In [250…

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

In [251…

```python
df = pd.read_csv('D:\\Scaler\\Scaler\\Fintech Domain Course\\Credit Python EDA\\Credit_score.
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\3909935991.py:1: DtypeWarning: Columns (26) ha
ve mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('D:\\Scaler\\Scaler\\Fintech Domain Course\\Credit Python EDA\\Credit_scor
e.csv')
```

In [252…

```python
df.head()
```

Out[252]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	NaN
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333

5 rows × 27 columns

In [253…

```python
df.shape
```

Out[253]:

```
(100000, 27)
```

In [254…

```python
df.columns
```

Out[254]:

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

In [255…

```python
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 [256…

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

Out[256]:

```
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 [257…

```
df.isna().mean()*100
```

Out[257]:
```
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
```

In [258…
```
df.describe(exclude=np.number).T
```

Out[258]:

	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

In [259…
```
df['Age'].value_counts()
```

Out[259]:
```
38      2833
28      2829
31      2806
26      2792
32      2749
        ...
471        1
1520       1
8663       1
3363       1
1342       1
Name: Age, Length: 1788, dtype: int64
```

Observations:

1. Customer_ID has 12500 unique values. It means we have data of 12500 customers.
2. Month has only 8 unique values. Better to analyse further which months are present.
3. Age has 1788 unique values. This looks strange as general age range is from 0-100.
4. SSN has 12501 unique values, whereas Customer_ID only has only 12500 unique values. There is a possibility that incorrect SSN 5. value is entered for one of the customer as same person can't have multiple SSN.

Step 1: Buidling Common Functions for Data Cleaning

Analysing Data: 1: Getting details of columns (Features) including data type, null values, unique values and value counts

In [260…
```
def column_info(df,column):
    print("Details of",column,"column")

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

    #Checking for null values
    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")

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

    #Checking for value counts
    print("\n Series of Unique Values:\n")
    print(df[column].value_counts())
```

Feature Engineering for Numerical columns: 1. Filling Missing values with 'mode'

In [261…
```
def feat_eng1_num_replace_with_mode(df, groupby, column):
    print("\n No. of missing values before feature engineering: ", df[column].isnull().sum())

    #Filling process with mode
    mode_process = df.groupby(groupby)[column].transform(lambda x: x.mode().iat[0])
    #mode_process = df.groupby(groupby)[column].transform(lambda x: x.mode(keepdims=True).iat


    df[column] = df[column].fillna(mode_process)

    print("\n No. of missing values after feature engineering: ", df[column].isnull().sum())
```

Feature Engineering for Numerical columns: 2. Handling Outliers and null values together

In [262…

```
def feat_eng2_replace_outliers_null(df, groupby, column):
    print("\n Min, Max Values:", df[column].apply([min, max]), sep='\n', end='\n')

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

    # Replace with NaN to outliers
    col = df[column].apply(lambda x: np.NaN if ((x<mini)|(x>maxi)|(x<0)) else x)

    # fill with mode values
    mode_by_group = df.groupby(groupby)[column].transform(lambda x: x.mode()[0] if not x.mode

    #mode_by_group = df.groupby(groupby)[column].transform(lambda x: x.mode(keepdims=True)[0]

    df[column] = col.fillna(mode_by_group)

    #Filling Remaining NaN Values with Mean
    df[column].fillna(df[column].mean(), inplace=True)

    print("\n After data Cleaning Min, Max Values:", df[column].apply([min, max]), sep='\n',
    print("\n No. of Unique values after Cleaning:",df[column].nunique())
    print("\n No. of Null values after Cleaning:",df[column].isnull().sum())
```

Feature Engineering for Numerical columns: 3. Removing undefined/garbage values

In [263…

```
def feat_eng3_num_replace_undefinedVal(df, groupby, column, strip=None, datatype=None, replac
    #Replace with np.nan
    if replace_value != None:
        df[column] = df[column].replace(replace_value, np.nan)
        print(f"\n Undefined value {replace_value} is replaced with np.nan")

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

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


    feat_eng2_replace_outliers_null(df, groupby, column)
```

Feature Engineering for Categorical columns: 1. Replacing with null values or filling Missing values with 'mode'

In [264…

```
def feat_eng4_cat_replace_with_null_mode(df, groupby, column, replace_value = None):
    print("\n Cleaning Categorical column: ", column)

    #Replace with null values
    if replace_value != None:
        df[column] = df[column].replace(replace_value, np.nan)

    feat_eng1_num_replace_with_mode(df, groupby, column)
```

Buidling Common Functions for Data Visualization

In [265…

```
#countplot

def plot_countplot(df, column, edited_column, rotation=0):
    print(f'\n{edited_column} Distribution')

    palette = "deep"
    sns.set_palette(palette)

    sns.countplot(data=df, x=column)

    plt.xlabel(f'{edited_column}')
    plt.ylabel('Number of Records')
    plt.title(f'{edited_column} Distribution')
    plt.xticks(rotation=rotation)

    plt.show()
```

In [266…

```
#displot

def plot_displot(df, column, edited_column, rotation=0, bins=20):
    print(f'\n{edited_column} Distribution')
    palette = "deep"
    sns.set_palette(palette)

    sns.displot(data=df, x=column, kde=True, bins=bins)

    plt.xlabel(f'{edited_column}')
    plt.ylabel('Number of Records')
    plt.title(f'{edited_column} Distribution')
    plt.xticks(rotation=rotation)

    plt.show()
```

In [267…

```
#stackedbar

def plot_stacked_bar(df, column1, column2, rotation=0):
    print(f'\n{column1} & {column2} Distribution')
    palette = "deep"
    sns.set_palette(palette)

    pd.crosstab(df[column1], df[column2]).plot(kind='bar', stacked=True)

    plt.xlabel(f'{column1}')
    plt.ylabel('Number of Records')
    plt.title(f'{column1} & {column2} Distribution')
    plt.xticks(rotation=rotation)

    plt.show()
```

Analysing Important Features for data cleaning

In [268…

```
#getting details of ID

column_info(df,'ID')
```

```
Details of ID column

DataType:  object

There are no null values

Number of Unique Values:  100000

 Series of Unique Values:

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

In [269…

```
#getting details of ID

column_info(df,'Customer_ID')
```

```
Details of Customer_ID column

DataType:  object

There are no null values

Number of Unique Values:  12500

 Series of Unique Values:

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: Customer_ID, Length: 12500, dtype: int64
```

In [270…

```
#getting details of Month

column_info(df,'Month')
```

```
Details of Month column

DataType:  object

There are no null values

Number of Unique Values:  8

 Series of Unique Values:

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

In [271…

```
#Convert Month to datetime object
df['Month'] = pd.to_datetime(df.Month, format='%B').dt.month
```

In [272…

```
df.columns
```

Out[272]:

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

In [273…

```
groupby = 'Customer_ID'
column = 'Name'


feat_eng4_cat_replace_with_null_mode(df, groupby, column, replace_value = None)
```

```
 Cleaning Categorical column:  Name

 No. of missing values before feature engineering:  9985

 No. of missing values after feature engineering:  0
```

In [274…

```
#getting details of SSN

groupby = 'Customer_ID'
replace_value = '#F%$D@*&8'
column ='SSN'

column_info(df,'SSN')

feat_eng4_cat_replace_with_null_mode(df, groupby, column, replace_value)
```

```
Details of SSN column

DataType:  object

There are no null values

Number of Unique Values:  12501

 Series of Unique Values:

#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: SSN, Length: 12501, dtype: int64

 Cleaning Categorical column:  SSN

 No. of missing values before feature engineering:  5572

 No. of missing values after feature engineering:  0
```

In [275…

```python
#Get Details of Type of Loan column
column_info(df,'Type_of_Loan')
```

```
Details of Type_of_Loan column

DataType:  object

There are  11408  null values

Number of Unique Values:  6260

 Series of Unique Values:

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 Consolidat
ion Loan, and Debt Consolidation Loan       8
Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan
8
Name: Type_of_Loan, Length: 6260, dtype: int64
```

In [276…

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

In [277…

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

Out[277]:

```
Not Specified
12816
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 Consolidat
ion Loan, and Debt Consolidation Loan        8
Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan
8
Name: Type_of_Loan, Length: 6260, dtype: int64
```

In [278…

```
column_name = 'Credit_Mix'

#Get Details of Type of Credit_Mix column

column_info(df,'Credit_Mix')
```

```
Details of Credit_Mix column

DataType:  object

There are no null values

Number of Unique Values:  4

 Series of Unique Values:

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

In [279…

```
# Data Cleaning
column = 'Credit_Mix'
groupby = 'Customer_ID'
replace_value = '_'

feat_eng4_cat_replace_with_null_mode(df, groupby, column, replace_value)
```

```
 Cleaning Categorical column:  Credit_Mix

 No. of missing values before feature engineering:  20195

 No. of missing values after feature engineering:  0
```

In [280…

```
#Get Details
column_info(df,'Payment_Behaviour')
```

```
Details of Payment_Behaviour column

DataType:  object

There are no null values

Number of Unique Values:  7

 Series of Unique Values:

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

In [281…

```
column = 'Payment_Behaviour'
groupby = 'Customer_ID'
replace_value = '!@9#%8'
```

In [282…

```
feat_eng4_cat_replace_with_null_mode(df, groupby, column, replace_value)
```

```
 Cleaning Categorical column:  Payment_Behaviour

 No. of missing values before feature engineering:  7600

 No. of missing values after feature engineering:  0
```

Numerical Features

In [283…

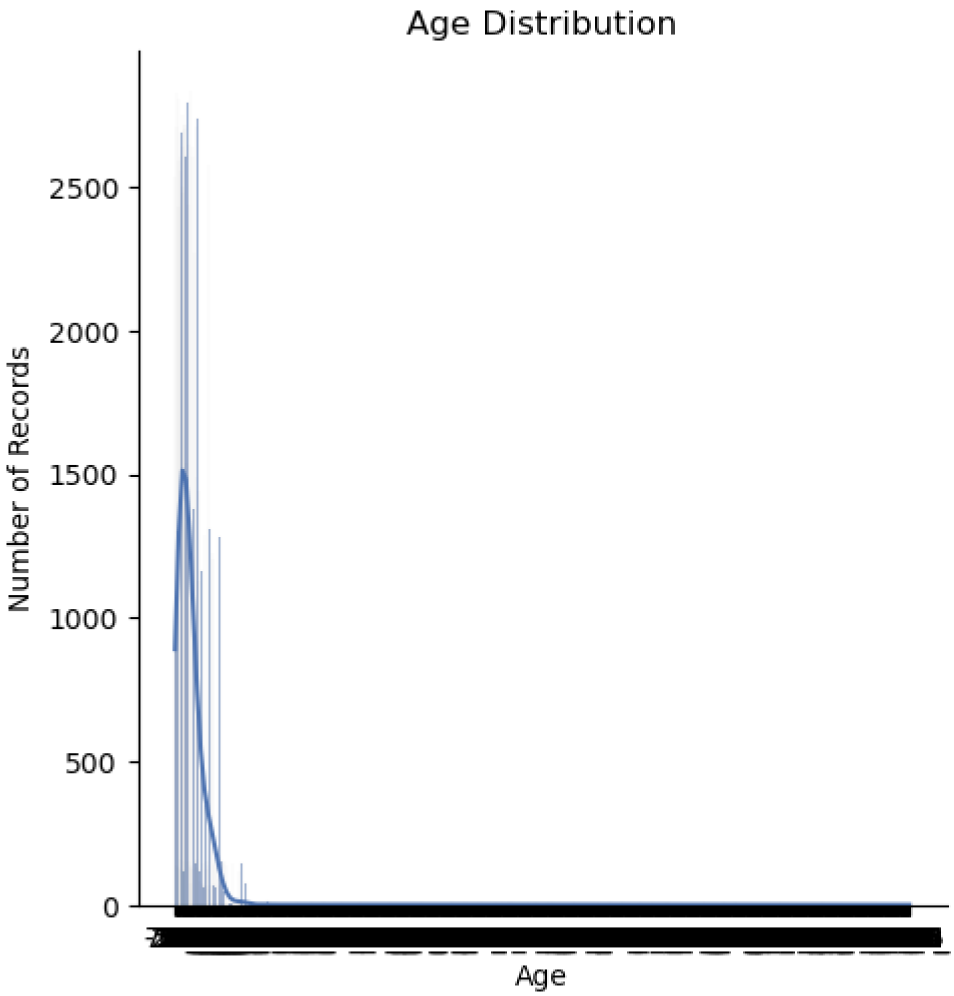
```
column = 'Age'

edited_column = 'Age'
#Get Details
column_info(df,'Age')
```

```
Details of Age column

DataType:  object

There are no null values

Number of Unique Values:  1788

 Series of Unique Values:

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

In [284…

```
#Plot Graph
#plot_displot(df,'Age','Age', bins=40)
plot_displot(df, column, edited_column, rotation=0, bins=20)
```

```
Age Distribution
```



In [285…

```
groupby = 'Customer_ID'
column = 'Age'

#Cleaning

feat_eng3_num_replace_undefinedVal(df, groupby, column, strip='_', datatype=int)
```

```
Trailing & leading _ are removed

Datatype of Age is changed to <class 'int'>

 Min, Max Values:
min    -500
max    8698
Name: Age, dtype: int64
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
```

```
 After data Cleaning Min, Max Values:
min    14.0
max    56.0
Name: Age, dtype: float64

 No. of Unique values after Cleaning: 43

 No. of Null values after Cleaning: 0
```

In [286…

```
column = 'Annual_Income'
groupby = 'Customer_ID'
edited_column = 'Annual Income'
```

```python
#Get Details
column_info(df,'Annual_Income')
```

```
Details of Annual_Income column

DataType:  object

There are no null values

Number of Unique Values:  18940

 Series of Unique Values:

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: Annual_Income, Length: 18940, dtype: int64
```

In [287…

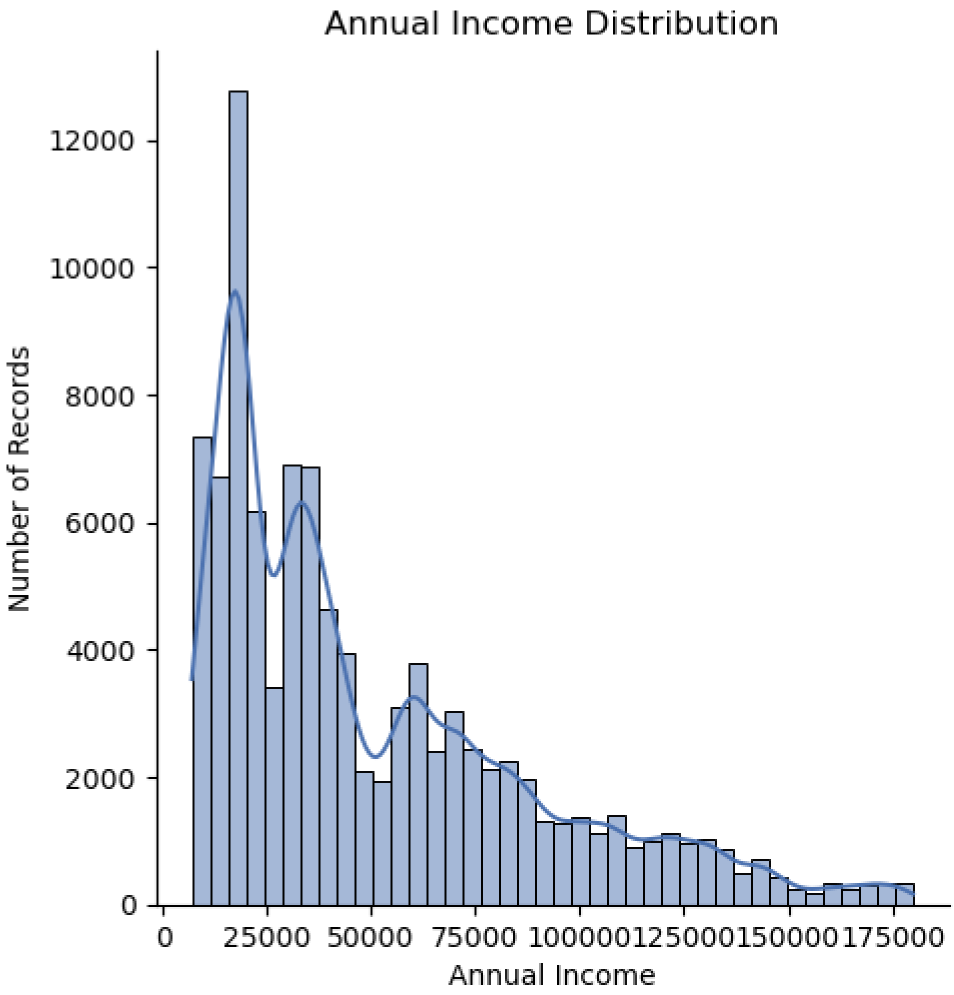
```python
#Cleaning
feat_eng3_num_replace_undefinedVal(df, groupby, column, strip='_', datatype=float)
```

```
Trailing & leading _ are removed

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

 Min, Max Values:
min        7005.93
max    24198062.00
Name: Annual_Income, dtype: float64
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
 After data Cleaning Min, Max Values:
min      7005.93
max    179987.28
Name: Annual_Income, dtype: float64

 No. of Unique values after Cleaning: 12614

 No. of Null values after Cleaning: 0
```

In [288…

```python
plot_displot(df,column,edited_column,bins=40)
```

```
Annual Income Distribution
```



In [289…
```
column = 'Num_Bank_Accounts'
edited_column = 'Number of Bank Accounts'
group_by = 'Customer_ID'
```

In [290…
```
#Get Details
column_info(df,'Num_Bank_Accounts')
```

```
Details of Num_Bank_Accounts column

DataType:  int64

There are no null values

Number of Unique Values:  943

 Series of Unique Values:

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

In [291…
```
#Cleaning
feat_eng3_num_replace_undefinedVal(df, groupby, column)
```

```
 Min, Max Values:
min      -1
max    1798
Name: Num_Bank_Accounts, dtype: int64
```

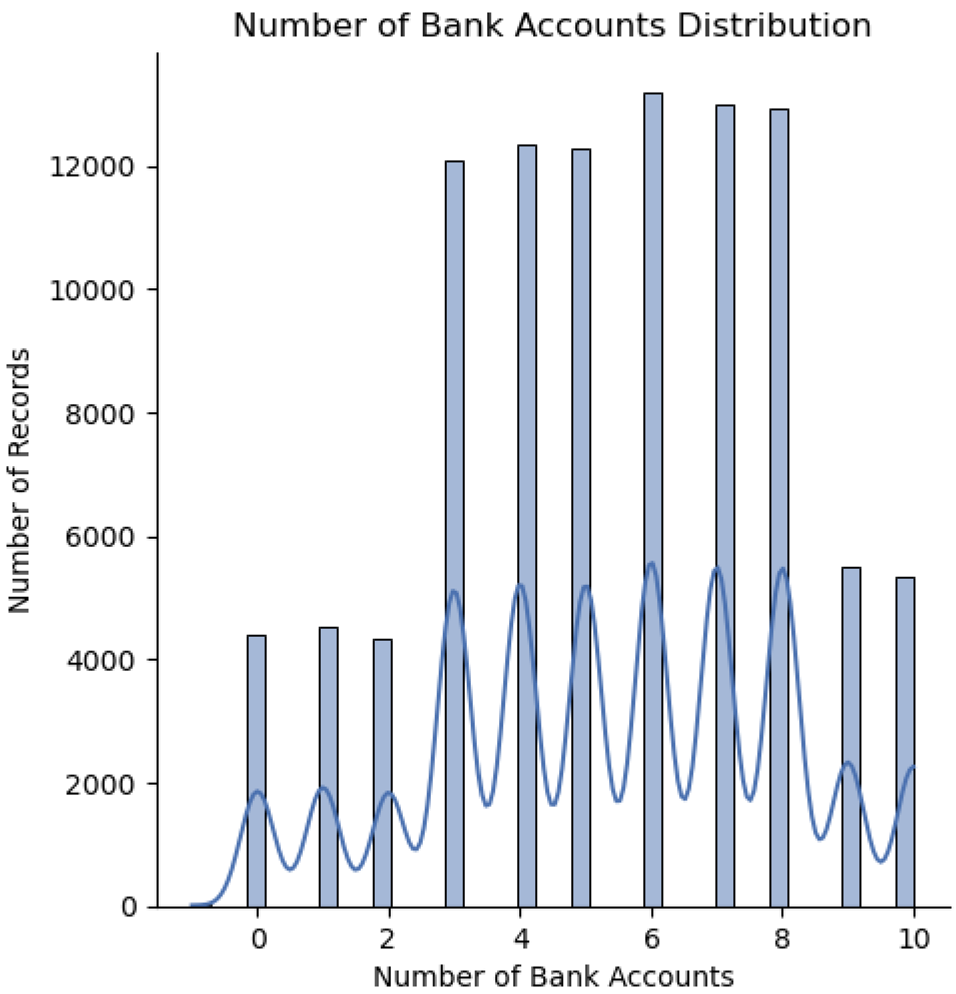
```
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
```

```
 After data Cleaning Min, Max Values:
min    -1.0
max    10.0
Name: Num_Bank_Accounts, dtype: float64

 No. of Unique values after Cleaning: 12

 No. of Null values after Cleaning: 0
```

In [292…

```
plot_displot(df,column,edited_column,bins=40)
```

```
Number of Bank Accounts Distribution
```



Feature Engineering

In order to calculate Credit Score, We will use the following compenents and weightage:

```
** (Payment_history * 0.35) +
** (credit utilization ratio * 0.15) +
** (Monthly_Debt_to_Income_Ratio * 0.15) +
```

```
** (No. of credit card a/c * 0.15) +
** (Employment Status (Occupation) * 0.10) +
** (Credit History Age * 0.10)
```

Payment History (Payment_history * 0.35):

Reasoning: Payment history is a fundamental factor in assessing creditworthiness. Timely payments contribute positively to the credit score, while delayed or missed payments can have a negative impact. Assigning the highest weight (0.35) to payment history reflects its significance in predicting an individual's ability to manage debt responsibly.

Credit Utilization Ratio (credit utilization ratio * 0.15):

Reasoning: The credit utilization ratio is the ratio of current credit card balances to credit limits. A low credit utilization ratio is generally considered favorable, indicating responsible credit usage. By assigning a weight of 0.30, we acknowledge the importance of maintaining a healthy balance between available credit and credit usage in determining creditworthiness.

Number of Credit Card Accounts (No. of credit card a/c * 0.15):

Reasoning: The number of active credit card accounts provides insights into an individual's credit management. Having a reasonable number of credit card accounts can positively impact credit scores. Assigning a weight of 0.15 acknowledges the role of credit diversity and responsible credit card usage in the overall creditworthiness assessment.

Monthly Debt to Income Ratio (No. of credit card a/c * 0.15):

Reasoning: Monthly Debt to Income Ratio provides insights into an individual's financial health. A lower ratio indicates that a person has more disposable income after meeting their debt obligations, which is generally considered a positive factor.

Employment Status (Occupation) (Employment Status * 0.10):

Reasoning: Employment status, represented by the Occupation column, is considered in assessing stability and financial capability. Certain occupations may indicate a steady income and job security. Assigning a weight of 0.10 recognizes the influence of employment status on an individual's ability to meet financial obligations.

Credit History Age (Credit History Age * 0.10):

Reasoning: The age of credit history reflects the length of time an individual has been using credit. A longer credit history is generally viewed positively as it provides a more extended track record of credit management. Assigning a weight of 0.10 recognizes the importance of a well-established credit history in determining creditworthiness.

In summary, the chosen components and their respective weightages aim to capture key aspects of an individual's financial behavior, responsible credit usage, and stability. The weights are assigned based on the relative impact of each component on predicting creditworthiness, as well as industry standards and best practices in credit scoring.

1. Data Preparation for Payment_History

We will use

```
a. Delay_from_due_date
b. Num_of_Delayed_Payment
c. Payment_of_Min_Amount
```

to create new feature - Payment_Histroy

In the context of credit score calculation, the Payment_History feature is a crucial component that reflects an individual's creditworthiness based on their past payment behavior. The choice of using Delay_from_due_date, Num_of_Delayed_Payment and Payment_of_Min_Amount for creating the Payment_History feature is rooted in the following reasoning:

Delay_from_due_date: Timeliness of Payments: Delay_from_due_date provides information about the delay in making payments from the due date. Timely payments are a crucial indicator of financial responsibility and discipline. Impact on Creditworthiness: A consistent history of delayed payments negatively affects creditworthiness. By including this component, the credit score model captures the historical pattern of payment delays.

Num_of_Delayed_Payment: Frequency of Delays: The number of delayed payments (Num_of_Delayed_Payment) reflects the frequency of instances where a borrower failed to make payments on time. Risk Assessment: A higher number of delayed payments indicates a higher risk of default and financial instability. Including this component helps lenders assess the level of risk associated with a borrower's payment behavior.

Payment_of_Min_Amount: Meeting Minimum Obligations: Payment_of_Min_Amount signifies whether the borrower consistently meets at least the minimum payment obligations. This is essential for maintaining a positive credit history. Credit Management Discipline: Borrowers who consistently pay at least the minimum amount due demonstrate a certain level of credit management discipline. Including this component contributes to a more comprehensive evaluation of payment behavior.

a: Delay_from_due_date

In [293…

```
column = 'Delay_from_due_date'
edited_column = 'Delay from Due Date'
groupby = 'Customer_ID'

column_info(df,column)
```

```
Details of Delay_from_due_date column

DataType:  int64

There are no null values

Number of Unique Values:  73

 Series of Unique Values:

 15    3596
 13    3424
 8     3324
 14    3313
 10    3281
       ...
-4       62
 65      56
-5       33
 66      32
 67      22
Name: Delay_from_due_date, Length: 73, dtype: int64
```

In [294…

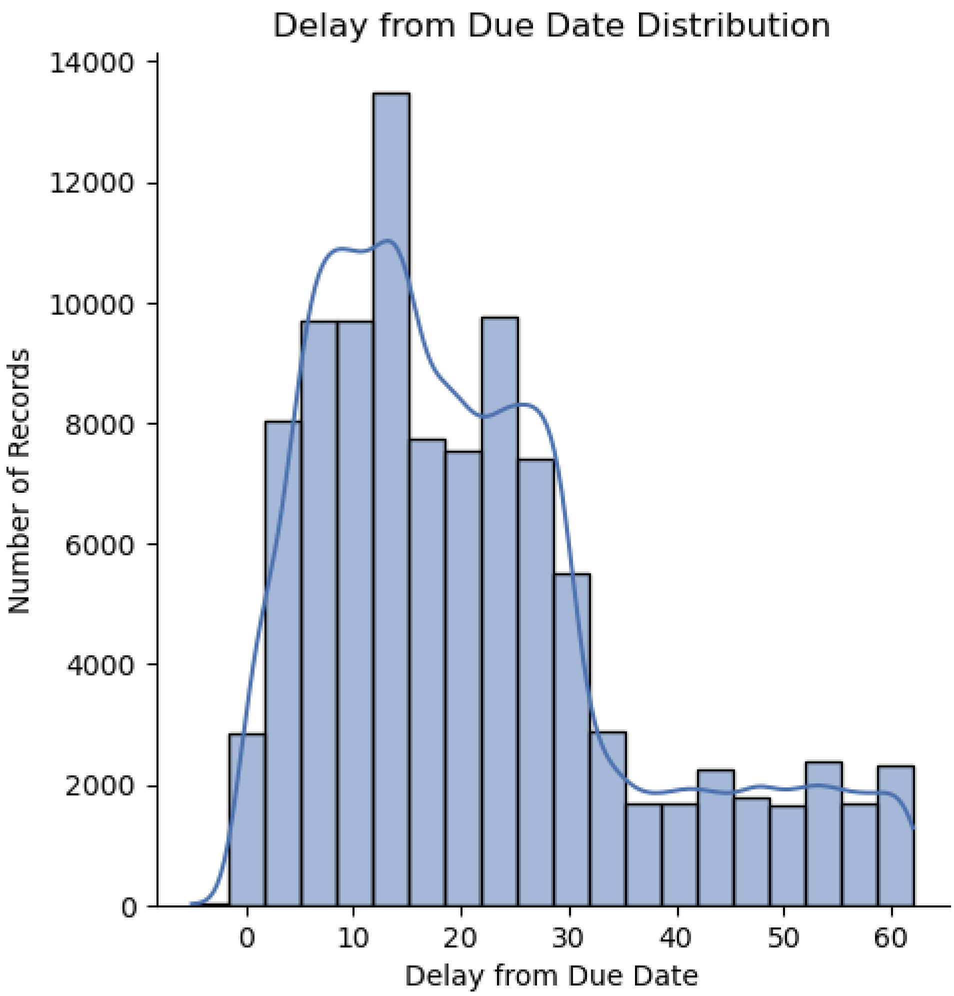
```
feat_eng3_num_replace_undefinedVal(df, groupby, column)
```

```
 Min, Max Values:
min    -5
max    67
Name: Delay_from_due_date, dtype: int64
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
 After data Cleaning Min, Max Values:
min    -5.0
max    62.0
Name: Delay_from_due_date, dtype: float64

 No. of Unique values after Cleaning: 68

 No. of Null values after Cleaning: 0
```

In [295…

```
plot_displot(df, column, edited_column, rotation=0, bins=20)
```

```
Delay from Due Date Distribution
```



b: Num_of_Delayed_Payment

In [296…

```
column = 'Num_of_Delayed_Payment'
edited_column = 'Number of Delayed Payment'

column_info(df,column)
```

```
Details of Num_of_Delayed_Payment column

DataType:  object

There are  7002  null values

Number of Unique Values:  749

 Series of Unique Values:

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

In [297…

```
groupby = 'Customer_ID'
feat_eng3_num_replace_undefinedVal(df, groupby, column, strip='_', datatype='float')
```

```
Trailing & leading _ are removed

Datatype of Num_of_Delayed_Payment is changed to float

 Min, Max Values:
min      -3.0
max    4397.0
Name: Num_of_Delayed_Payment, dtype: float64
```

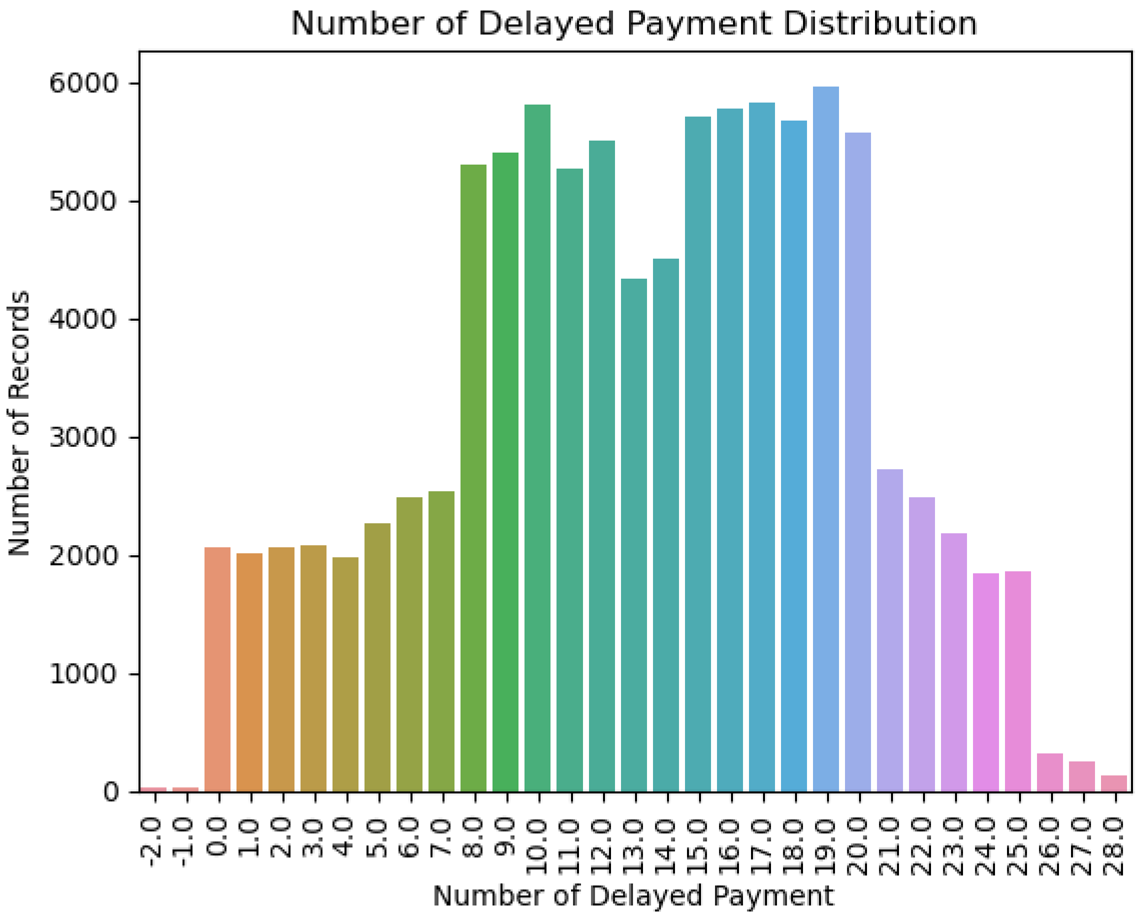
```
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
```

```
 After data Cleaning Min, Max Values:
min    -2.0
max    28.0
Name: Num_of_Delayed_Payment, dtype: float64

 No. of Unique values after Cleaning: 31

 No. of Null values after Cleaning: 0
```

In [298…

```
plot_countplot(df, column, edited_column, rotation=90)
```

```
Number of Delayed Payment Distribution
```



c: Payment_of_Min_Amount

In [299…

```
column = 'Payment_of_Min_Amount'

#Get Details
column_info(df,column)
```

```
Details of Payment_of_Min_Amount column

DataType:  object

There are no null values

Number of Unique Values:  3

 Series of Unique Values:

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

In [300…

```
#Implementing Label encoding Feature
df["Payment_of_Min_Amount"] = df["Payment_of_Min_Amount"].replace({"Yes": 1, "No": 0, "NM": 0
```

In [301…

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

In [302…

```
df[["Payment_History_Score"]]
```

Out[302]:

	Payment_History_Score
0	-10.0
1	-7.0
2	-10.0
3	-9.0
4	-10.0
...	...
99995	-30.0
99996	-25.0
99997	-33.0
99998	-26.0
99999	-24.0

100000 rows × 1 columns

2. Data Preparation for Credit_Utilization_Ratio

We will use Credit_Utilization_Ratio column directly from the data-set

In [303…

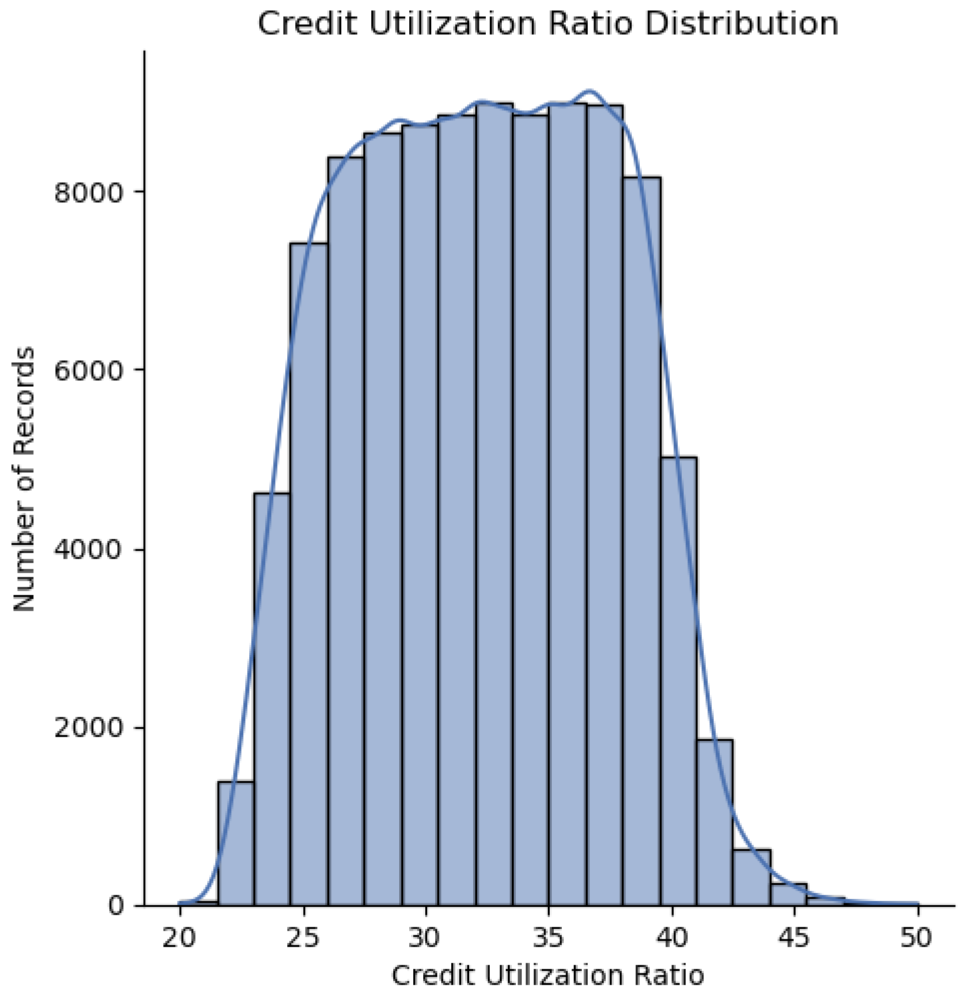
```
column = 'Credit_Utilization_Ratio'
edited_column = 'Credit Utilization Ratio'

column_info(df,column)
```

```
Details of Credit_Utilization_Ratio column

DataType:  float64

There are no null values

Number of Unique Values:  99998

 Series of Unique Values:

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: Credit_Utilization_Ratio, Length: 99998, dtype: int64
```

In [304…

```
    plot_displot(df, column, edited_column)
```

```
Credit Utilization Ratio Distribution
```



3. Data Preparation for Monthly_Debt_to_Income_Ratio

We will use

```
a. Outstanding_Debt
b. Monthly_Inhand_Salary
```

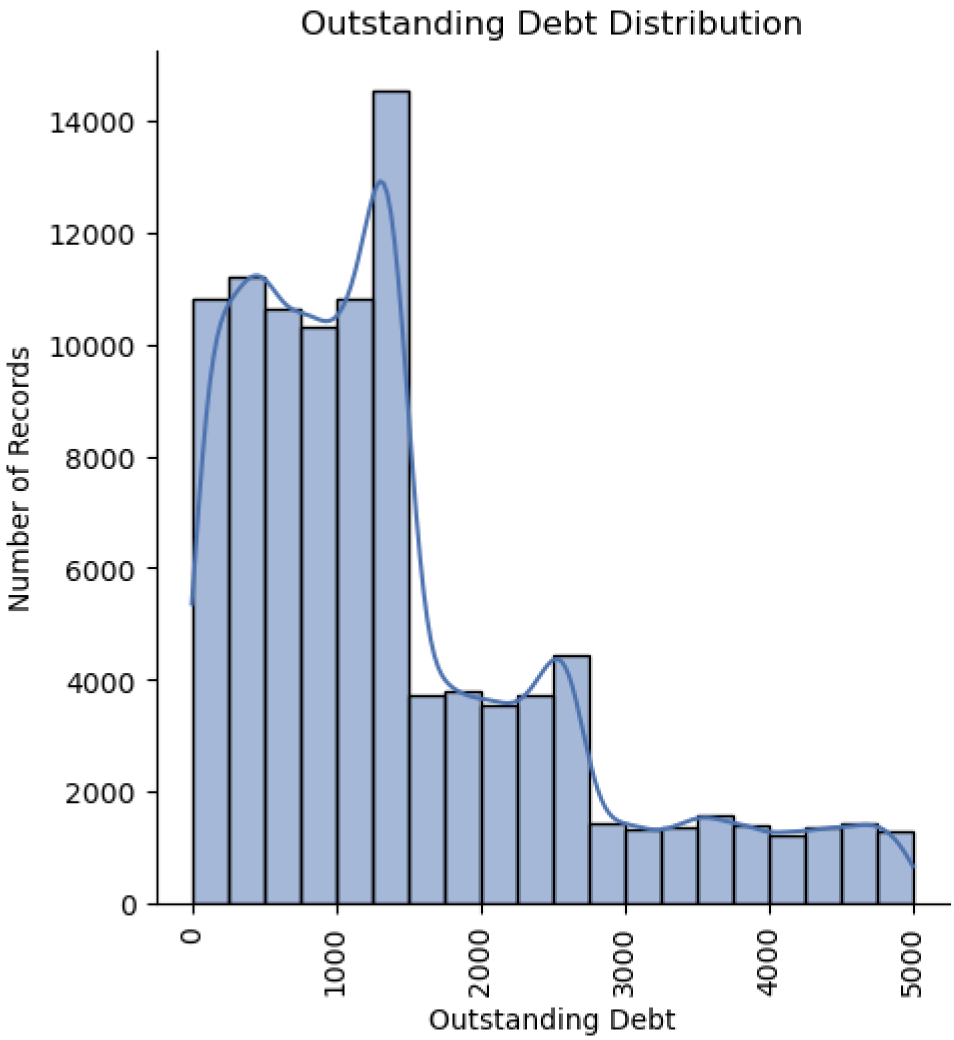
to create new feature - Monthly_Debt_to_Income_Ratio

a: Outstanding_Debt

In [305…

```
column = 'Outstanding_Debt'
groupby = 'Customer_ID'
edited_column = 'Outstanding Debt'

#Get Details
column_info(df,column)

#Cleaning
feat_eng3_num_replace_undefinedVal(df, groupby, column, strip='_',datatype=float)


#Plot Graph
plot_displot(df,'Outstanding_Debt', 'Outstanding Debt', rotation=90)
```

```
Details of Outstanding_Debt column

DataType:  object

There are no null values

Number of Unique Values:  13178

 Series of Unique Values:

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: Outstanding_Debt, Length: 13178, dtype: int64

Trailing & leading _ are removed

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

 Min, Max Values:
min       0.23
max    4998.07
Name: Outstanding_Debt, dtype: float64
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
 After data Cleaning Min, Max Values:
min       0.23
max    4998.07
Name: Outstanding_Debt, dtype: float64

 No. of Unique values after Cleaning: 12203

 No. of Null values after Cleaning: 0

Outstanding Debt Distribution
```



b: Monthly_Inhand_Salary

In [306…

```
column = 'Monthly_Inhand_Salary'
groupby = 'Customer_ID'

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

```
Details of Monthly_Inhand_Salary column

DataType:  float64

There are  15002  null values

Number of Unique Values:  13235

 Series of Unique Values:

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: Monthly_Inhand_Salary, Length: 13235, dtype: int64
```

In [307…

```
#Cleaning
feat_eng3_num_replace_undefinedVal(df, groupby, column)
```

```
 Min, Max Values:
min      303.645417
max    15204.633330
Name: Monthly_Inhand_Salary, dtype: float64
```

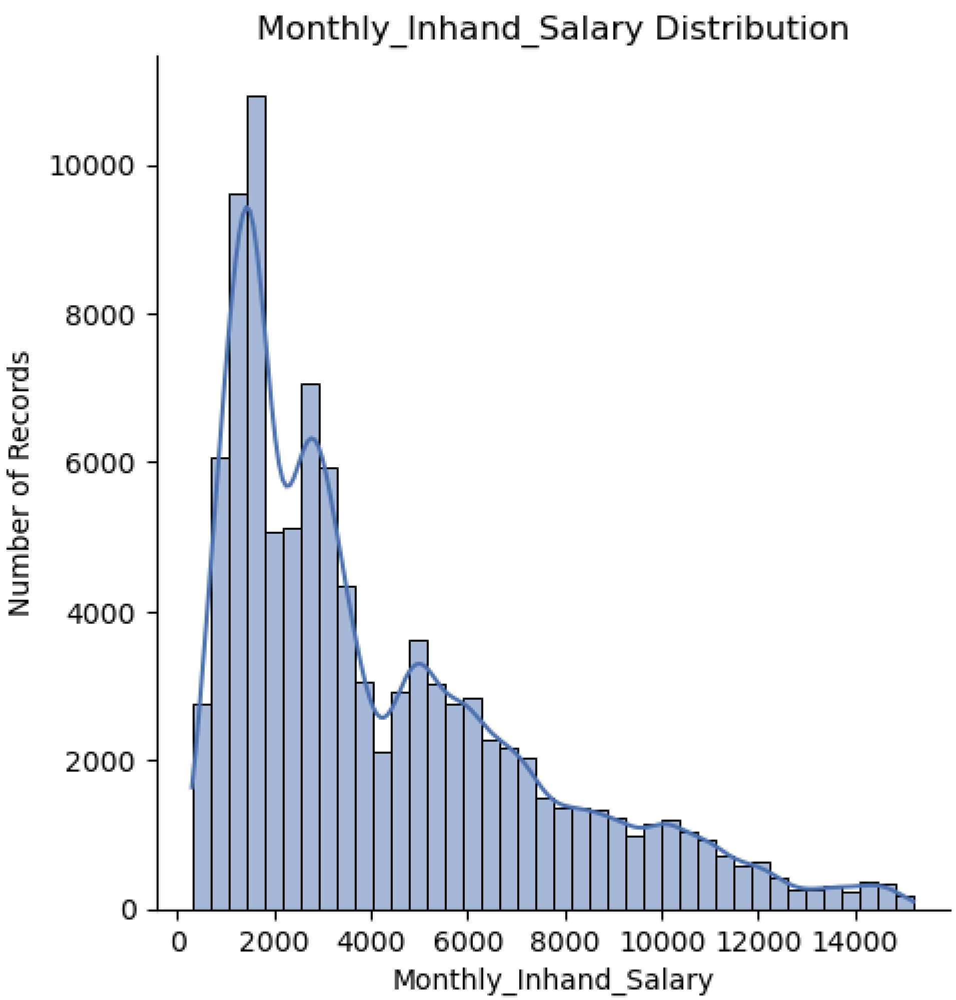
```
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
```

```
 After data Cleaning Min, Max Values:
min      303.645417
max    15204.633330
Name: Monthly_Inhand_Salary, dtype: float64

 No. of Unique values after Cleaning: 13235

 No. of Null values after Cleaning: 0
```

In [308…

```
edited_column = 'Monthly_Inhand_Salary'
plot_displot(df, column, edited_column, bins=40)
```

Monthly_Inhand_Salary Distribution



In [309…

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

In [310…

```
df[['Monthly_Debt_to_Income_Ratio']]
```

Out[310]:

	Monthly_Debt_to_Income_Ratio
0	0.443863
1	0.443863
2	0.443863
3	0.443863
4	0.443863
...	...
99995	0.149544
99996	0.149544
99997	0.149544
99998	0.149544
99999	0.149544

100000 rows × 1 columns

4. Data Preparation for Num_Credit_Card

In [311…

```
column = 'Num_Credit_Card'
edited_column = 'Number of Credit Card'

column_info(df,column)
```

```
Details of Num_Credit_Card column

DataType:  int64

There are no null values

Number of Unique Values:  1179

 Series of Unique Values:

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

In [312…

```
groupby = 'Customer_ID'
feat_eng3_num_replace_undefinedVal(df, groupby, column)
```

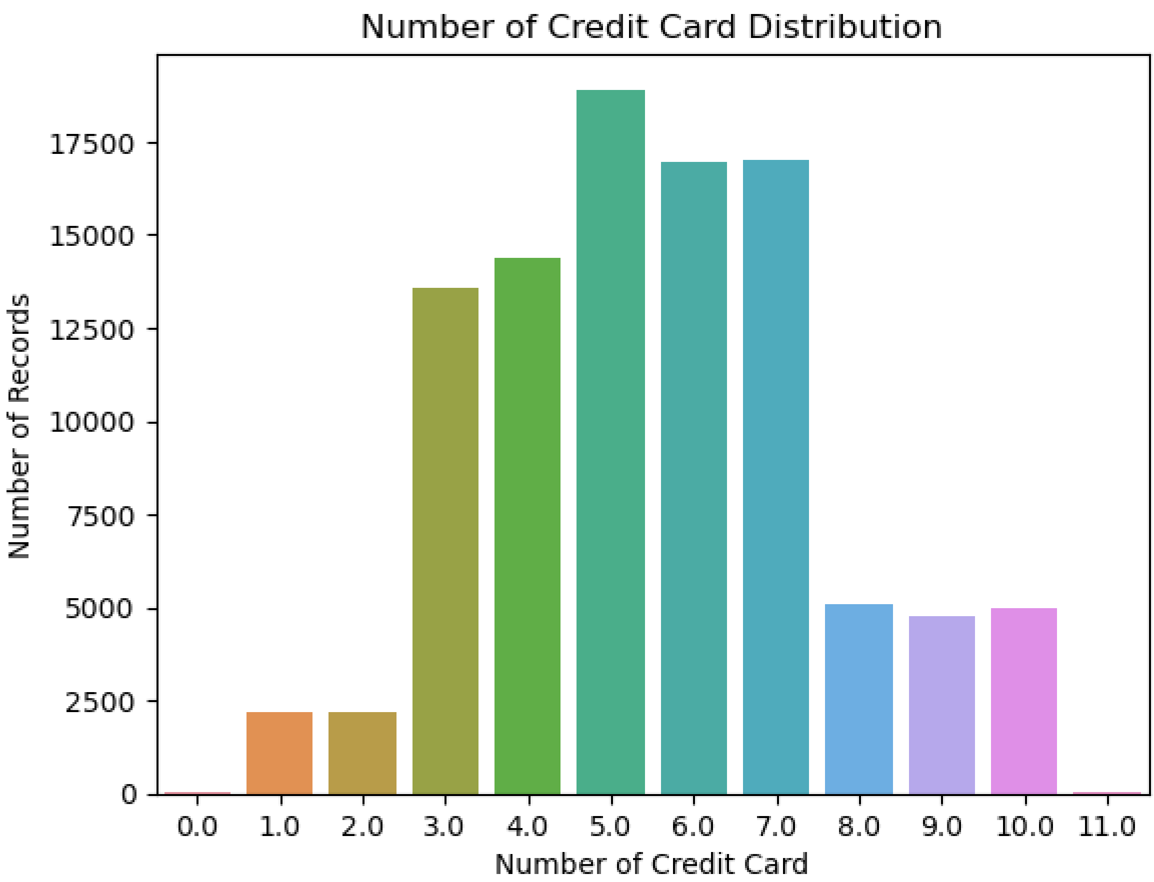
```
 Min, Max Values:
min       0
max    1499
Name: Num_Credit_Card, dtype: int64
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
 After data Cleaning Min, Max Values:
min      0.0
max     11.0
Name: Num_Credit_Card, dtype: float64

 No. of Unique values after Cleaning: 12

 No. of Null values after Cleaning: 0
```

In [313…

```
plot_countplot(df,column,edited_column)
```

```
Number of Credit Card Distribution
```



5. Data Preparation for Employment Status

** As In credit scoring models, employment status is often considered as a significant factor because it provides insights into an individual's financial stability and ability to repay debts. Individuals with stable employment are generally considered less risky borrowers. However, the given dataset doesn't have a specific 'Employment_Status' column, so we use the 'Occupation' column as a proxy for employment status

** 'Occupation' can serve as a proxy for employment status, as certain occupations are associated with stable employment (e.g., doctors, teachers) while others may indicate more variable or entrepreneurial sources of income.

In [314…

```
column = 'Occupation'
groupby = 'Customer_ID'

#getting details of Occupation
```

```
column_info(df,'Occupation')
```

```
Details of Occupation column

DataType:  object

There are no null values

Number of Unique Values:  16

 Series of Unique Values:

_______          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 [315…

```
replace_value = '_______'
#user_friendly_name = 'Occupation'

feat_eng4_cat_replace_with_null_mode(df, groupby, column, replace_value)
```

```
 Cleaning Categorical column:  Occupation

 No. of missing values before feature engineering:  7062

 No. of missing values after feature engineering:  0
```

In [316…

```
# Map occupation categories to employment status weights
employment_status_weights = {
    'Lawyer': 0.10,
    'Architect': 0.10,
    'Engineer': 0.10,
    'Scientist': 0.10,
    'Mechanic': 0.05,
    'Accountant': 0.10,
    'Developer': 0.10,
    'Media_Manager': 0.10,
    'Teacher': 0.10,
    'Entrepreneur': 0.10,
    'Doctor': 0.10,
    'Journalist': 0.05,
    'Manager': 0.10,
    'Musician': 0.05,
    'Writer': 0.05
}
```

In [317…

```
#df['Occupation'].map(employment_status_weights).fillna(0) * weights['Employment_Status']
# Calculate Employment_Status component
df['Employment_Status'] = df['Occupation'].map(employment_status_weights).fillna(0)
```

In [318…

```
# Display the resulting DataFrame
df[['Occupation', 'Employment_Status']]
```

Out[318]:

	Occupation	Employment_Status
0	Scientist	0.10
1	Scientist	0.10
2	Scientist	0.10
3	Scientist	0.10
4	Scientist	0.10
...	...	...
99995	Mechanic	0.05
99996	Mechanic	0.05
99997	Mechanic	0.05
99998	Mechanic	0.05
99999	Mechanic	0.05

100000 rows × 2 columns

6. Data Preparation for Credit History Age

In [319…

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

Out[319]:

```
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
                         ...
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: Credit_History_Age, Length: 404, dtype: int64
```

In [320…

```
def Month_Converter(val):
    if pd.notnull(val):
        years = int(val.split(' ')[0])
        month = int(val.split(' ')[3])
        return (years*12)+month
    else:
        return val
```

In [321…

```
df['Credit_History_Age'] = df['Credit_History_Age'].apply(lambda x: Month_Converter(x)).astyp
```

In [322…

```
df[['Credit_History_Age']]
```

Out[322]:

	Credit_History_Age
0	265.0
1	NaN
2	267.0
3	268.0
4	269.0
...	...
99995	378.0
99996	379.0
99997	380.0
99998	381.0
99999	382.0

100000 rows × 1 columns

In [323…

```
column = 'Credit_History_Age'

column_info(df,column)
```

```
Details of Credit_History_Age column

DataType:  float64

There are  9030  null values

Number of Unique Values:  404

 Series of Unique Values:

191.0    446
232.0    445
233.0    444
215.0    443
231.0    441
        ...
3.0       20
2.0       15
403.0     14
404.0     12
1.0        2
Name: Credit_History_Age, Length: 404, dtype: int64
```

In [324…

```
groupby = 'Customer_ID'
feat_eng3_num_replace_undefinedVal(df, groupby, column, datatype=float)
```

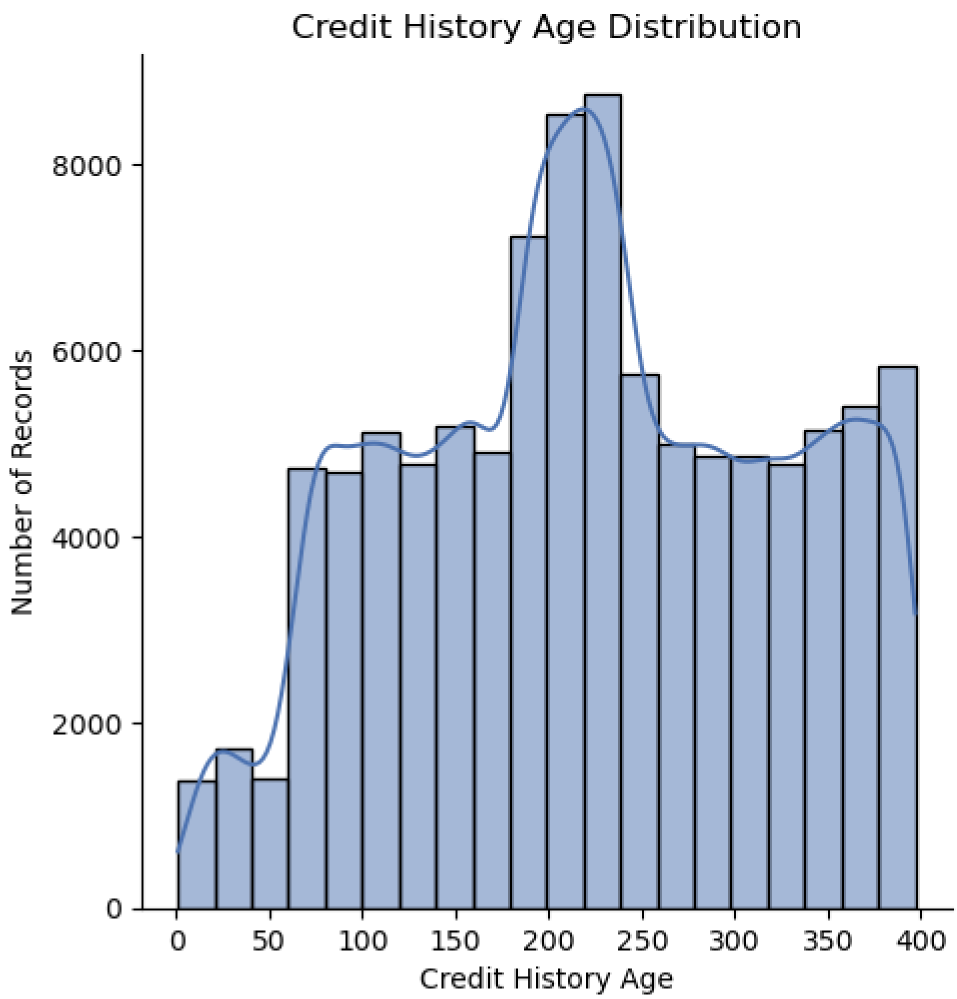
```
Datatype of Credit_History_Age is changed to <class 'float'>

 Min, Max Values:
min      1.0
max    404.0
Name: Credit_History_Age, dtype: float64
C:\Users\hp\AppData\Local\Temp\ipykernel_24568\1175568200.py:5: FutureWarning: Unlike other r
eduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preser
ves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken will be eliminate
d, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid th
is warning.
  x, y = df_dropped.apply(lambda x: stats.mode(x)).apply([min, max])
```

```
 After data Cleaning Min, Max Values:
min       1.0
max     397.0
Name: Credit_History_Age, dtype: float64

 No. of Unique values after Cleaning: 397

 No. of Null values after Cleaning: 0
```

In [325…

```
edited_column = 'Credit History Age'
#Plot Graph
plot_displot(df, column, edited_column)
```

```
Credit History Age Distribution
```



In [326…

```
df[['Payment_History_Score', 'Credit_Utilization_Ratio', 'Monthly_Debt_to_Income_Ratio', 'Num
```

Out[326]:

	Payment_History_Score	Credit_Utilization_Ratio	Monthly_Debt_to_Income_Ratio	Num_Credit_Card	Empl
0	-10.0	26.822620	0.443863	4.0	
1	-7.0	31.944960	0.443863	4.0	
2	-10.0	28.609352	0.443863	4.0	
3	-9.0	31.377862	0.443863	4.0	
4	-10.0	24.797347	0.443863	4.0	
...	...	...	...	...	
99995	-30.0	34.663572	0.149544	6.0	
99996	-25.0	40.565631	0.149544	6.0	
99997	-33.0	41.255522	0.149544	6.0	
99998	-26.0	33.638208	0.149544	6.0	
99999	-24.0	34.192463	0.149544	6.0	

100000 rows × 6 columns

Calcuting Credit Score

1. Group by Customer ID, handling month-level data and calculating scores

2. Standardize values for numerical features

3. Calculate weighted scores

4. Normalize scores to a range of 0 to 100

In [327…

```python
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"),
    Credit_Utilization_Ratio=("Credit_Utilization_Ratio", "mean"),
    Monthly_Debt_to_Income_Ratio=("Monthly_Debt_to_Income_Ratio", "mean"),
    Num_Credit_Card=("Num_Credit_Card", "mean"),
    Employment_Status=("Employment_Status", "mean"),
    Credit_History_Age=("Credit_History_Age", "max"),  # Using maximum age as it seems to be
    )


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

    # Calculate weighted scores
    grouped_data["credit_score"] = (
      0.35 * grouped_data["Payment_History_Score"]
      + 0.15 * (1-grouped_data["Monthly_Debt_to_Income_Ratio"]) #Inverse relation as lower th
      + 0.15 * (1-grouped_data["Credit_Utilization_Ratio"]) #inverse relation
      + 0.15 * (grouped_data["Num_Credit_Card"])
      + 0.10 * grouped_data["Employment_Status"]
      + 0.10 * grouped_data["Credit_History_Age"]
     )

     # Normalize scores to a range of 0 to 100
    grouped_data["credit_score"] = (grouped_data["credit_score"] - grouped_data["credit_score
```

```python
    return grouped_data.reset_index()
```

In [328…

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

Out[328]:

	Customer_ID	credit_score
0	CUS_0x1000	33.641503
1	CUS_0x1009	76.667166
2	CUS_0x100b	67.038379
3	CUS_0x1011	69.160474
4	CUS_0x1013	64.932873
...	...	...
12495	CUS_0xff3	72.232778
12496	CUS_0xff4	74.823735
12497	CUS_0xff6	86.485154
12498	CUS_0xffc	46.146841
12499	CUS_0xffd	72.326264

12500 rows × 2 columns

In [329…

```python
# Assuming 'result_df' is your DataFrame with the calculated credit scores
# Adjust the percentile values as needed
good_threshold = credit_scores_df['credit_score'].quantile(0.75)
poor_threshold = credit_scores_df['credit_score'].quantile(0.25)

# Create a new column 'Credit_Score_Category'
credit_scores_df['Credit_Score_Category'] = pd.cut(
    credit_scores_df['credit_score'],
    bins=[-float('inf'), poor_threshold, good_threshold, float('inf')],
    labels=['Poor', 'Standard', 'Good'],
    include_lowest=True
)
```

In [330…

```python
credit_scores_df
```

Out[330]:

	Customer_ID	Payment_History_Score	Credit_Utilization_Ratio	Monthly_Debt_to_Income_Ratio	Num_Cred
0	CUS_0x1000	-2.768543	0.578666	-0.148142	-0
1	CUS_0x1009	0.517785	-1.186664	-0.612642	-0
2	CUS_0x100b	0.699239	1.240576	-0.559874	-0
3	CUS_0x1011	-0.382763	-2.246615	-0.574815	-1
4	CUS_0x1013	0.692518	-0.170455	-0.518636	-1
...	...	...	...	...	
12495	CUS_0xff3	0.699239	0.293234	0.261471	0
12496	CUS_0xff4	0.363213	0.151942	-0.379225	0
12497	CUS_0xff6	1.552743	0.472144	-0.623956	0
12498	CUS_0xffc	-1.437883	1.182659	-0.436019	1
12499	CUS_0xffd	0.020467	-0.189711	-0.256574	0

12500 rows × 9 columns

In [331…

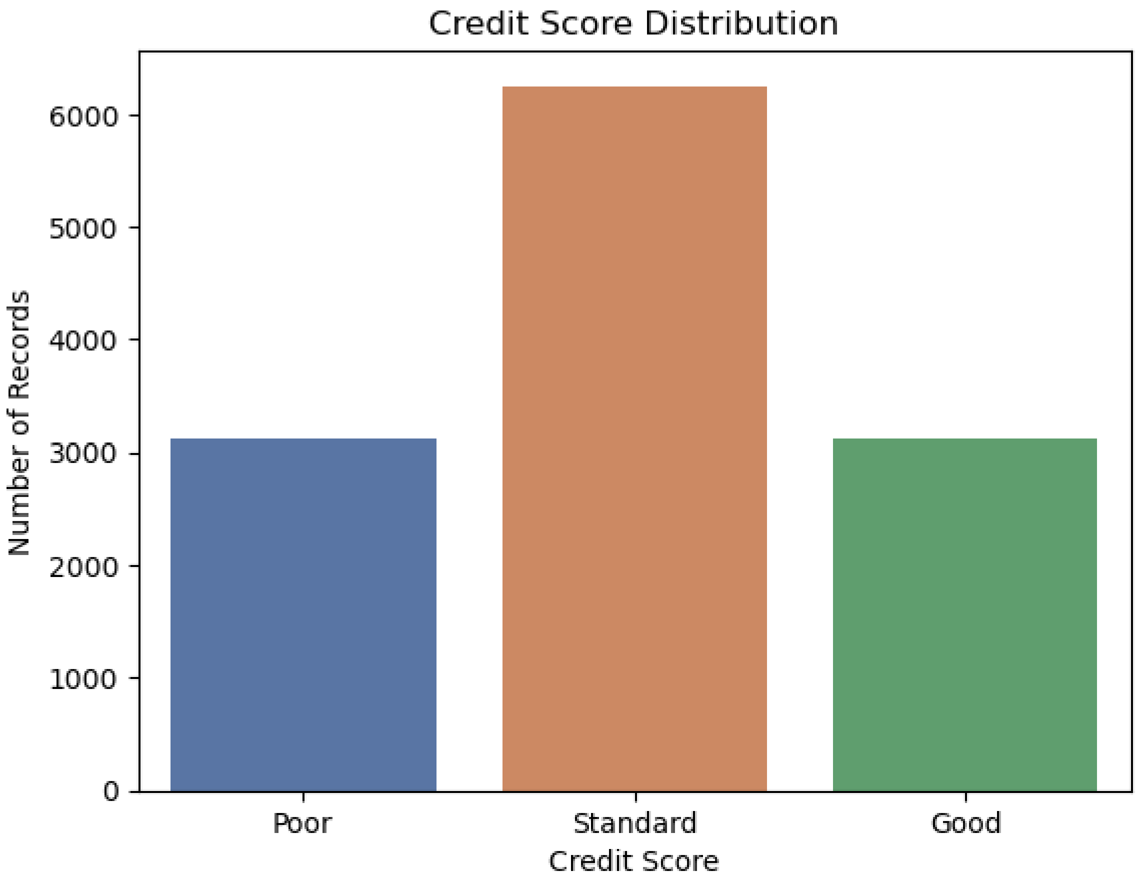
```
column = 'Credit_Score_Category'
edited_column = 'Credit Score'

#Get Details
column_info(credit_scores_df,column)
```

```
Details of Credit_Score_Category column

DataType:  category

There are no null values

Number of Unique Values:  3

 Series of Unique Values:

Standard    6250
Poor        3125
Good        3125
Name: Credit_Score_Category, dtype: int64
```

In [332…

```
#Plot Graph
plot_countplot(credit_scores_df,column,edited_column)
```

```
Credit Score Distribution
```



Summary

Insights from the dataset reveal that individual customer data is available for an 8-month period spanning from January to August. The dataset includes various loan types, such as auto loans, credit-builder loans, debt consolidation loans, home equity loans, mortgage loans, payday loans, personal loans, and student loans.

A notable trend is observed in the customers' annual income, which predominantly exhibits a right-skewed distribution, indicating that most customers have lower annual incomes.

Furthermore, the analysis of monthly income distribution follows a similar pattern, with a predominant right-skewed trend among customers. Regarding the number of bank accounts, the majority of customers maintain between 3 to 8 accounts. The distribution of the number of credit cards spans from 0 to 11, with a concentration between 3 to 7, peaking at 5.

Interest rates on loans vary across the dataset, ranging from 1% to 34%. The delay from the due date is observed to be concentrated within the 0 to 30 days range. Notably, only a limited number of customers invest amounts exceeding 2,000 per month. When it comes to the number of loans taken by customers, the typical range falls between 2 to 4 loans, with a maximum recorded at 9. These insights offer a comprehensive overview of the financial dynamics and behaviors observed in the dataset.

Advantage/Impact of Credit Score:

The credit score serves as a vital tool in further financial analysis, providing a concise measure of an individual's creditworthiness. Lenders commonly use credit scores to assess the risk associated with extending credit, determining interest rates, and making lending decisions. A higher credit score often leads to more favorable loan terms, lower interest rates, and increased access to financial products. Additionally, a good credit score can positively influence various aspects of financial life, such as securing housing, obtaining favorable insurance rates, and even impacting employability in certain industries.

Therefore, a well-calculated credit score contributes significantly to informed decision-making in financial and lending domains.

```
In [ ]:
```