=None):
    """
    Cleans a numerical field in a DataFrame.

    Parameters:
        df (DataFrame): The DataFrame containing the data.
        groupby (str): The column name to group the data by.
        column (str): The column name to clean.
        strip (str, optional): Characters to strip from the column values. Defaults to None.
        datatype (str, optional): The desired data type for the column. Defaults to None.
        replace_value (object, optional): Value to replace with NaN. Defaults to None.

    Returns:
        None: The function modifies the DataFrame in place.
    """
```

```
    print("\nCleaning steps:")

    # Replace specified value with NaN
    if replace_value is not None:
        df[column] = df[column].replace(replace_value, np.nan)
        print(f"\nGarbage value {replace_value} is replaced with np.nan")

    # Remove trailing and leading special characters
    if df[column].dtype == object and strip is not None:
        df[column] = df[column].str.strip(strip)
        print(f"\nTrailing and leading {strip} are removed")

    # Change data type
    if datatype is not None:
        df[column] = df[column].astype(datatype)
        print(f"\nDatatype of {column} is changed to {datatype}")

    # Fix inconsistent values using the `fix_inconsistent_values` function
    fix_inconsistent_values(df, groupby, column)
```

In [34]:

```
def plot_distplot(df, column, user_friendly_column_name, rotation=0, bins=20):
    """
    Plots a distribution plot (histogram with KDE) for a specified column in a DataFrame.

    Parameters:
        df (DataFrame): The DataFrame containing the data.
        column (str): The name of the column to plot.
        user_friendly_column_name (str): A user-friendly name for the column (used in labels and title).
        rotation (int, optional): The rotation angle for x-axis labels. Defaults to 0.
        bins (int, optional): The number of bins for the histogram. Defaults to 20.
    """

    print("\n")
    print(f"\n{user_friendly_column_name} Distribution")

    # Set the color palette
    palette = "deep"
    sns.set_palette(palette)

    # Create the distribution plot
    sns.displot(data=df, x=column, kde=True, bins=bins)
```

```python
    # Set labels and title
    plt.xlabel(f"{user_friendly_column_name}")
    plt.ylabel('Number of Records')
    plt.title(f"{user_friendly_column_name} Distribution")

    # Rotate x-axis labels if necessary
    plt.xticks(rotation=rotation)

    plt.show()
```

In [35]:
```python
get_column_details(df, 'Age')
```

```
Details of Age column

DataType:  object

There are no null values

Number of Unique Values:  1788

Distribution of column:

Age
38      2833
28      2829
31      2806
26      2792
32      2749
        ...
471        1
1520       1
8663       1
3363       1
1342       1
Name: count, Length: 1788, dtype: int64
```

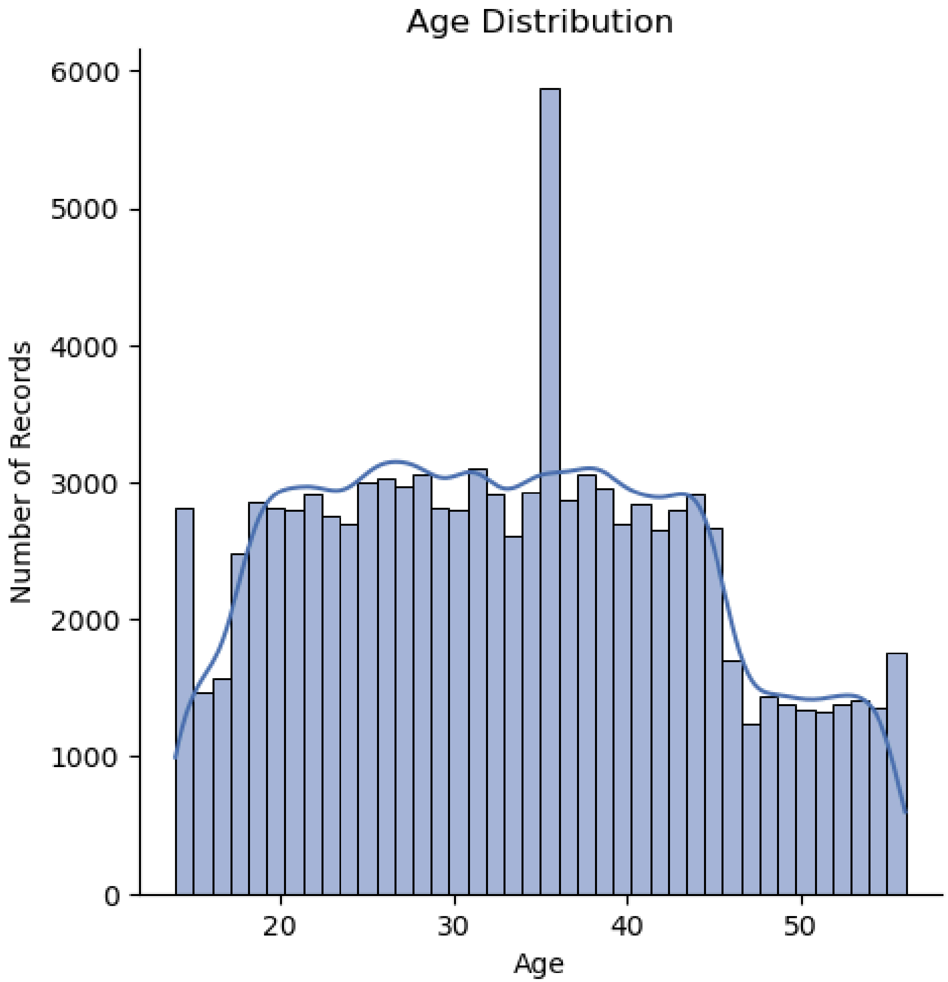
In [36]:
```python
column_name = 'Age'
group_by = 'Customer_ID'
user_friendly_name = 'Age'
```

```
#Cleaning
clean_numerical_field(df,group_by,column_name,strip='_', datatype='int')
```

```
Cleaning steps:

Trailing and leading _ are removed

Datatype of Age is changed to int

Existing Min, Max Values:
-500
8698

After Cleaning Min, Max Values:
14.0
56.0

Number of unique values after cleaning: 63
Number of null values after cleaning: 0
```

In [37]:
```
plot_distplot(df,column_name,user_friendly_name,bins=40)
```

```
Age Distribution
```



In [38]:
```
# Annual Income

get_column_details(df, 'Annual_Income')
```

```
Details of Annual_Income column

DataType:  object

There are no null values

Number of Unique Values:  18940

Distribution of column:

Annual_Income
36585.12     16
20867.67     16
17273.83     16
9141.63      15
33029.66     15
             ..
20269.93_     1
15157.25_     1
44955.64_     1
76650.12_     1
4262933       1
Name: count, Length: 18940, dtype: int64
```

In [39]:

```
column_name = 'Annual_Income'
group_by = 'Customer_ID'
user_friendly_name = 'Annual Income'

#Cleaning
clean_numerical_field(df,group_by,column_name, strip='_', datatype='float')

#Plot Graph
plot_distplot(df, column_name, user_friendly_name, bins=40)
```

Cleaning steps:

Trailing and leading _ are removed

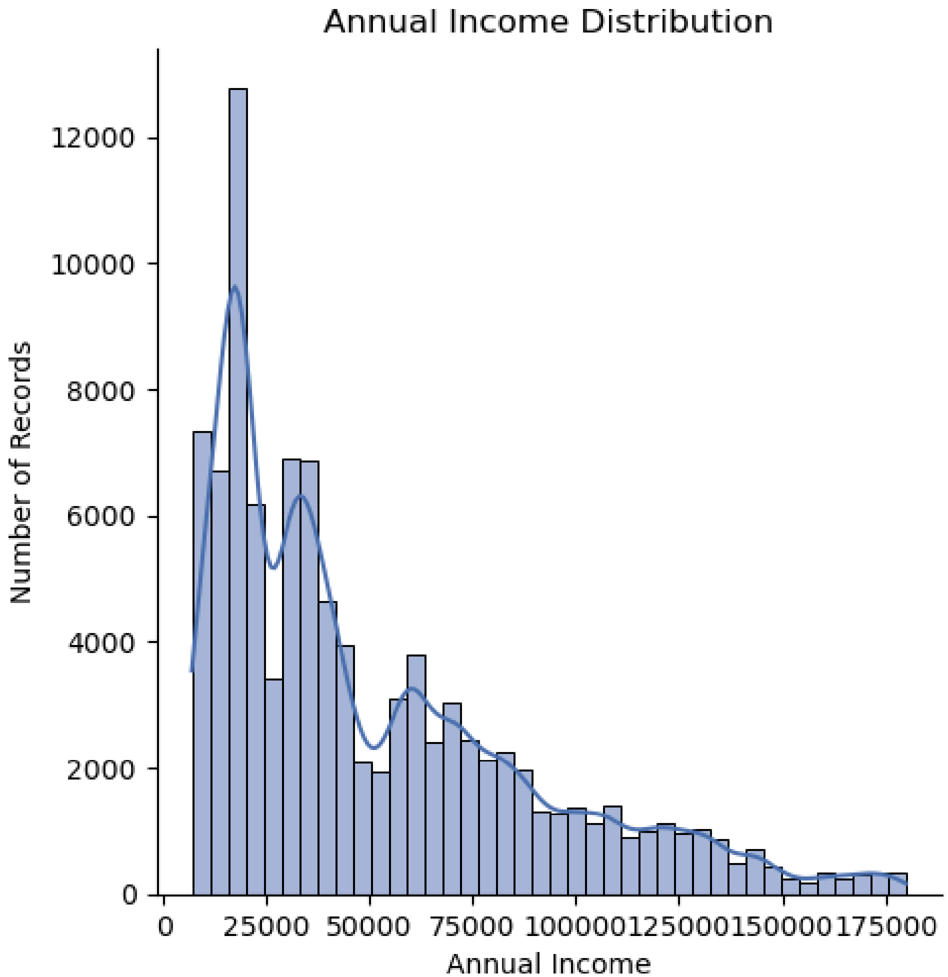
Datatype of Annual_Income is changed to float

Existing Min, Max Values:
7005.93
24198062.0

After Cleaning Min, Max Values:
7005.93
179987.28

Number of unique values after cleaning: 12489
Number of null values after cleaning: 0

Annual Income Distribution



In [40]:

```
# Monthly Inhand Salary

# Get Details
get_column_details(df, 'Monthly_Inhand_Salary')
```

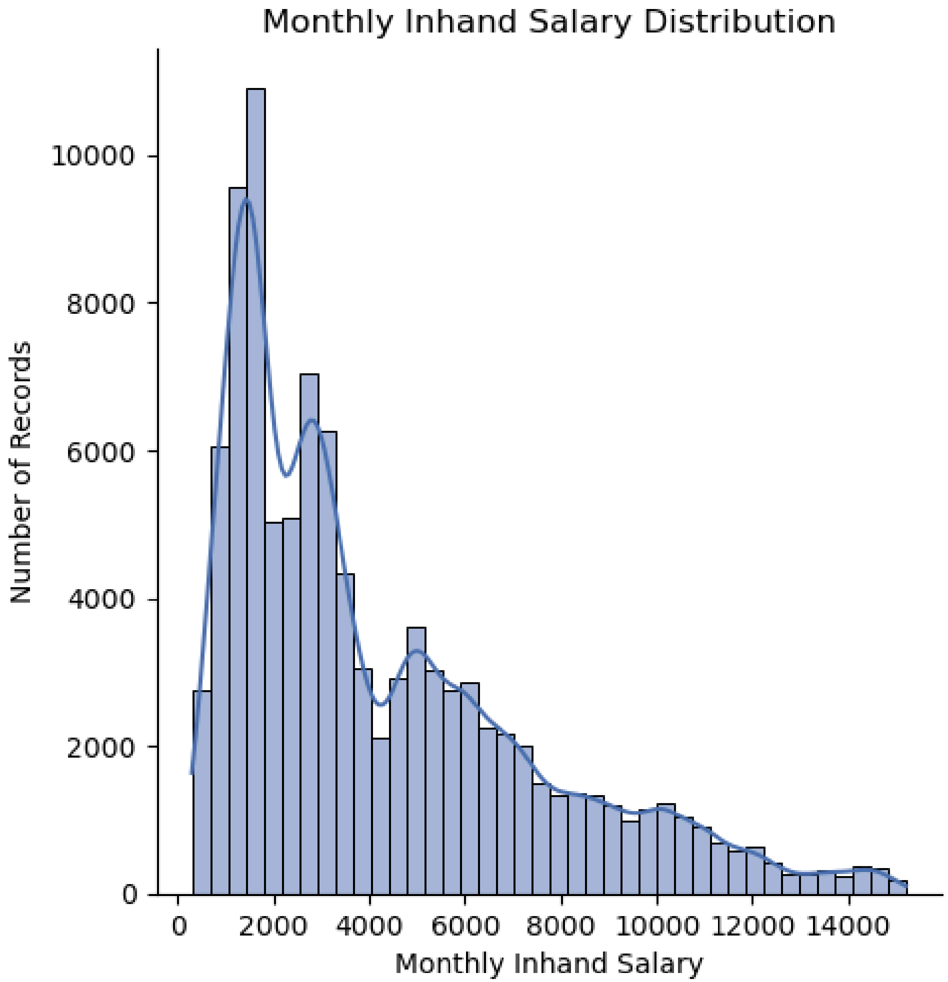
```
Details of Monthly_Inhand_Salary column

DataType:  float64

There are  15002  null values

Number of Unique Values:  13235

Distribution of column:

Monthly_Inhand_Salary
6769.130000    15
6358.956667    15
2295.058333    15
6082.187500    15
3080.555000    14
               ..
1087.546445     1
3189.212103     1
5640.117744     1
7727.560450     1
2443.654131     1
Name: count, Length: 13235, dtype: int64
```

In [41]:

```python
column_name = 'Monthly_Inhand_Salary'
group_by = 'Customer_ID'
user_friendly_name = 'Monthly Inhand Salary'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, bins=40)
```

```
Cleaning steps:

Existing Min, Max Values:
303.6454167
15204.63333
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
After Cleaning Min, Max Values:
303.6454167
15204.63333

Number of unique values after cleaning: 12487
Number of null values after cleaning: 0



Monthly Inhand Salary Distribution
```



In [42]:
```
get_column_details(df, 'Num_Bank_Accounts')
```

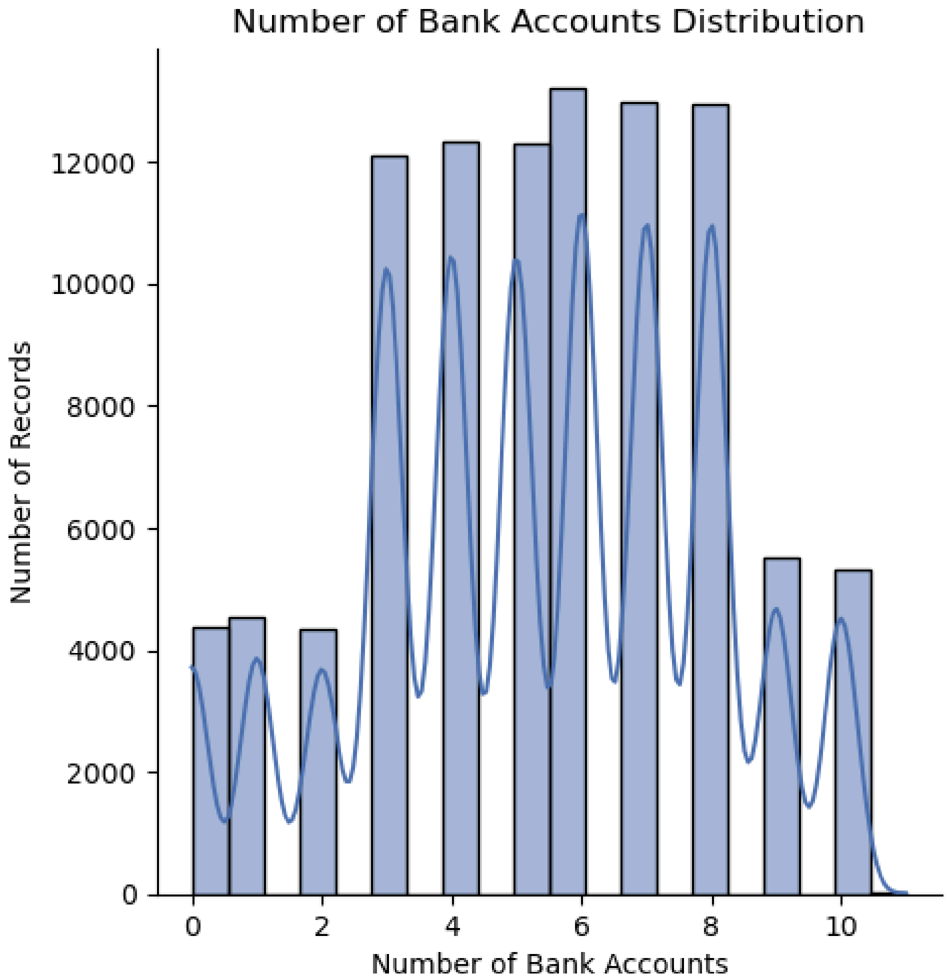
```
Details of Num_Bank_Accounts column

DataType:  int64

There are no null values

Number of Unique Values:  943

Distribution of column:

Num_Bank_Accounts
6       13001
7       12823
8       12765
4       12186
5       12118
        ...
1626        1
1470        1
887         1
211         1
697         1
Name: count, Length: 943, dtype: int64
```

In [43]:
```python
column_name = 'Num_Bank_Accounts'
group_by = 'Customer_ID'
user_friendly_name = 'Number of Bank Accounts'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name)
```

```
Cleaning steps:

Existing Min, Max Values:
-1
1798
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
After Cleaning Min, Max Values:
0.0
11.0

Number of unique values after cleaning: 12
Number of null values after cleaning: 0
```

Number of Bank Accounts Distribution



In [44]:
```
# Get Details
get_column_details(df, 'Num_Credit_Card')
```

```
Details of Num_Credit_Card column

DataType:  int64

There are no null values

Number of Unique Values:  1179

Distribution of column:

Num_Credit_Card
5       18459
7       16615
6       16559
4       14030
3       13277
        ...
791         1
1118        1
657         1
640         1
679         1
Name: count, Length: 1179, dtype: int64
```

In [45]:

```python
column_name = 'Num_Credit_Card'
group_by = 'Customer_ID'
user_friendly_name = 'Number of Credit Cards'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name)
```

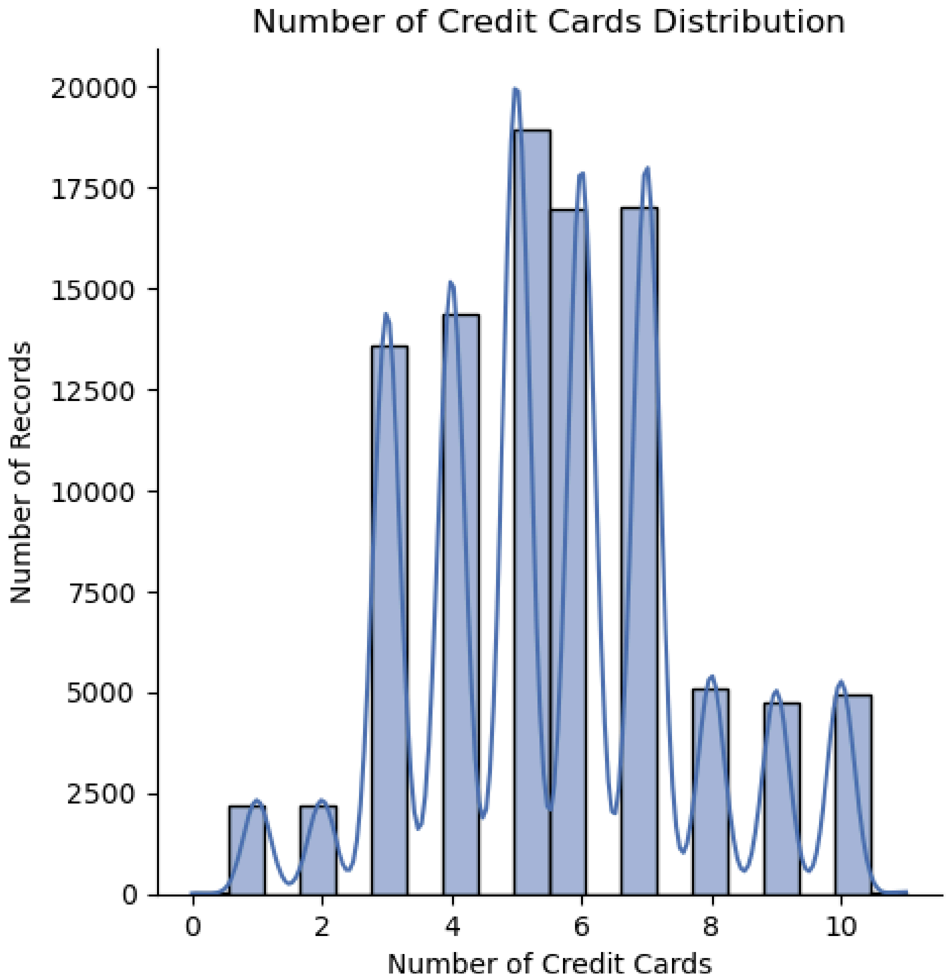
Cleaning steps:

Existing Min, Max Values:
0
1499

After Cleaning Min, Max Values:
0.0
11.0

Number of unique values after cleaning: 12
Number of null values after cleaning: 0

Number of Credit Cards Distribution



In [46]:
```
get_column_details(df, 'Interest_Rate')
```

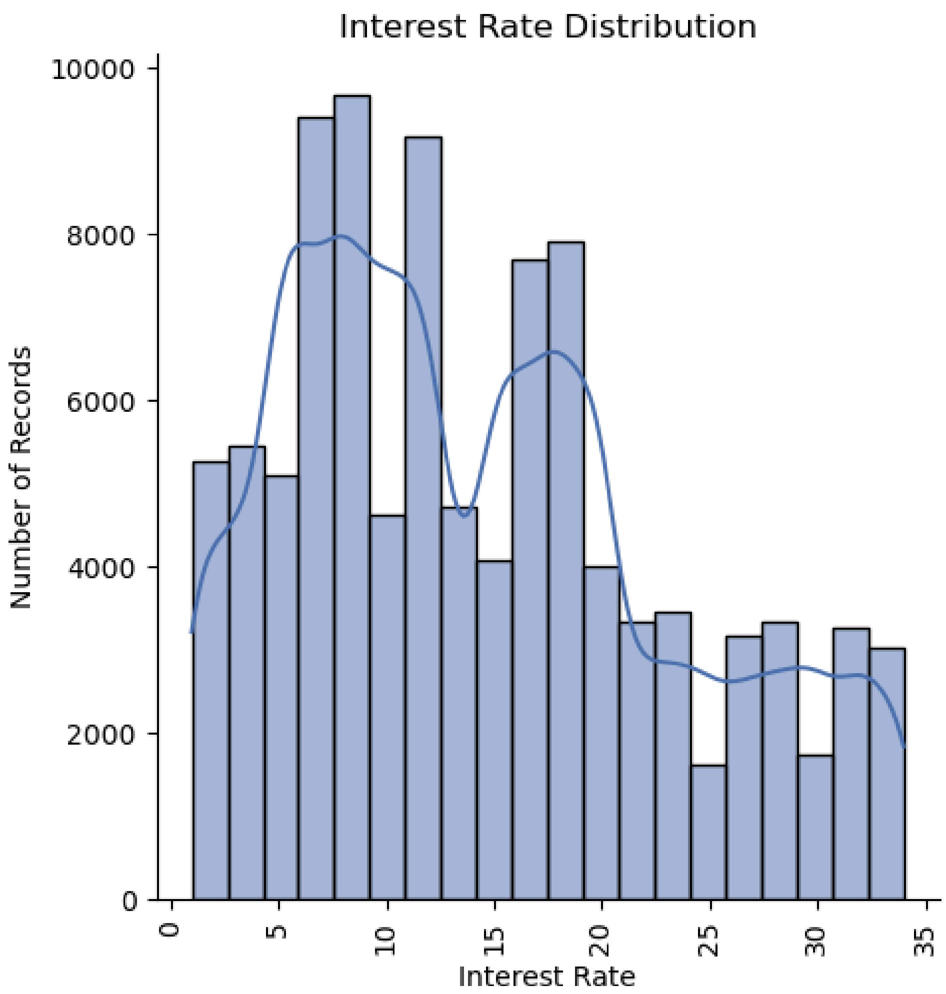
```
Details of Interest_Rate column

DataType:  int64

There are no null values

Number of Unique Values:  1750

Distribution of column:

Interest_Rate
8       5012
5       4979
6       4721
12      4540
10      4540
        ...
4995       1
1899       1
2120       1
5762       1
5729       1
Name: count, Length: 1750, dtype: int64
```

In [47]:

```python
column_name = 'Interest_Rate'
group_by = 'Customer_ID'
user_friendly_name = 'Interest Rate'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, rotation=90)
```

```
Cleaning steps:

Existing Min, Max Values:
1
5797

After Cleaning Min, Max Values:
1.0
34.0

Number of unique values after cleaning: 34
Number of null values after cleaning: 0
```

Interest Rate Distribution



In [48]:
```
get_column_details(df, 'Delay_from_due_date')
```

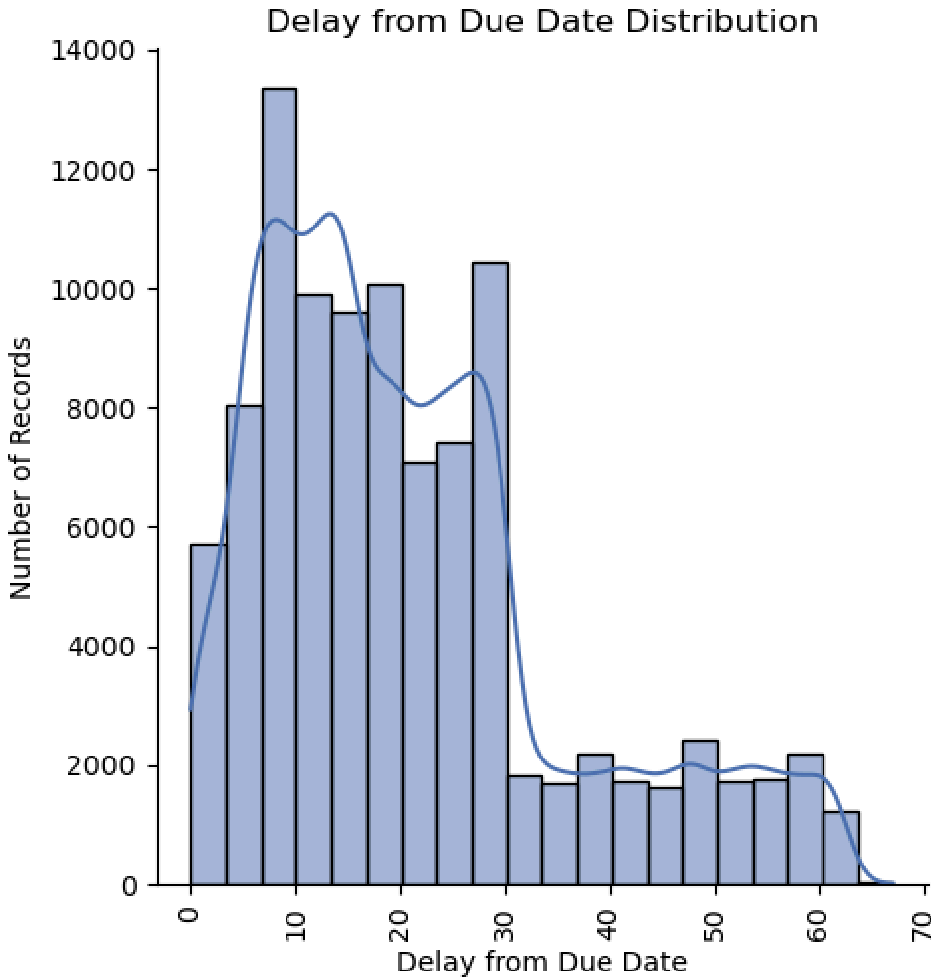
```
Details of Delay_from_due_date column

DataType:  int64

There are no null values

Number of Unique Values:  73

Distribution of column:

Delay_from_due_date
 15    3596
 13    3424
 8     3324
 14    3313
 10    3281
 7     3234
 9     3233
 11    3182
 12    3141
 6     3137
 5     3042
 19    2638
 18    2637
 27    2623
 16    2566
 24    2533
 17    2524
 25    2506
 20    2489
 21    2411
 28    2397
 23    2387
 26    2386
 29    2383
 22    2334
 30    2309
 4     1722
 3     1686
 2     1342
 1     1326
```

0	1195
31	802
33	791
32	787
34	675
47	654
48	628
52	625
54	624
42	623
35	614
44	602
36	594
38	592
41	586
53	585
50	576
40	572
55	560
56	555
58	553
57	552
62	545
49	538
45	536
51	535
60	533
39	525
61	514
59	507
43	502
37	490
46	490
-1	210
-2	168
-3	118
63	69
64	64
-4	62
65	56
-5	33

```
66      32
67      22
Name: count, dtype: int64
```

In [49]:
```python
column_name = 'Delay_from_due_date'
group_by = 'Customer_ID'
user_friendly_name = 'Delay from Due Date'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, rotation=90)
```

```
Cleaning steps:

Existing Min, Max Values:
-5
67

After Cleaning Min, Max Values:
0.0
67.0

Number of unique values after cleaning: 85
Number of null values after cleaning: 0



Delay from Due Date Distribution
```




```
In [50]: get_column_details(df, 'Num_of_Delayed_Payment')
```

```
Details of Num_of_Delayed_Payment column

DataType:  object

There are  7002  null values

Number of Unique Values:  749

Distribution of column:

Num_of_Delayed_Payment
19      5327
17      5261
16      5173
10      5153
18      5083
        ...
848_       1
4134       1
1530       1
1502       1
2047       1
Name: count, Length: 749, dtype: int64
```

In [51]:

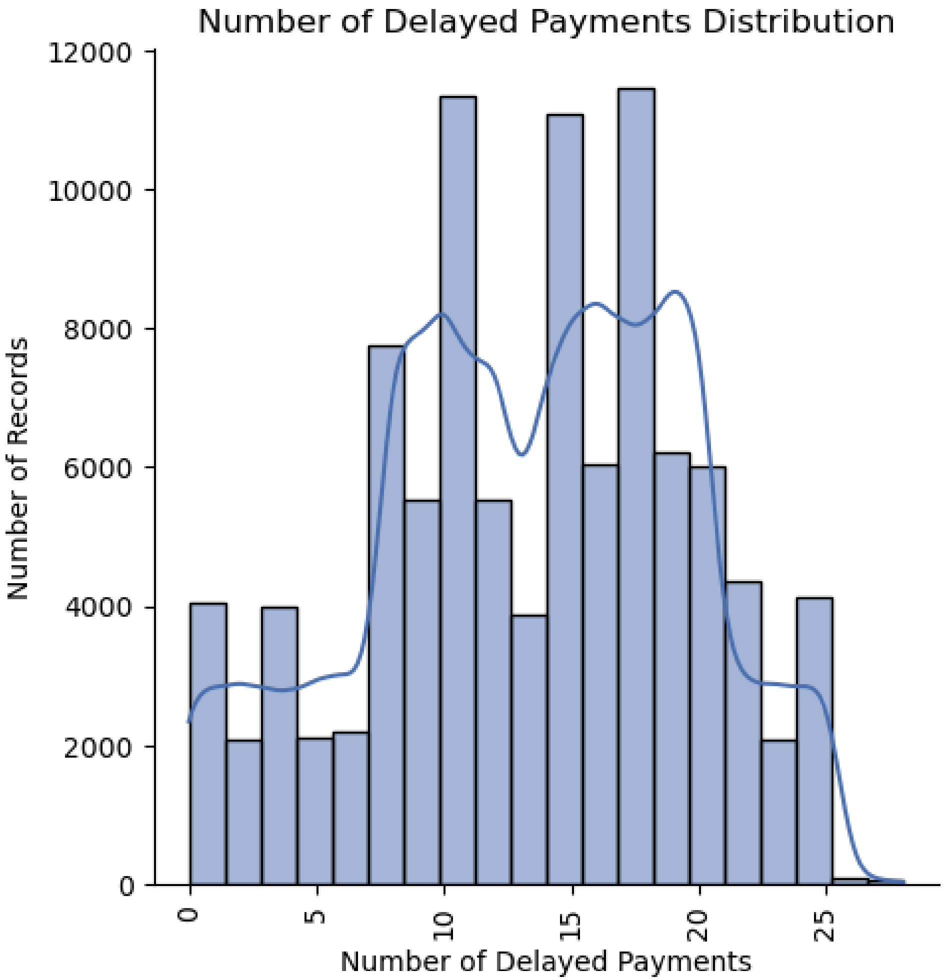
```python
column_name = 'Num_of_Delayed_Payment'
group_by = 'Customer_ID'
user_friendly_name = 'Number of Delayed Payments'

# Cleaning
clean_numerical_field(df, group_by, column_name, strip='_', datatype='float')

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, rotation=90)
```

```
Cleaning steps:

Trailing and leading _ are removed

Datatype of Num_of_Delayed_Payment is changed to float

Existing Min, Max Values:
-3.0
4397.0
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
```

```
After Cleaning Min, Max Values:
0.0
28.0

Number of unique values after cleaning: 47
Number of null values after cleaning: 0
```

Number of Delayed Payments Distribution



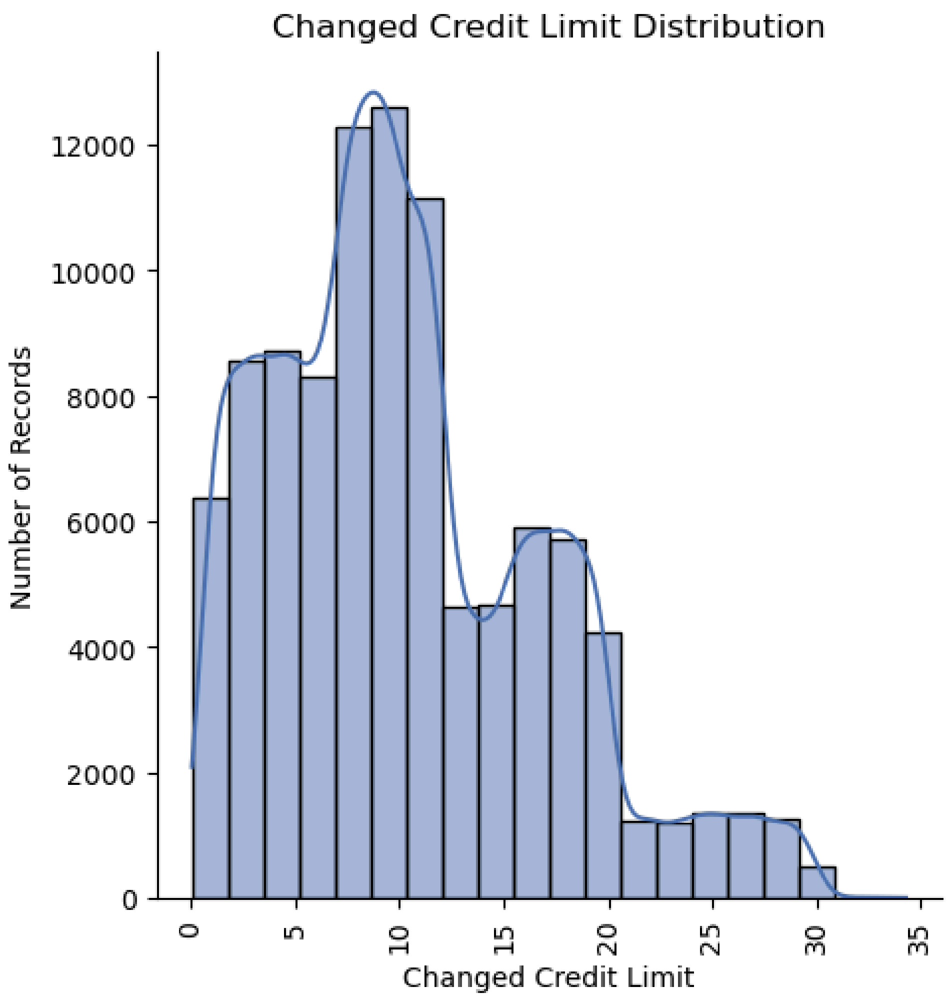
In [52]:
```
get_column_details(df, 'Changed_Credit_Limit')
```

```
Details of Changed_Credit_Limit column

DataType:  object

There are no null values

Number of Unique Values:  3635

Distribution of column:

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

In [53]:

```
column_name = 'Changed_Credit_Limit'
group_by = 'Customer_ID'
user_friendly_name = 'Changed Credit Limit'

# Cleaning
clean_numerical_field(df, group_by, column_name, strip='_', datatype='float', replace_value='_')

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, rotation=90)
```

```
Cleaning steps:

Garbage value _ is replaced with np.nan

Trailing and leading _ are removed

Datatype of Changed_Credit_Limit is changed to float

Existing Min, Max Values:
-6.49
36.97
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
After Cleaning Min, Max Values:
0.03
34.3

Number of unique values after cleaning: 2587
Number of null values after cleaning: 0



Changed Credit Limit Distribution
```



In [54]:
```
get_column_details(df, 'Num_Credit_Inquiries')
```

```
Details of Num_Credit_Inquiries column

DataType:  float64

There are  1965  null values

Number of Unique Values:  1223

Distribution of column:

Num_Credit_Inquiries
4.0       11271
3.0        8890
6.0        8111
7.0        8058
2.0        8028
          ...
1721.0        1
1750.0        1
2397.0        1
621.0         1
74.0          1
Name: count, Length: 1223, dtype: int64
```

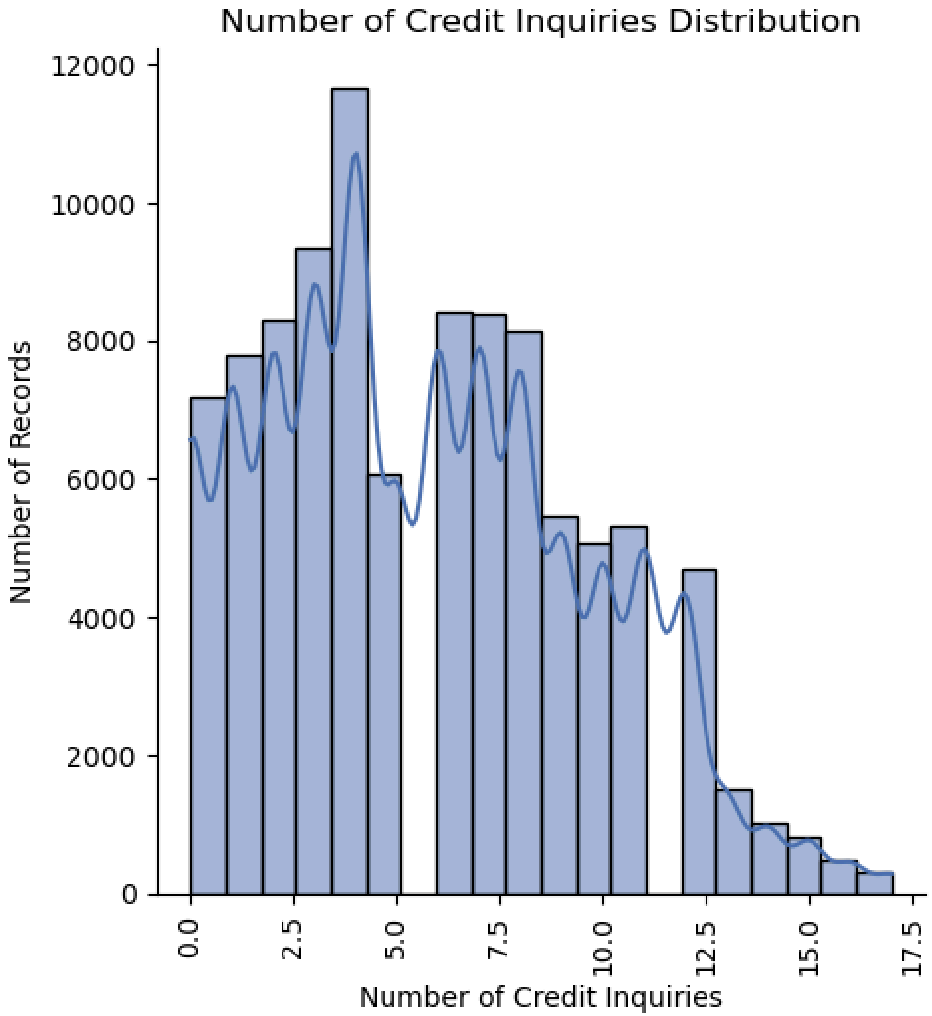
In [55]:

```
column_name = 'Num_Credit_Inquiries'
group_by = 'Customer_ID'
user_friendly_name = 'Number of Credit Inquiries'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, rotation=90)
```

```
Cleaning steps:

Existing Min, Max Values:
0.0
2597.0
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
After Cleaning Min, Max Values:
0.0
17.0

Number of unique values after cleaning: 23
Number of null values after cleaning: 0
```

Number of Credit Inquiries Distribution



In [56]:
```
get_column_details(df, 'Outstanding_Debt')
```

```
Details of Outstanding_Debt column

DataType:  object

There are no null values

Number of Unique Values:  13178

Distribution of column:

Outstanding_Debt
1360.45     24
460.46      23
1151.7      23
1109.03     23
467.7       16
            ..
245.46_      1
645.77_      1
174.79_      1
1181.13_     1
1013.53_     1
Name: count, Length: 13178, dtype: int64
```

In [57]:

```python
column_name = 'Outstanding_Debt'
group_by = 'Customer_ID'
user_friendly_name = 'Outstanding Debt'

# Cleaning
clean_numerical_field(df, group_by, column_name, strip='_', datatype='float')

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, rotation=90)
```

Cleaning steps:

Trailing and leading _ are removed

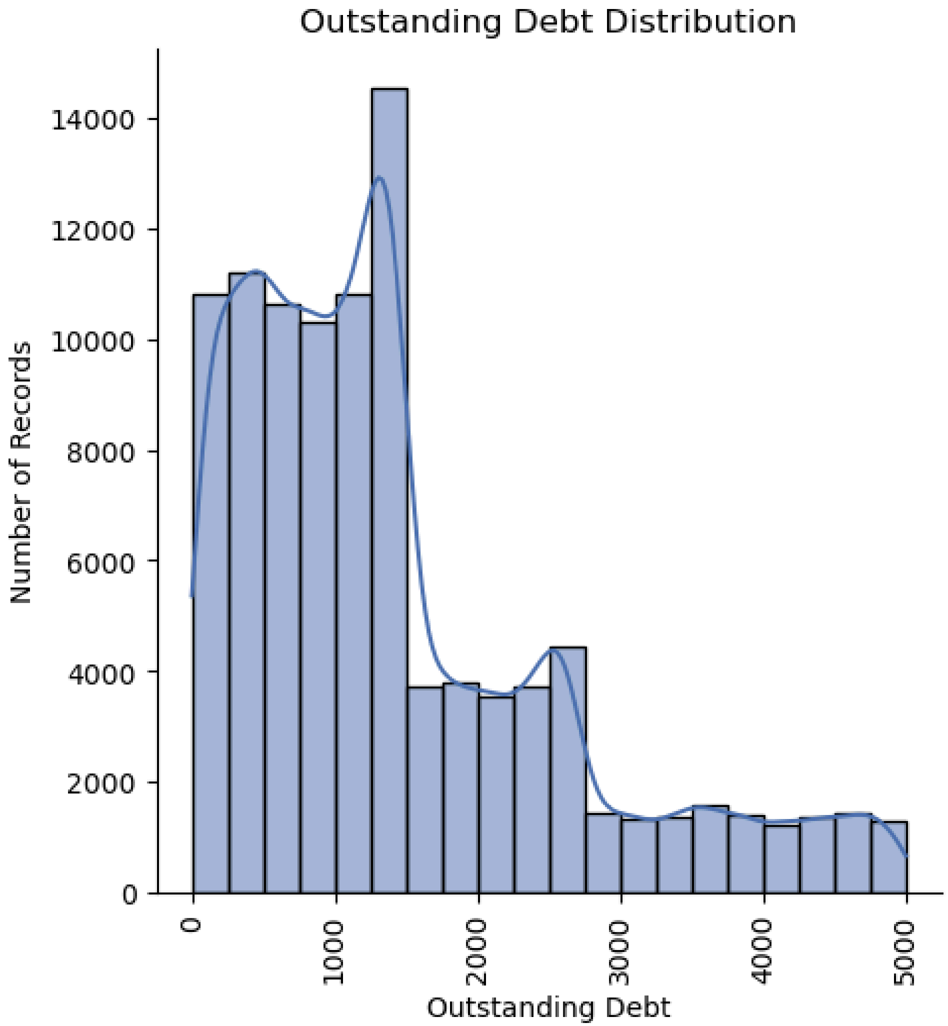
Datatype of Outstanding_Debt is changed to float

Existing Min, Max Values:
0.23
4998.07

After Cleaning Min, Max Values:
0.23
4998.07

Number of unique values after cleaning: 12203
Number of null values after cleaning: 0

Outstanding Debt Distribution

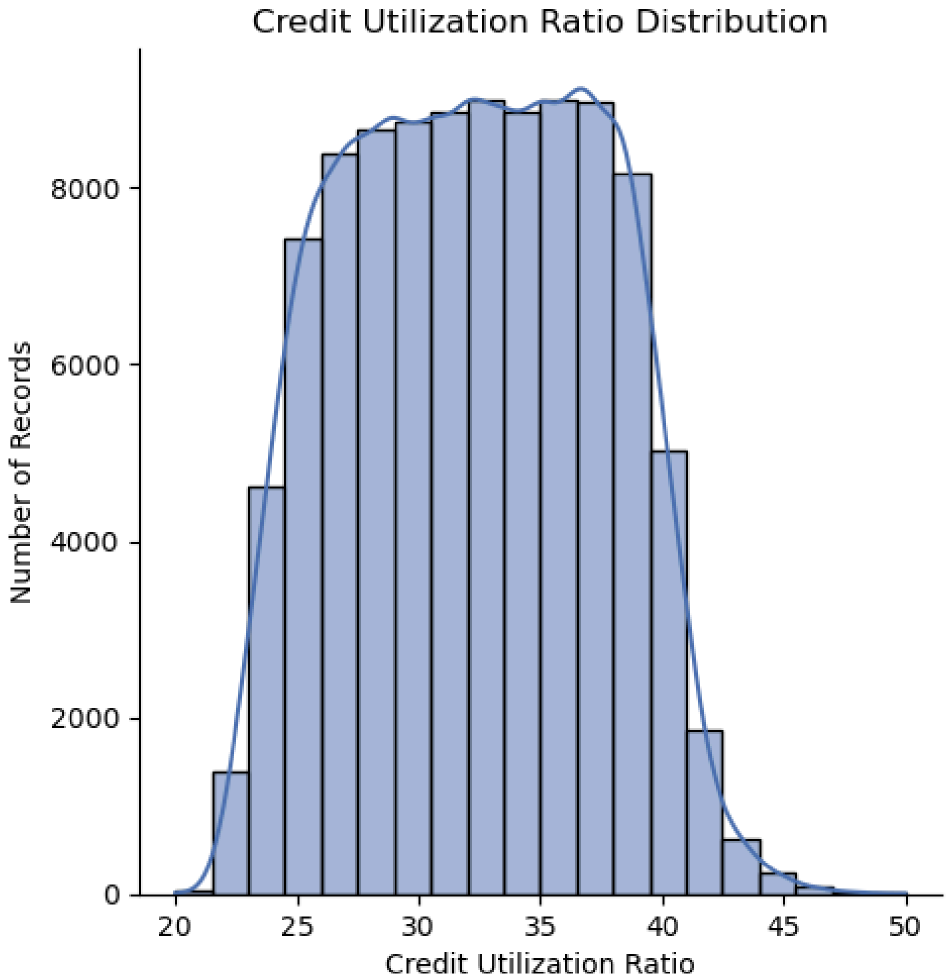


In [58]:
```
get_column_details(df, 'Credit_Utilization_Ratio')
```

```
Details of Credit_Utilization_Ratio column

DataType:  float64

There are no null values

Number of Unique Values:  99998

Distribution of column:

Credit_Utilization_Ratio
26.407909    2
33.163023    2
26.822620    1
30.462162    1
33.933755    1
            ..
38.730069    1
30.017515    1
27.279794    1
27.002436    1
34.192463    1
Name: count, Length: 99998, dtype: int64
```

In [59]:
```python
column_name = 'Credit_Utilization_Ratio'
group_by = 'Customer_ID'
user_friendly_name = 'Credit Utilization Ratio'

# No cleaning required

# Plot Graph
plot_distplot(df, column_name, user_friendly_name)
```

```
Credit Utilization Ratio Distribution
```



In [60]:
```
df['Credit_History_Age'].value_counts()
```

```
Out[60]: Credit_History_Age
15 Years and 11 Months    446
19 Years and 4 Months     445
19 Years and 5 Months     444
17 Years and 11 Months    443
19 Years and 3 Months     441
17 Years and 9 Months     438
15 Years and 10 Months    436
17 Years and 10 Months    435
15 Years and 9 Months     432
18 Years and 3 Months     428
18 Years and 4 Months     426
18 Years and 2 Months     426
19 Years and 9 Months     422
17 Years and 8 Months     419
15 Years and 8 Months     415
18 Years and 11 Months    414
16 Years and 2 Months     412
18 Years and 5 Months     410
18 Years and 10 Months    408
19 Years and 11 Months    405
19 Years and 10 Months    403
19 Years and 8 Months     403
17 Years and 3 Months     403
18 Years and 9 Months     402
17 Years and 5 Months     402
19 Years and 2 Months     401
18 Years and 8 Months     396
16 Years and 3 Months     396
16 Years and 8 Months     396
16 Years and 5 Months     396
17 Years and 2 Months     395
17 Years and 4 Months     395
16 Years and 4 Months     393
16 Years and 10 Months    386
18 Years and 0 Months     382
16 Years and 9 Months     381
16 Years and 0 Months     381
16 Years and 11 Months    373
17 Years and 7 Months     372
18 Years and 1 Months     371
```

19 Years and 6 Months	362
15 Years and 7 Months	357
16 Years and 1 Months	356
19 Years and 7 Months	355
19 Years and 0 Months	354
20 Years and 2 Months	354
18 Years and 6 Months	351
19 Years and 1 Months	349
17 Years and 6 Months	349
20 Years and 0 Months	344
15 Years and 5 Months	339
20 Years and 3 Months	337
17 Years and 1 Months	336
18 Years and 7 Months	333
16 Years and 6 Months	331
17 Years and 0 Months	331
16 Years and 7 Months	325
20 Years and 4 Months	318
15 Years and 6 Months	315
20 Years and 1 Months	309
20 Years and 5 Months	297
15 Years and 4 Months	294
6 Years and 5 Months	283
13 Years and 8 Months	283
13 Years and 10 Months	275
6 Years and 8 Months	275
6 Years and 2 Months	273
30 Years and 3 Months	272
13 Years and 5 Months	271
13 Years and 11 Months	270
15 Years and 3 Months	270
6 Years and 3 Months	269
30 Years and 8 Months	269
30 Years and 4 Months	269
13 Years and 9 Months	269
6 Years and 4 Months	268
30 Years and 2 Months	268
31 Years and 11 Months	268
29 Years and 11 Months	264
32 Years and 10 Months	264
30 Years and 10 Months	264

5 Years and 11 Months	263
23 Years and 5 Months	263
30 Years and 5 Months	263
29 Years and 10 Months	261
12 Years and 8 Months	261
9 Years and 9 Months	261
5 Years and 10 Months	258
13 Years and 4 Months	258
6 Years and 9 Months	258
12 Years and 11 Months	257
26 Years and 2 Months	257
8 Years and 10 Months	256
13 Years and 3 Months	254
32 Years and 4 Months	254
32 Years and 2 Months	254
7 Years and 11 Months	253
9 Years and 10 Months	253
30 Years and 9 Months	253
32 Years and 3 Months	253
23 Years and 9 Months	252
8 Years and 3 Months	252
8 Years and 5 Months	252
14 Years and 2 Months	252
28 Years and 11 Months	252
6 Years and 10 Months	251
23 Years and 10 Months	251
32 Years and 5 Months	251
9 Years and 8 Months	251
32 Years and 11 Months	251
13 Years and 2 Months	251
28 Years and 10 Months	250
32 Years and 9 Months	250
12 Years and 10 Months	249
30 Years and 11 Months	249
31 Years and 2 Months	249
23 Years and 11 Months	249
25 Years and 11 Months	248
21 Years and 10 Months	248
29 Years and 5 Months	248
23 Years and 3 Months	247
13 Years and 7 Months	247

6 Years and 6 Months	247
8 Years and 4 Months	247
31 Years and 3 Months	247
26 Years and 4 Months	247
24 Years and 5 Months	246
28 Years and 5 Months	246
9 Years and 5 Months	246
8 Years and 2 Months	246
22 Years and 4 Months	246
12 Years and 9 Months	246
22 Years and 10 Months	246
29 Years and 8 Months	246
20 Years and 6 Months	245
24 Years and 2 Months	245
22 Years and 5 Months	245
27 Years and 11 Months	245
21 Years and 11 Months	244
21 Years and 8 Months	244
11 Years and 9 Months	244
5 Years and 9 Months	244
9 Years and 3 Months	244
6 Years and 7 Months	244
9 Years and 4 Months	243
21 Years and 9 Months	243
20 Years and 8 Months	243
31 Years and 9 Months	243
29 Years and 4 Months	242
28 Years and 4 Months	242
22 Years and 3 Months	242
28 Years and 8 Months	242
27 Years and 4 Months	242
24 Years and 9 Months	242
26 Years and 11 Months	242
12 Years and 5 Months	242
28 Years and 2 Months	242
24 Years and 4 Months	241
23 Years and 4 Months	241
24 Years and 8 Months	241
11 Years and 4 Months	241
9 Years and 11 Months	241
11 Years and 11 Months	240

7 Years and 10 Months	240
11 Years and 5 Months	240
31 Years and 8 Months	240
29 Years and 3 Months	240
32 Years and 8 Months	240
11 Years and 10 Months	240
26 Years and 3 Months	240
24 Years and 11 Months	240
7 Years and 8 Months	240
8 Years and 11 Months	239
13 Years and 6 Months	239
29 Years and 9 Months	239
21 Years and 5 Months	238
28 Years and 3 Months	238
12 Years and 4 Months	238
31 Years and 10 Months	237
32 Years and 1 Months	237
31 Years and 5 Months	236
10 Years and 2 Months	236
24 Years and 10 Months	235
10 Years and 8 Months	235
28 Years and 9 Months	235
23 Years and 8 Months	235
21 Years and 4 Months	234
24 Years and 3 Months	234
25 Years and 2 Months	234
26 Years and 8 Months	234
25 Years and 10 Months	234
22 Years and 8 Months	233
31 Years and 4 Months	233
12 Years and 2 Months	233
27 Years and 10 Months	232
26 Years and 9 Months	232
8 Years and 9 Months	232
29 Years and 2 Months	232
11 Years and 3 Months	231
8 Years and 8 Months	231
7 Years and 9 Months	231
6 Years and 0 Months	230
11 Years and 8 Months	230
27 Years and 5 Months	230

22 Years and 11 Months	230
12 Years and 3 Months	229
11 Years and 2 Months	229
21 Years and 3 Months	228
27 Years and 2 Months	228
6 Years and 1 Months	228
25 Years and 9 Months	228
6 Years and 11 Months	228
22 Years and 9 Months	228
22 Years and 2 Months	228
14 Years and 0 Months	228
10 Years and 11 Months	227
20 Years and 9 Months	227
9 Years and 2 Months	226
26 Years and 10 Months	226
33 Years and 0 Months	226
25 Years and 3 Months	226
30 Years and 1 Months	226
27 Years and 3 Months	226
8 Years and 1 Months	225
26 Years and 5 Months	225
10 Years and 10 Months	225
14 Years and 3 Months	225
23 Years and 2 Months	224
26 Years and 0 Months	224
21 Years and 2 Months	224
10 Years and 4 Months	223
31 Years and 1 Months	223
14 Years and 1 Months	223
32 Years and 0 Months	222
8 Years and 0 Months	222
15 Years and 2 Months	222
12 Years and 7 Months	222
13 Years and 0 Months	222
20 Years and 11 Months	221
10 Years and 9 Months	221
32 Years and 6 Months	221
14 Years and 4 Months	220
5 Years and 8 Months	220
9 Years and 7 Months	219
20 Years and 10 Months	219

24 Years and 0 Months	219
27 Years and 8 Months	219
10 Years and 3 Months	219
30 Years and 6 Months	219
30 Years and 7 Months	218
10 Years and 5 Months	218
31 Years and 0 Months	217
21 Years and 6 Months	217
23 Years and 7 Months	217
29 Years and 1 Months	216
22 Years and 6 Months	216
24 Years and 1 Months	216
26 Years and 1 Months	216
29 Years and 0 Months	216
7 Years and 2 Months	215
27 Years and 9 Months	214
7 Years and 5 Months	214
9 Years and 6 Months	212
8 Years and 6 Months	212
30 Years and 0 Months	212
10 Years and 1 Months	212
7 Years and 4 Months	211
25 Years and 4 Months	211
20 Years and 7 Months	211
11 Years and 6 Months	211
28 Years and 1 Months	211
12 Years and 6 Months	210
33 Years and 1 Months	209
13 Years and 1 Months	209
22 Years and 0 Months	209
12 Years and 1 Months	208
29 Years and 7 Months	208
23 Years and 6 Months	207
33 Years and 2 Months	207
24 Years and 6 Months	207
27 Years and 1 Months	206
27 Years and 0 Months	206
14 Years and 5 Months	206
11 Years and 7 Months	206
9 Years and 1 Months	206
23 Years and 1 Months	206

28 Years and 7 Months	205
10 Years and 0 Months	205
7 Years and 3 Months	205
22 Years and 1 Months	204
31 Years and 6 Months	204
28 Years and 6 Months	204
25 Years and 0 Months	203
32 Years and 7 Months	202
31 Years and 7 Months	202
26 Years and 6 Months	202
7 Years and 0 Months	201
10 Years and 7 Months	201
9 Years and 0 Months	201
21 Years and 7 Months	201
28 Years and 0 Months	200
29 Years and 6 Months	200
11 Years and 0 Months	200
8 Years and 7 Months	200
25 Years and 8 Months	199
24 Years and 7 Months	198
25 Years and 1 Months	197
23 Years and 0 Months	197
10 Years and 6 Months	196
12 Years and 0 Months	195
7 Years and 7 Months	194
7 Years and 6 Months	194
22 Years and 7 Months	194
21 Years and 0 Months	194
21 Years and 1 Months	193
25 Years and 5 Months	193
11 Years and 1 Months	191
27 Years and 6 Months	191
14 Years and 8 Months	184
27 Years and 7 Months	183
26 Years and 7 Months	182
5 Years and 7 Months	182
25 Years and 7 Months	179
5 Years and 5 Months	177
7 Years and 1 Months	177
14 Years and 10 Months	176
14 Years and 11 Months	173

5 Years and 6 Months	171
33 Years and 3 Months	170
15 Years and 1 Months	170
25 Years and 6 Months	170
14 Years and 9 Months	169
14 Years and 6 Months	163
15 Years and 0 Months	158
5 Years and 4 Months	144
14 Years and 7 Months	141
33 Years and 4 Months	133
5 Years and 3 Months	123
5 Years and 2 Months	106
2 Years and 2 Months	97
1 Years and 10 Months	95
1 Years and 4 Months	94
1 Years and 9 Months	93
1 Years and 5 Months	91
2 Years and 3 Months	90
1 Years and 8 Months	89
33 Years and 5 Months	89
1 Years and 3 Months	87
2 Years and 1 Months	84
1 Years and 11 Months	83
2 Years and 4 Months	83
2 Years and 5 Months	82
1 Years and 7 Months	81
2 Years and 8 Months	81
0 Years and 10 Months	79
5 Years and 1 Months	77
1 Years and 2 Months	77
0 Years and 11 Months	77
4 Years and 11 Months	76
3 Years and 2 Months	76
3 Years and 4 Months	76
3 Years and 5 Months	74
1 Years and 6 Months	74
4 Years and 5 Months	73
2 Years and 6 Months	73
3 Years and 3 Months	73
2 Years and 11 Months	73
2 Years and 0 Months	72

```
1 Years and 1 Months       70
4 Years and 8 Months       70
3 Years and 6 Months       69
2 Years and 9 Months       68
0 Years and 9 Months       68
2 Years and 7 Months       68
5 Years and 0 Months       68
1 Years and 0 Months       67
2 Years and 10 Months      67
3 Years and 8 Months       66
4 Years and 3 Months       66
3 Years and 0 Months       64
3 Years and 10 Months      64
3 Years and 7 Months       64
4 Years and 4 Months       63
4 Years and 10 Months      63
3 Years and 1 Months       63
3 Years and 11 Months      62
4 Years and 9 Months       61
4 Years and 7 Months       61
4 Years and 2 Months       60
4 Years and 6 Months       60
4 Years and 1 Months       59
0 Years and 8 Months       59
3 Years and 9 Months       58
0 Years and 7 Months       52
4 Years and 0 Months       50
33 Years and 6 Months      46
0 Years and 5 Months       42
0 Years and 6 Months       41
0 Years and 4 Months       35
0 Years and 3 Months       20
0 Years and 2 Months       15
33 Years and 7 Months      14
33 Years and 8 Months      12
0 Years and 1 Months        2
Name: count, dtype: int64
```

In [61]:

```
def Month_Converter(val):
    if pd.notnull(val):
        years = int(val.split(' ')[0])
```

```python
        month = int(val.split(' ')[3])
        return (years * 12) + month
    else:
        return val

df['Credit_History_In_Months'] = df['Credit_History_Age'].apply(lambda x: Month_Converter(x)).astype(float)
```

In [62]:
```python
get_column_details(df, 'Credit_History_In_Months')
```

```
Details of Credit_History_In_Months column

DataType:  float64

There are  9030  null values

Number of Unique Values:  404

Distribution of column:

Credit_History_In_Months
191.0    446
232.0    445
233.0    444
215.0    443
231.0    441
213.0    438
190.0    436
214.0    435
189.0    432
219.0    428
220.0    426
218.0    426
237.0    422
212.0    419
188.0    415
227.0    414
194.0    412
221.0    410
226.0    408
239.0    405
238.0    403
236.0    403
207.0    403
225.0    402
209.0    402
230.0    401
224.0    396
195.0    396
200.0    396
197.0    396
```

206.0	395
208.0	395
196.0	393
202.0	386
216.0	382
201.0	381
192.0	381
203.0	373
211.0	372
217.0	371
234.0	362
187.0	357
193.0	356
235.0	355
228.0	354
242.0	354
222.0	351
229.0	349
210.0	349
240.0	344
185.0	339
243.0	337
205.0	336
223.0	333
198.0	331
204.0	331
199.0	325
244.0	318
186.0	315
241.0	309
245.0	297
184.0	294
77.0	283
164.0	283
166.0	275
80.0	275
74.0	273
363.0	272
161.0	271
167.0	270
183.0	270

75.0	269
368.0	269
364.0	269
165.0	269
76.0	268
362.0	268
383.0	268
359.0	264
394.0	264
370.0	264
71.0	263
281.0	263
365.0	263
358.0	261
152.0	261
117.0	261
70.0	258
160.0	258
81.0	258
155.0	257
314.0	257
106.0	256
159.0	254
388.0	254
386.0	254
95.0	253
118.0	253
369.0	253
387.0	253
285.0	252
99.0	252
101.0	252
170.0	252
347.0	252
82.0	251
286.0	251
389.0	251
116.0	251
395.0	251
158.0	251
346.0	250

393.0	250
154.0	249
371.0	249
374.0	249
287.0	249
311.0	248
262.0	248
353.0	248
279.0	247
163.0	247
78.0	247
100.0	247
375.0	247
316.0	247
293.0	246
341.0	246
113.0	246
98.0	246
268.0	246
153.0	246
274.0	246
356.0	246
246.0	245
290.0	245
269.0	245
335.0	245
263.0	244
260.0	244
141.0	244
69.0	244
111.0	244
79.0	244
112.0	243
261.0	243
248.0	243
381.0	243
352.0	242
340.0	242
267.0	242
344.0	242
328.0	242

297.0	242
323.0	242
149.0	242
338.0	242
292.0	241
280.0	241
296.0	241
136.0	241
119.0	241
143.0	240
94.0	240
137.0	240
380.0	240
351.0	240
392.0	240
142.0	240
315.0	240
299.0	240
92.0	240
107.0	239
162.0	239
357.0	239
257.0	238
339.0	238
148.0	238
382.0	237
385.0	237
377.0	236
122.0	236
298.0	235
128.0	235
345.0	235
284.0	235
256.0	234
291.0	234
302.0	234
320.0	234
310.0	234
272.0	233
376.0	233
146.0	233

334.0	232
321.0	232
105.0	232
350.0	232
135.0	231
104.0	231
93.0	231
72.0	230
140.0	230
329.0	230
275.0	230
147.0	229
134.0	229
255.0	228
326.0	228
73.0	228
309.0	228
83.0	228
273.0	228
266.0	228
168.0	228
131.0	227
249.0	227
110.0	226
322.0	226
396.0	226
303.0	226
361.0	226
327.0	226
97.0	225
317.0	225
130.0	225
171.0	225
278.0	224
312.0	224
254.0	224
124.0	223
373.0	223
169.0	223
384.0	222
96.0	222

182.0	222
151.0	222
156.0	222
251.0	221
129.0	221
390.0	221
172.0	220
68.0	220
115.0	219
250.0	219
288.0	219
332.0	219
123.0	219
366.0	219
367.0	218
125.0	218
372.0	217
258.0	217
283.0	217
349.0	216
270.0	216
289.0	216
313.0	216
348.0	216
86.0	215
333.0	214
89.0	214
114.0	212
102.0	212
360.0	212
121.0	212
88.0	211
304.0	211
247.0	211
138.0	211
337.0	211
150.0	210
397.0	209
157.0	209
264.0	209
145.0	208

355.0	208
282.0	207
398.0	207
294.0	207
325.0	206
324.0	206
173.0	206
139.0	206
109.0	206
277.0	206
343.0	205
120.0	205
87.0	205
265.0	204
378.0	204
342.0	204
300.0	203
391.0	202
379.0	202
318.0	202
84.0	201
127.0	201
108.0	201
259.0	201
336.0	200
354.0	200
132.0	200
103.0	200
308.0	199
295.0	198
301.0	197
276.0	197
126.0	196
144.0	195
91.0	194
90.0	194
271.0	194
252.0	194
253.0	193
305.0	193
133.0	191

330.0	191
176.0	184
331.0	183
319.0	182
67.0	182
307.0	179
65.0	177
85.0	177
178.0	176
179.0	173
66.0	171
399.0	170
181.0	170
306.0	170
177.0	169
174.0	163
180.0	158
64.0	144
175.0	141
400.0	133
63.0	123
62.0	106
26.0	97
22.0	95
16.0	94
21.0	93
17.0	91
27.0	90
20.0	89
401.0	89
15.0	87
25.0	84
23.0	83
28.0	83
29.0	82
19.0	81
32.0	81
10.0	79
61.0	77
14.0	77
11.0	77

59.0	76
38.0	76
40.0	76
41.0	74
18.0	74
53.0	73
30.0	73
39.0	73
35.0	73
24.0	72
13.0	70
56.0	70
42.0	69
33.0	68
9.0	68
31.0	68
60.0	68
12.0	67
34.0	67
44.0	66
51.0	66
36.0	64
46.0	64
43.0	64
52.0	63
58.0	63
37.0	63
47.0	62
57.0	61
55.0	61
50.0	60
54.0	60
49.0	59
8.0	59
45.0	58
7.0	52
48.0	50
402.0	46
5.0	42
6.0	41
4.0	35

```
3.0        20
2.0        15
403.0      14
404.0      12
1.0         2
Name: count, dtype: int64
```

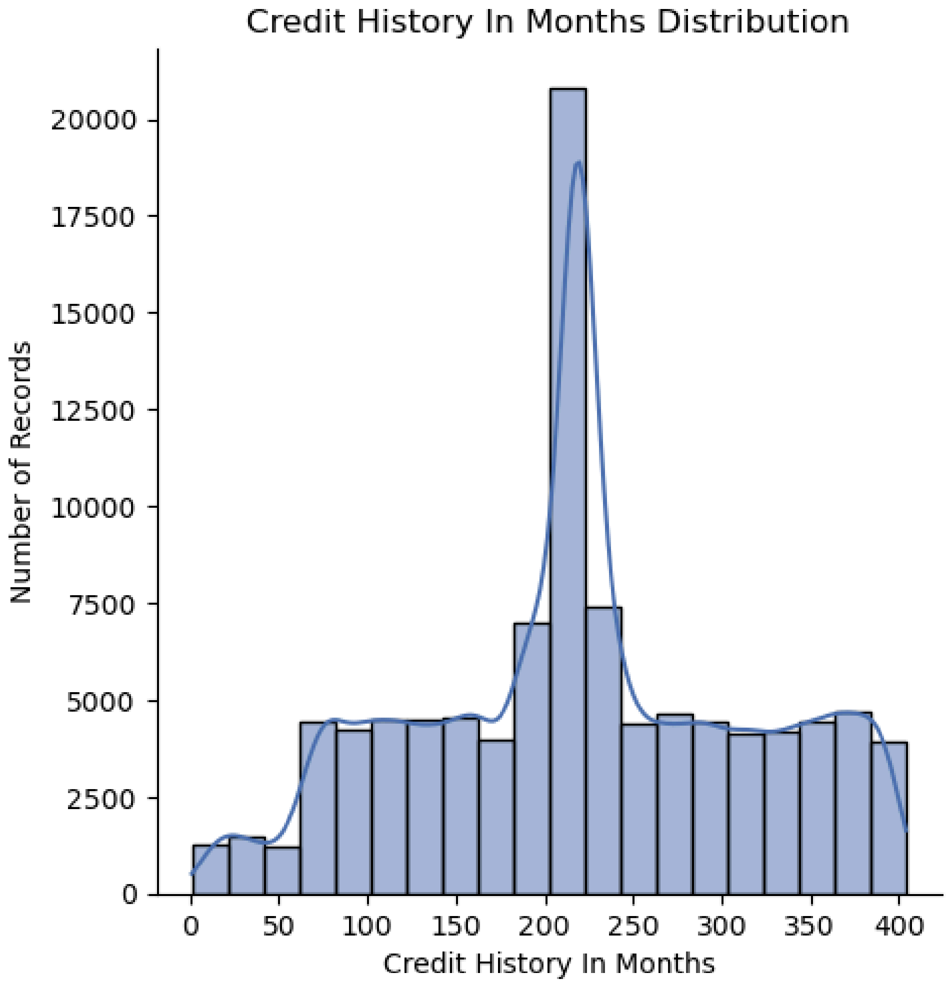
In [63]:

```python
column_name = 'Credit_History_In_Months'
group_by = 'Customer_ID'
user_friendly_name = 'Credit History In Months'

# Cleaning
clean_numerical_field(df, group_by, column_name, datatype='float')

# Plot Graph
plot_distplot(df, column_name, user_friendly_name)
```

```
Cleaning steps:

Datatype of Credit_History_In_Months is changed to float

Existing Min, Max Values:
1.0
404.0
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
```

```
After Cleaning Min, Max Values:
1.0
404.0

Number of unique values after cleaning: 404
Number of null values after cleaning: 0



Credit History In Months Distribution
```



In [64]:
```
get_column_details(df, 'Total_EMI_per_month')
```

```
Details of Total_EMI_per_month column

DataType:  float64

There are no null values

Number of Unique Values:  14950

Distribution of column:

Total_EMI_per_month
0.000000        10613
49.574949           8
73.533361           8
22.960835           8
38.661127           8
                ...
36408.000000        1
23760.000000        1
24612.000000        1
24325.000000        1
58638.000000        1
Name: count, Length: 14950, dtype: int64
```

In [65]:
```
column_name = 'Total_EMI_per_month'
group_by = 'Customer_ID'
user_friendly_name = 'Total EMI per month'

# Cleaning
clean_numerical_field(df, group_by, column_name)

# Plot Graph
plot_distplot(df, column_name, user_friendly_name)
```

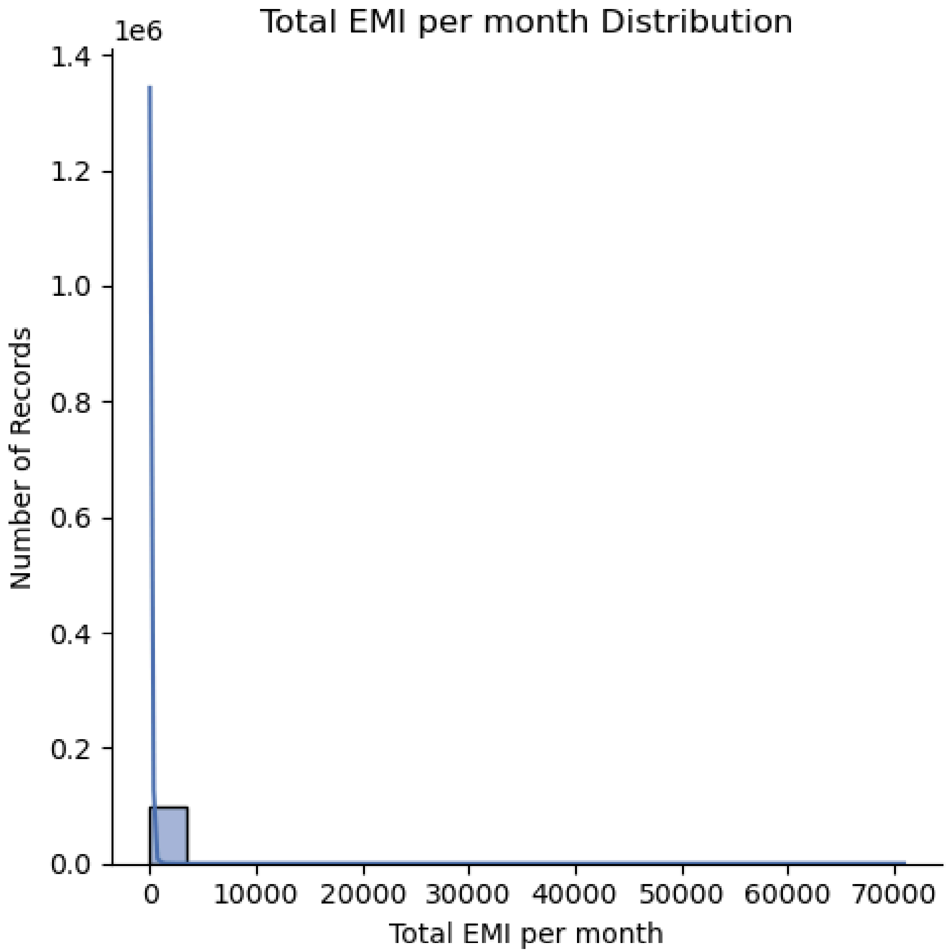
Cleaning steps:

Existing Min, Max Values:
0.0
82331.0

After Cleaning Min, Max Values:
0.0
70943.0

Number of unique values after cleaning: 11300
Number of null values after cleaning: 0

Total EMI per month Distribution



In [66]:
```
get_column_details(df, 'Amount_invested_monthly')
```

```
Details of Amount_invested_monthly column

DataType:  object

There are  4479  null values

Number of Unique Values:  91049

Distribution of column:

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
```

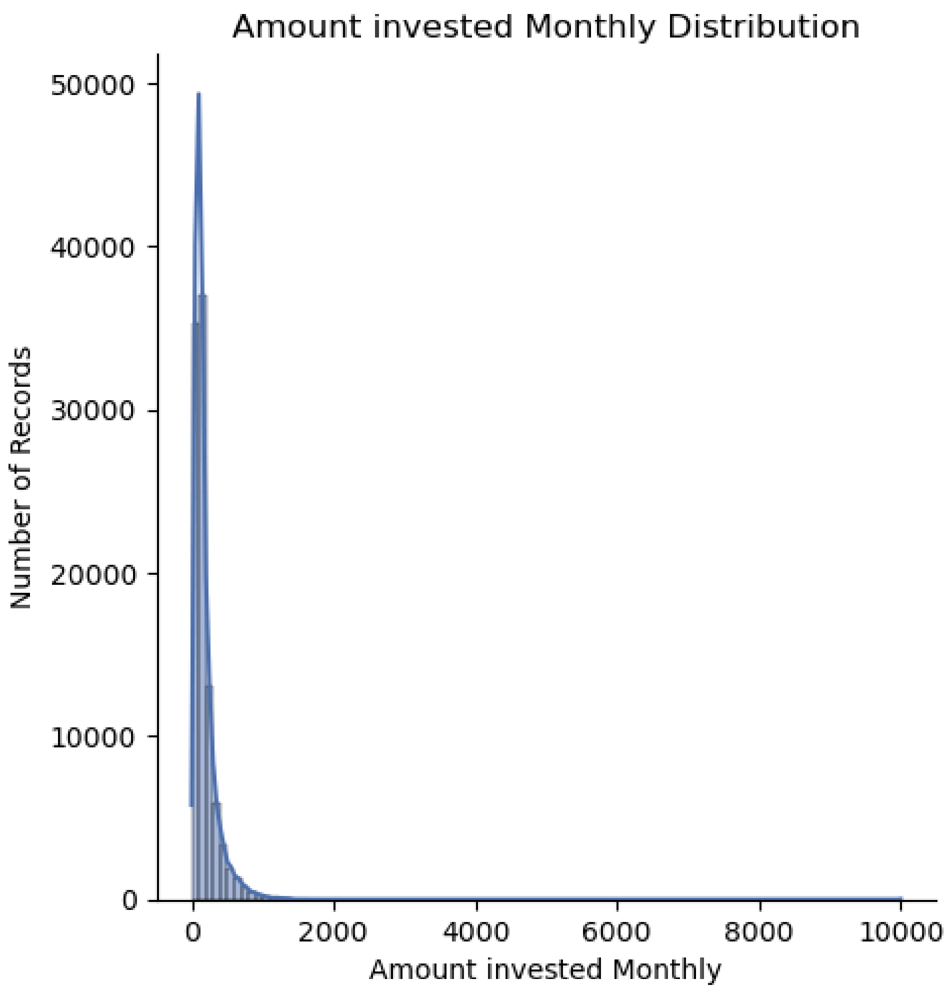
In [67]:
```
column_name = 'Amount_invested_monthly'
group_by = 'Customer_ID'
user_friendly_name = 'Amount invested Monthly'

# Cleaning
clean_numerical_field(df, group_by, column_name, datatype='float', strip='_')

# Plot Graph
plot_distplot(df, column_name, user_friendly_name, bins=100)
```

```
Cleaning steps:

Trailing and leading _ are removed

Datatype of Amount_invested_monthly is changed to float

Existing Min, Max Values:
0.0
10000.0
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
ing values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df
[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
```

```
After Cleaning Min, Max Values:
0.0
10000.0

Number of unique values after cleaning: 67721
Number of null values after cleaning: 0
```

Amount invested Monthly Distribution



In [68]:
```
get_column_details(df, 'Monthly_Balance')
```

```
Details of Monthly_Balance column

DataType:  object

There are  1200  null values

Number of Unique Values:  98790

Distribution of column:

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
```

In [69]:

```python
column_name = 'Monthly_Balance'
group_by = 'Customer_ID'
user_friendly_name = 'Monthly Balance'

#Cleaning
df[column_name].replace('', np.nan)

clean_numerical_field(df, group_by, column_name, strip='_', datatype=float, replace_value='__-333333333333333333333333333__')

#Plot Graph
plot_distplot(df, column_name, user_friendly_name, bins=30)
```

Cleaning steps:

Garbage value __-333333333333333333333333333__ is replaced with np.nan

Trailing and leading _ are removed

Datatype of Monthly_Balance is changed to <class 'float'>

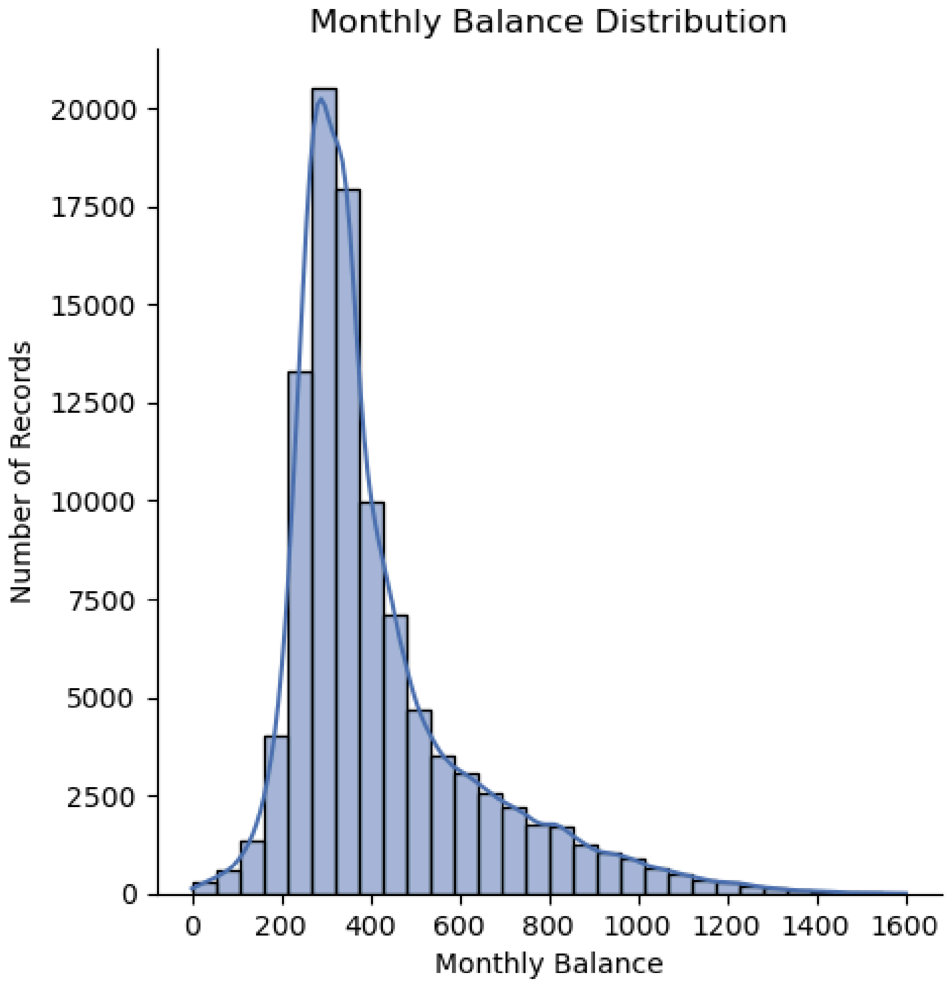
Existing Min, Max Values:
0.007759665
1602.040519

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2512144296.py:52: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[column].fillna(global_median, inplace=True)
```

After Cleaning Min, Max Values:
0.095482496
1602.040519

Number of unique values after cleaning: 87616
Number of null values after cleaning: 0

Monthly Balance Distribution



In [70]:
```
get_column_details(df, 'Num_of_Loan')
```

Details of Num_of_Loan column

DataType: object

There are no null values

Number of Unique Values: 434

Distribution of column:

```
Num_of_Loan
3        14386
2        14250
4        14016
0        10380
1        10083
6         7405
7         6930
5         6865
-100      3876
9         3542
8         3035
2_         782
4_         727
3_         718
0_         550
1_         523
7_         414
6_         398
5_         332
9_         160
8_         156
1150         4
773          3
1228         3
430          3
288          3
1480         3
1384         2
1181         2
404          2
```

875	2
23	2
1241	2
1259	2
192	2
229	2
911	2
217	2
1209	2
1412	2
1353	2
95	2
50	2
275	2
58	2
1354	2
697	2
898	2
284	2
1365	2
936	2
955	2
33	2
661	2
290	2
330	2
1236	2
733	2
1217	2
31	2
172	2
855	2
1463	2
1127	2
141	2
251	2
466	2
1214	2
1017	2
352	2
501	2

49	2
1464	2
433	1
1023	1
1430	1
1054	1
1307	1
1040	1
679	1
227	1
1077	1
295	1
1008	1
1171_	1
581	1
1320	1
917	1
814	1
579	1
1132_	1
1196	1
1457	1
1159	1
1470	1
809	1
190	1
227_	1
1297	1
968	1
19	1
868	1
126	1
657	1
1222	1
372	1
1478	1
437	1
889	1
1382	1
83	1
701	1

201	1
56	1
1294	1
881	1
237	1
101	1
999	1
538	1
910	1
216	1
785_	1
1225	1
832	1
659	1
462	1
148	1
349	1
1085	1
742	1
1447	1
291	1
365	1
87	1
633	1
1216	1
285	1
1189	1
1482	1
1359	1
18	1
27_	1
41	1
1131_	1
1393	1
143_	1
387	1
289	1
1419	1
621	1
801	1
978	1

831	1
629	1
447	1
628	1
84	1
1039	1
1074	1
1279	1
869	1
415	1
242	1
527_	1
574	1
182	1
546	1
1112	1
1210	1
799	1
1298	1
1289	1
1160	1
526	1
838	1
515	1
464	1
863	1
958	1
54	1
1265	1
1129	1
1345	1
897	1
1027_	1
635	1
529	1
662	1
653	1
927	1
497	1
656	1
463	1

507	1
350	1
147	1
208	1
924	1
778	1
696_	1
300	1
1103	1
254	1
873	1
652	1
548	1
867	1
686	1
65	1
378_	1
1091	1
1002	1
926	1
606	1
344	1
316	1
215	1
991	1
1340	1
1274	1
520	1
571	1
1372	1
1036	1
1371	1
1496	1
1296	1
753	1
1053	1
228	1
504	1
636	1
196	1
439	1

1014	1
1300	1
136	1
1271	1
1400	1
1219_	1
802	1
1178	1
17	1
78	1
1137	1
292	1
1129_	1
1406	1
1006	1
819	1
1154	1
1424	1
1312	1
905	1
1461	1
153	1
1348	1
323	1
649	1
1363	1
614	1
348	1
654	1
1204	1
1094	1
1135	1
684	1
545	1
795	1
359	1
590	1
696	1
418	1
983	1
449	1

527	1
89	1
904	1
1131	1
420	1
1474	1
1096	1
55	1
455	1
1416	1
143	1
1369	1
816	1
70	1
1465	1
1185_	1
995	1
132	1
238	1
737	1
640	1
663	1
119	1
617	1
597_	1
843	1
313	1
728	1
1106	1
1485	1
484	1
720	1
444	1
341	1
563	1
146	1
945	1
472	1
622	1
967	1
92_	1

1019	1
1302	1
39	1
1451	1
329	1
699	1
193	1
1347_	1
1035	1
1047	1
52	1
1187	1
1433	1
457	1
1152	1
1110	1
1001	1
424	1
186	1
931	1
848	1
716	1
319	1
638	1
1182	1
32	1
1227	1
1225_	1
359_	1
833	1
416	1
757	1
1387	1
351	1
282	1
191	1
939	1
860	1
1329	1
100	1
781	1

1318	1
394	1
943	1
302	1
1202	1
639	1
859	1
243	1
267	1
29	1
596	1
935	1
940	1
891	1
895	1
1441	1
1311_	1
131_	1
540	1
123	1
336	1
820	1
492	1
311	1
597	1
996	1
332	1
510	1
562	1
952	1
138	1
1313	1
1151	1
438	1
834	1
1046	1
961	1
174	1
752	1
231	1
1319	1

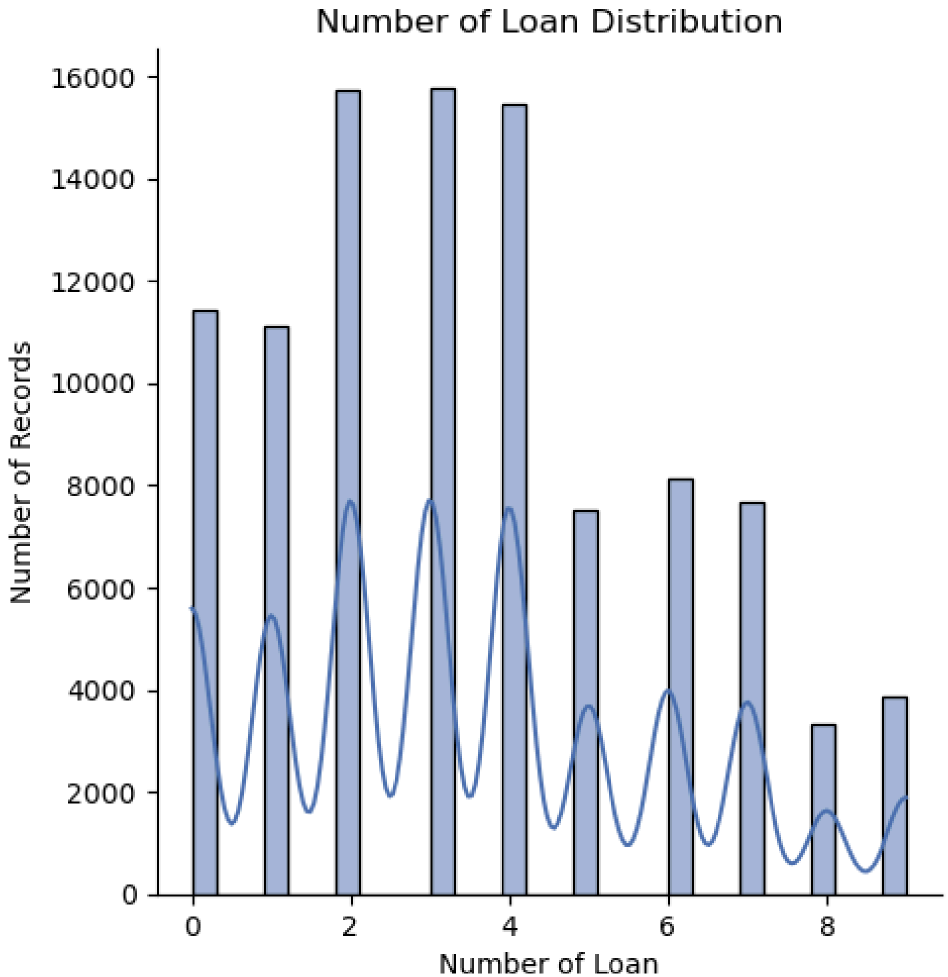
```
601         1
321         1
1391        1
1495        1
1459_       1
494         1
274         1
1484        1
1070        1
280         1
235_        1
719         1
198         1
1015        1
841         1
1439        1
613         1
1048        1
777         1
1088        1
164         1
157         1
137         1
1257        1
1030        1
405         1
241         1
630_        1
252         1
745         1
1320_       1
103         1
1444        1
392         1
966         1
Name: count, dtype: int64
```

In [71]:

```
column_name = 'Num_of_Loan'
group_by = 'Customer_ID'
user_friendly_name = 'Number of Loan'
```

```python
#Cleaning
clean_numerical_field(df, group_by, column_name, strip='_', datatype=float)

#Plot Graph
plot_distplot(df, column_name, user_friendly_name, bins=30)
```

```
Cleaning steps:

Trailing and leading _ are removed

Datatype of Num_of_Loan is changed to <class 'float'>

Existing Min, Max Values:
-100.0
1496.0

After Cleaning Min, Max Values:
0.0
9.0

Number of unique values after cleaning: 10
Number of null values after cleaning: 0



Number of Loan Distribution
```




```
In [72]: df.isna().sum()
```

Out[72]:
```
ID                           0
Customer_ID                  0
Month                        0
Name                         0
Age                          0
SSN                          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        9030
Payment_of_Min_Amount        0
Total_EMI_per_month          0
Amount_invested_monthly      0
Payment_Behaviour            0
Monthly_Balance              0
Type_of_Loan_Split           0
Credit_History_In_Months     0
dtype: int64
```

In [73]:
```
df.to_csv('processed_credit_score.csv', index=False)
```

In [74]:
```
df = pd.read_csv("processed_credit_score.csv")
df.head(10)
```

Out[74]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_C
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
5	0x1607	CUS_0xd40	June	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
6	0x1608	CUS_0xd40	July	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_C
7	0x1609	CUS_0xd40	August	Aaron Maashoh	23.0	821-00-0265	Scientist	19114.12	1824.843333	3.0	
8	0x160e	CUS_0x21b1	January	Rick Rothackerj	28.0	004-07-5839	Teacher	34847.84	3037.986667	2.0	
9	0x160f	CUS_0x21b1	February	Rick Rothackerj	28.0	004-07-5839	Teacher	34847.84	3037.986667	2.0	

Label Encoding Features

In [75]:
```
df["Payment_of_Min_Amount"] = df["Payment_of_Min_Amount"].replace({"Yes": 1, "No": 0, "NM": 0})
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\1251687005.py:1: FutureWarning: Downcasting behavior in `replace` is deprecat
ed and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["Payment_of_Min_Amount"] = df["Payment_of_Min_Amount"].replace({"Yes": 1, "No": 0, "NM": 0})
```

In [76]:
```
df["Credit_Mix"] = df["Credit_Mix"].replace({"Standard":1, "Good":2, "Bad":0})

df["Payment_Behaviour"] = df["Payment_Behaviour"].replace({
                                            "Low_spent_Small_value_payments": 1,
                                            "High_spent_Small_value_payments": 2,

                                            "Low_spent_Medium_value_payments": 3,
                                            "High_spent_Medium_value_payments": 4,

                                            "Low_spent_Large_value_payments": 5,
                                            "High_spent_Large_value_payments": 6
```

```
              })
```

```
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2501471634.py:1: FutureWarning: Downcasting behavior in `replace` is deprecat
ed and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["Credit_Mix"] = df["Credit_Mix"].replace({"Standard":1, "Good":2, "Bad":0})
C:\Users\Minisha\AppData\Local\Temp\ipykernel_4748\2501471634.py:3: FutureWarning: Downcasting behavior in `replace` is deprecat
ed and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["Payment_Behaviour"] = df["Payment_Behaviour"].replace({
```

Feature Engineering

In [77]:
```
# debt to income ratio

df['Monthly_Debt_to_Income_Ratio'] = df['Outstanding_Debt'] / df['Monthly_Inhand_Salary']

df['Monthly_Debt_Repayment_Capacity'] = df['Monthly_Inhand_Salary'] - df['Total_EMI_per_month']

df["Payment_History_Score"] = (
    -1 * df["Delay_from_due_date"]
    - 1 * df["Num_of_Delayed_Payment"]
    + 1 * df["Payment_of_Min_Amount"]
)
```

Selected features for credit score calculation with their weights:

1. Payment History Score
    - Weight: 0.40
    - Strongest predictor of future credit behavior.
2. Credit History Age in Months
    - Weight: 0.20
    - Longer credit history indicates responsible credit usage. Weighted moderately to reflect its significance.
3. Monthly Debt-to-Income Ratio (MDTIR)

- Weight: 0.15
- Lower ratio indicates better ability to manage debt. Weighted lower due to potential fluctuations in income.

4. Credit Utilization Ratio

    - Weight: 0.10
    - Lower ratio suggests responsible credit card usage. Weighted lower as it's a snapshot of current utilization.

5. Monthly Debt Repayment Capacity

    - Weight: 0.05
    - Reflects ability to manage existing debt.

6. Outstanding Debt

- Weight: 0.05
- Higher debt increases risk of default.

7. Num_Credit_Inquiries

- Weight: 0.03
- Fewer inquiries suggest lower credit-seeking behavior.

8. Payment Behaviour

- Weight: 0.02
- Insights into spending patterns and payment tendencies.

In [78]:
```
# Credit score calculation function
def calculate_credit_score(data):

    # Group by Customer ID, handling month-level data and calculating scores
    grouped_data = data.groupby("Customer_ID").agg(
        Payment_History_Score=("Payment_History_Score", "mean"),  # Use average payment history score
        Credit_History_In_Months=("Credit_History_In_Months", "max"),  # Use maximum history age
        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"),
        Outstanding_Debt=("Outstanding_Debt", "mean"),
```

```python
        Num_Credit_Inquiries=("Num_Credit_Inquiries", "sum"),
        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"] = (
        0.40 * grouped_data["Payment_History_Score"]
        + 0.20 * grouped_data["Credit_History_In_Months"]
        + 0.15 * (1 - grouped_data["Monthly_Debt_to_Income_Ratio"])  # Inverse relation as lower value is better
        + 0.10 * (1 - grouped_data["Credit_Utilization_Ratio"])  # Inverse relation
        + 0.05 * grouped_data["Monthly_Debt_Repayment_Capacity"]
        + 0.05 * grouped_data["Outstanding_Debt"]
        + 0.03 * (1 - grouped_data["Num_Credit_Inquiries"])  # Inverse relation
        + 0.02 * 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())
        / (grouped_data["credit_score"].max() - grouped_data["credit_score"].min()) * 100
    )

    return grouped_data.reset_index()

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

Out[78]:

	Customer_ID	credit_score
0	CUS_0x1000	32.273715
1	CUS_0x1009	85.384618
2	CUS_0x100b	75.070844
3	CUS_0x1011	69.079794
4	CUS_0x1013	78.834776
...	...	...
12495	CUS_0xff3	70.144295
12496	CUS_0xff4	70.824669
12497	CUS_0xff6	90.394219
12498	CUS_0xffc	50.526534
12499	CUS_0xffd	69.311865

12500 rows × 2 columns

Summary

1. We have record of 12500 unique customers

2. In the dataset, we have data for each customer over the course of 8 months (from January to August)

3. We have following types of loans

- auto loan
- credit-builder loan
- debt consolidation loan
- home equity loan
- mortgage loan
- not specified

- payday loan
- personal loan
- student loan

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

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

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

7. 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.

8. Interest rate ranges from 1% to 34%.

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

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

11. Customers typically take anywhere fro m 2 to 4 loans, w ith the maximumnumber being 9.

For credit score calculation we have used following features with their respective weights

- Selected features for credit score calculation with their weights:

   1. Payment history score: (Weight: 0.40)
   2. Credit History Age in Months (Weight: 0.20)
   3. Monthly Debt-to-Income Ratio (MDTIR) (Weight: 0.15)
   4. Credit Utilization Ratio (Weight: 0.10)
   5. Monthly Debt Repayment Capacity (Weight: 0.05)
   6. Outstanding Debt (Weight: 0.05)
   7. Num_Credit_Inquiries (Weight: 0.03)
   8. Payment Behaviour (Weight: 0.02)

Recommendation:

- This is a very simple model that we have used for credit score calculation but to improve the reliability of this score.
- We can explore with different weighting schemes, or try different features.
- We should also explore other methods preferably ML based solutions that you can explore once you study different ML algorithms.

Performing Linear Regression to find important features

In [79]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor


# Remove 'Annual_Income' due to high multicollinearity
features_revised = df[['Age', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
                                        'Num_Credit_Card', 'Interest_Rate', 'Outstanding_Debt',
                                        'Total_EMI_per_month', 'Amount_invested_monthly',
                                        'Credit_Mix', 'Monthly_Balance']]

# Setting the target variable as the 'Credit_Utilization_Ratio'
target = df['Credit_Utilization_Ratio']

# Check for multicollinearity using Variance Inflation Factor (VIF)
scaler = StandardScaler()

# Re-scaling the features
features_scaled_revised = scaler.fit_transform(features_revised)

# Re-run the VIF check after removing 'Annual_Income'
vif_data_revised = pd.DataFrame()
vif_data_revised["Feature"] = features_revised.columns
vif_data_revised["VIF"] = [variance_inflation_factor(features_scaled_revised, i) for i in range(features_scaled_revised.shape[1

# Print VIF data to check multicollinearity again
print("Revised VIF after removing 'Annual Income':\n", vif_data_revised)
```

```
# Splitting the data into training and testing sets
X_train_revised, X_test_revised, y_train_revised, y_test_revised = train_test_split(features_scaled_revised, target, test_size=

# Re-run the model
model_revised = LinearRegression()
model_revised.fit(X_train_revised, y_train_revised)

# Making predictions
y_pred_revised = model_revised.predict(X_test_revised)

# Calculating performance
mse_revised = mean_squared_error(y_test_revised, y_pred_revised)
r2_revised = r2_score(y_test_revised, y_pred_revised)

# Extracting the coefficients of the features
coefficients_revised = pd.DataFrame({
    'Feature': features_revised.columns,
    'Coefficient': model_revised.coef_
}).sort_values(by='Coefficient', ascending=False)

# Output results
print(f"Revised Mean Squared Error: {mse_revised}")
print(f"Revised R-squared: {r2_revised}")
print("Revised Feature Coefficients:\n", coefficients_revised)
```

```
Revised VIF after removing 'Annual Income':
                   Feature       VIF
0                      Age  1.069381
1     Monthly_Inhand_Salary  3.772042
2         Num_Bank_Accounts  2.131048
3           Num_Credit_Card  1.505356
4             Interest_Rate  2.506955
5          Outstanding_Debt  2.076494
6        Total_EMI_per_month  1.035618
7   Amount_invested_monthly  1.687860
8                Credit_Mix  3.887516
9           Monthly_Balance  2.941189
Revised Mean Squared Error: 24.90514549367167
Revised R-squared: 0.05158643437760124
Revised Feature Coefficients:
                   Feature  Coefficient
9           Monthly_Balance     0.998839
1     Monthly_Inhand_Salary     0.239792
8                Credit_Mix     0.058927
4             Interest_Rate     0.028763
5          Outstanding_Debt     0.024285
2         Num_Bank_Accounts     0.002070
6        Total_EMI_per_month     0.001136
3           Num_Credit_Card    -0.003804
0                      Age    -0.008336
7   Amount_invested_monthly    -0.160940
```

Summary

1. **Positive Predictors of Credit Utilization**:
    - **Monthly Balance**: Strong positive relationship; higher balances indicate higher credit usage.
    - **Monthly Inhand Salary**: Higher salaries correlate with increased credit utilization, suggesting greater access to credit.
    - **Credit Mix**: A diverse mix of credit products positively impacts utilization, indicating comfort with using credit.
    - **Interest Rate**: Slightly positive relationship; customers with higher interest rates tend to use more credit.
2. **Negative Predictors of Credit Utilization**:

- **Amount Invested Monthly**: Strong negative impact; customers who invest more tend to use less credit, indicating financially prudent behavior.
- **Age**: Slight negative correlation; older customers typically use less credit than younger ones, possibly due to more stable financial situations.
- **Num Credit Cards**: Negative relationship suggests that customers with more credit cards use less credit overall.

Recommendations:

1. **Encourage Financially Prudent Behaviors**: Offer incentives, such as lower interest rates, for customers who demonstrate consistent saving or investing habits.

2. **Target High Income Earners**: Design specific credit products for higher-income customers who are more likely to utilize credit comfortably.

3. **Personalized Credit Offers**: Create targeted promotions for customers with diverse credit mixes to enhance loyalty and responsible credit usage.

4. **Educational Programs for Younger Customers**: Provide financial education to younger customers to promote responsible credit use and savings.

5. **Credit Monitoring Tools**: Offer tools that help customers track their credit utilization and receive alerts for high usage levels.

6. **Incentivize Low Credit Utilization**: Create rewards for customers who maintain low credit utilization, such as cashback or loyalty points.

7. **Segment Customers by Credit Behavior**: Develop customer segments to provide targeted credit products and educational resources based on their financial behaviors.

8. **Promote Investment and Savings Products**: Encourage responsible financial habits by bundling savings and investment products with credit offerings.