PACKAGE IMPORTING

In [ ]:
```
%pip install datasist
#import relevant libraries and frameworks
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as plb
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# Data Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PowerTransformer
from datasist.structdata import detect_outliers
from sklearn.metrics import mean_squared_error
from imblearn.over_sampling import SMOTE
from sklearn.impute import SimpleImputer
import category_encoders as ce
import re

# Modeling and evaluation
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import (
    BaggingClassifier,
    ExtraTreesClassifier,
    RandomForestClassifier,
    StackingClassifier,
    HistGradientBoostingClassifier
)
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
import joblib
```

In [ ]:
```
# Packages options
sns.set(rc={'figure.figsize': [14, 7]}, font_scale=1.2) # Standard figure size f
np.seterr(divide='ignore', invalid='ignore', over='ignore') ;

import warnings
warnings.filterwarnings("ignore")
```

LOADING THE DATA (TRAINING AND VALIDATION/TESTING)

In [ ]:
```
#import data(train and test)
train = pd.read_csv('train.csv', low_memory=False)
test = pd.read_csv('test.csv', low_memory=False)
```

In [ ]:
```
train.head()
```

Out[ ]:

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

5 rows × 28 columns

In [ ]:
```
test.head()
```

Out[ ]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income
0	0x160a	CUS_0xd40	September	Aaron Maashoh	23	821-00-0265	Scientist	19114.12
1	0x160b	CUS_0xd40	October	Aaron Maashoh	24	821-00-0265	Scientist	19114.12
2	0x160c	CUS_0xd40	November	Aaron Maashoh	24	821-00-0265	Scientist	19114.12
3	0x160d	CUS_0xd40	December	Aaron Maashoh	24_	821-00-0265	Scientist	19114.12
4	0x1616	CUS_0x21b1	September	Rick Rothackerj	28	004-07-5839	_______	34847.84

5 rows × 27 columns

DESCRIPTIVE STATISTICS

In [ ]:
```
#check the shape of the train and test data
print(train.shape, test.shape)
```

```
(100000, 28) (50000, 27)
```

For the test set all variables are present except the target column

In [ ]:
```
#check the info of the train and test data
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 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 
 27  Credit_Score              100000 non-null  object 
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

In [ ]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 27 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   ID                        50000 non-null  object 
 1   Customer_ID               50000 non-null  object 
 2   Month                     50000 non-null  object 
 3   Name                      44985 non-null  object 
 4   Age                       50000 non-null  object 
 5   SSN                       50000 non-null  object 
 6   Occupation                50000 non-null  object 
 7   Annual_Income             50000 non-null  object 
 8   Monthly_Inhand_Salary     42502 non-null  float64
 9   Num_Bank_Accounts         50000 non-null  int64  
 10  Num_Credit_Card           50000 non-null  int64  
 11  Interest_Rate             50000 non-null  int64  
 12  Num_of_Loan               50000 non-null  object 
 13  Type_of_Loan              44296 non-null  object 
 14  Delay_from_due_date       50000 non-null  int64  
 15  Num_of_Delayed_Payment    46502 non-null  object 
 16  Changed_Credit_Limit      50000 non-null  object 
 17  Num_Credit_Inquiries      48965 non-null  float64
 18  Credit_Mix                50000 non-null  object 
 19  Outstanding_Debt          50000 non-null  object 
 20  Credit_Utilization_Ratio  50000 non-null  float64
 21  Credit_History_Age        45530 non-null  object 
 22  Payment_of_Min_Amount     50000 non-null  object 
 23  Total_EMI_per_month       50000 non-null  float64
 24  Amount_invested_monthly   47729 non-null  object 
 25  Payment_Behaviour         50000 non-null  object 
 26  Monthly_Balance           49438 non-null  object 
dtypes: float64(4), int64(4), object(19)
memory usage: 10.3+ MB
```

In [ ]:

```
#check the summary of the train data
train.describe()
```

Out[ ]:

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	D
count	84998.000000	100000.000000	100000.00000	100000.000000	
mean	4194.170850	17.091280	22.47443	72.466040	
std	3183.686167	117.404834	129.05741	466.422621	
min	303.645417	-1.000000	0.00000	1.000000	
25%	1625.568229	3.000000	4.00000	8.000000	
50%	3093.745000	6.000000	5.00000	13.000000	
75%	5957.448333	7.000000	7.00000	20.000000	
max	15204.633333	1798.000000	1499.00000	5797.000000	

In [ ]:

```
#check the summary of the train data
test.describe()
```

Out[ ]:

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	De
count	42502.000000	50000.000000	50000.000000	50000.000000	
mean	4182.004291	16.838260	22.921480	68.772640	
std	3174.109304	116.396848	129.314804	451.602363	
min	303.645417	-1.000000	0.000000	1.000000	
25%	1625.188333	3.000000	4.000000	8.000000	
50%	3086.305000	6.000000	5.000000	13.000000	
75%	5934.189094	7.000000	7.000000	20.000000	
max	15204.633333	1798.000000	1499.000000	5799.000000	

In [ ]:

```
#check for duplicates in the train and test data
train.duplicated().sum(), test.duplicated().sum()
```

Out[ ]: (0, 0)

EXPLORATIVE DATA ANALYSIS

In [ ]:

```
train.head()
```

Out[ ]:

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

5 rows × 28 columns

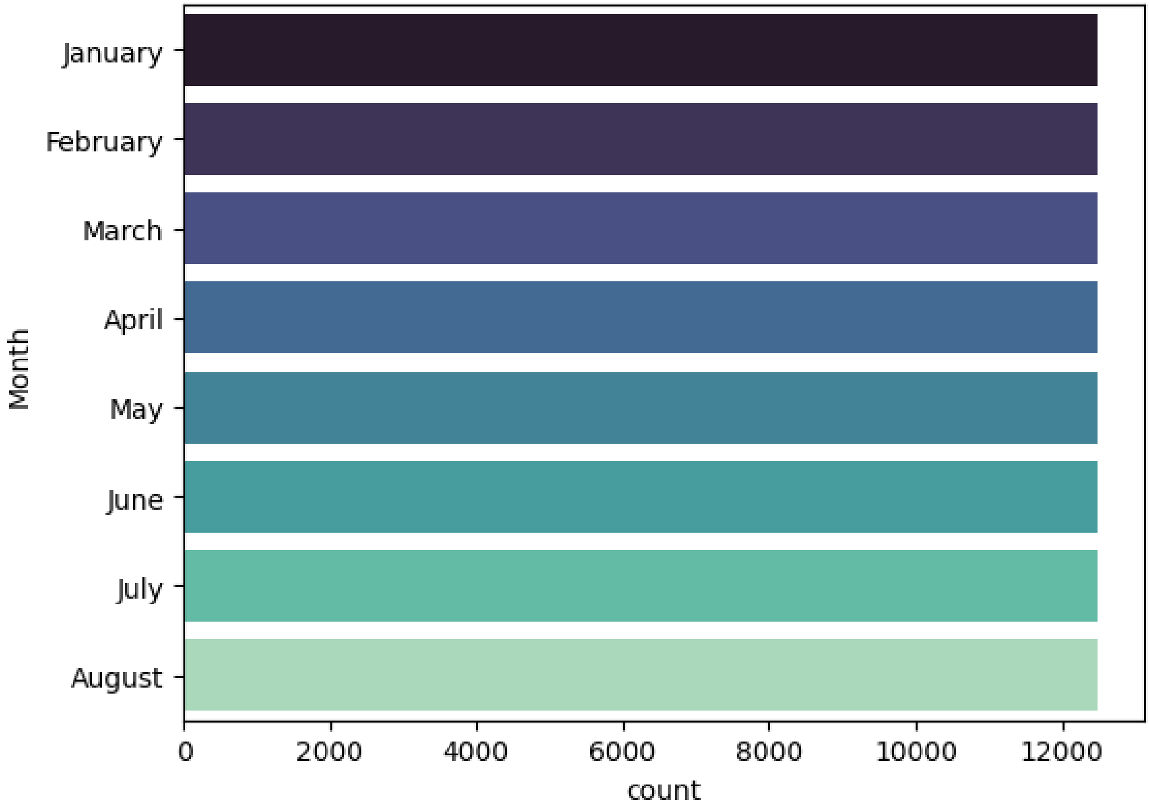
In [ ]:

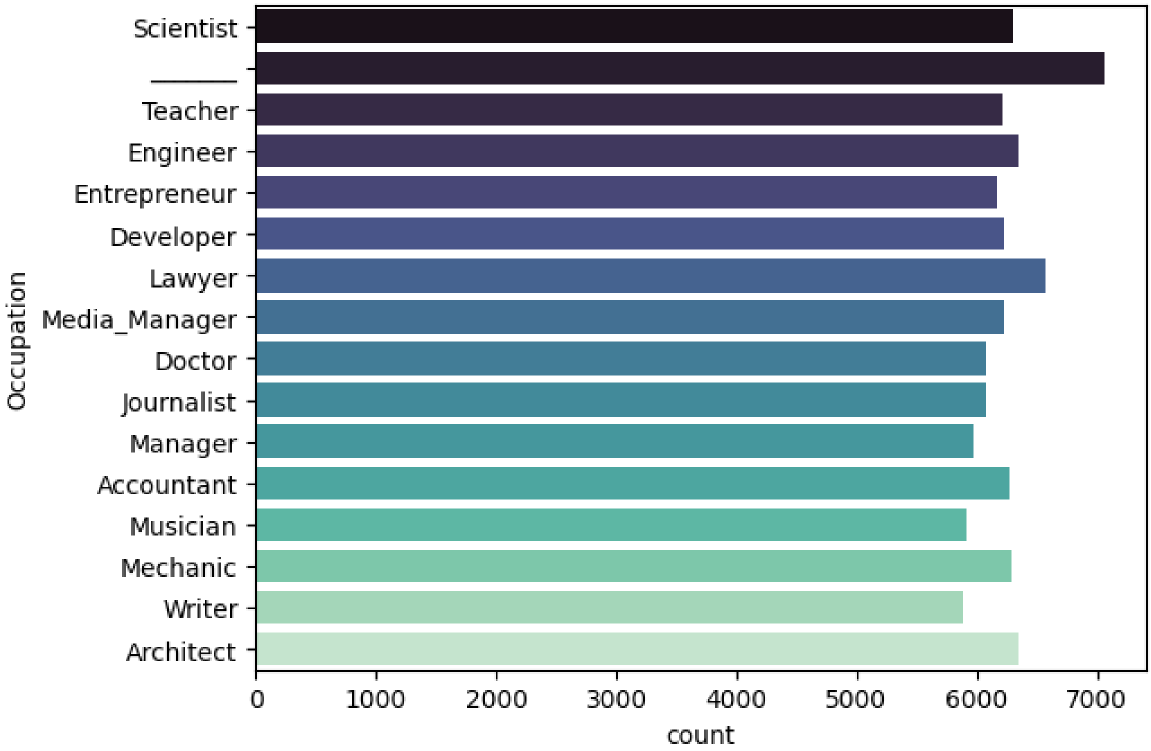
```
train.columns
```

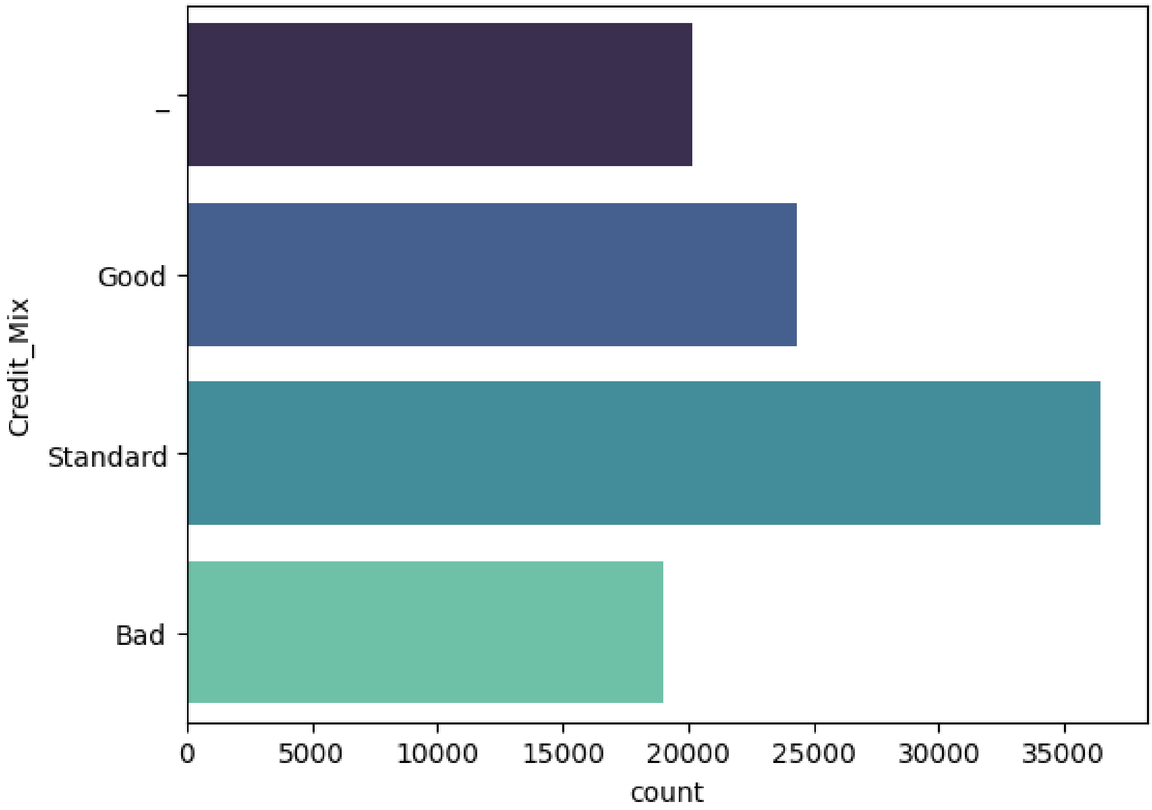
```
Out[ ]: Index(['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
               'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
               'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
               'Delay_from_due_date', 'Num_of_Delayed_Payment', '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',
               'Credit_Score'],
              dtype='object')
```

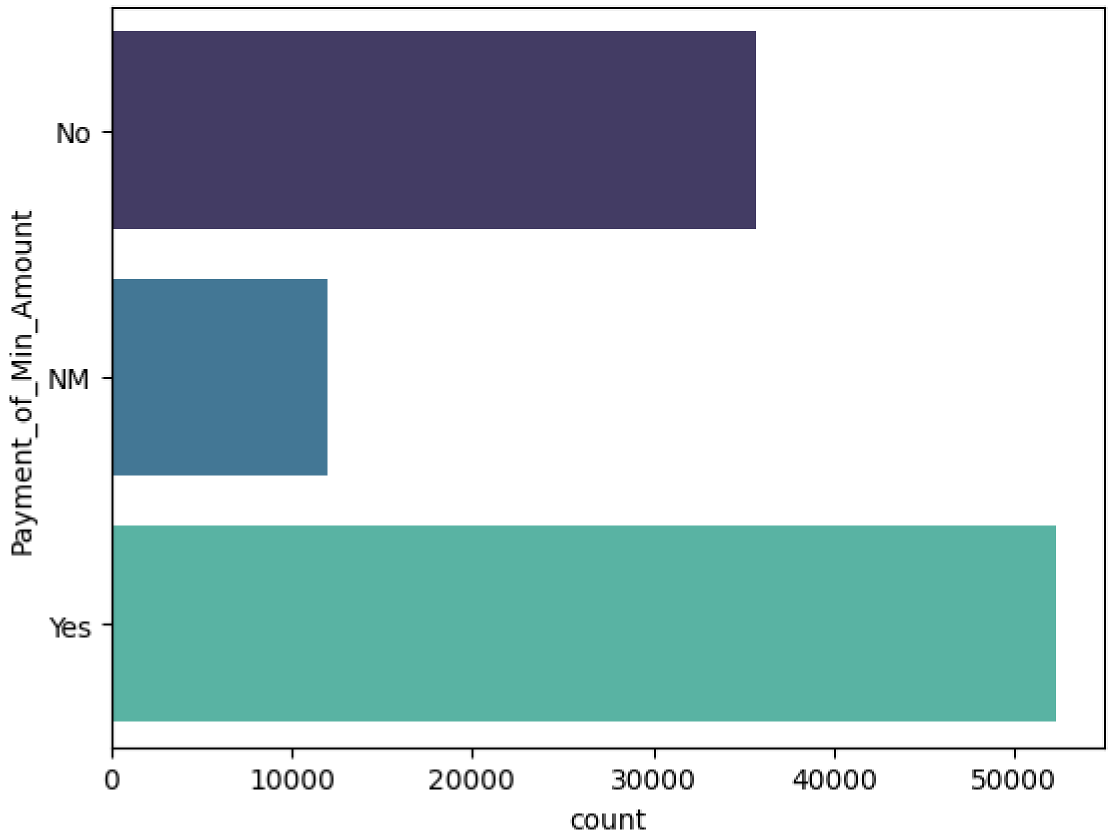
```
In [ ]: categ_cols = ['Month','Occupation','Credit_Mix','Payment_of_Min_Amount', 'Paymen
```

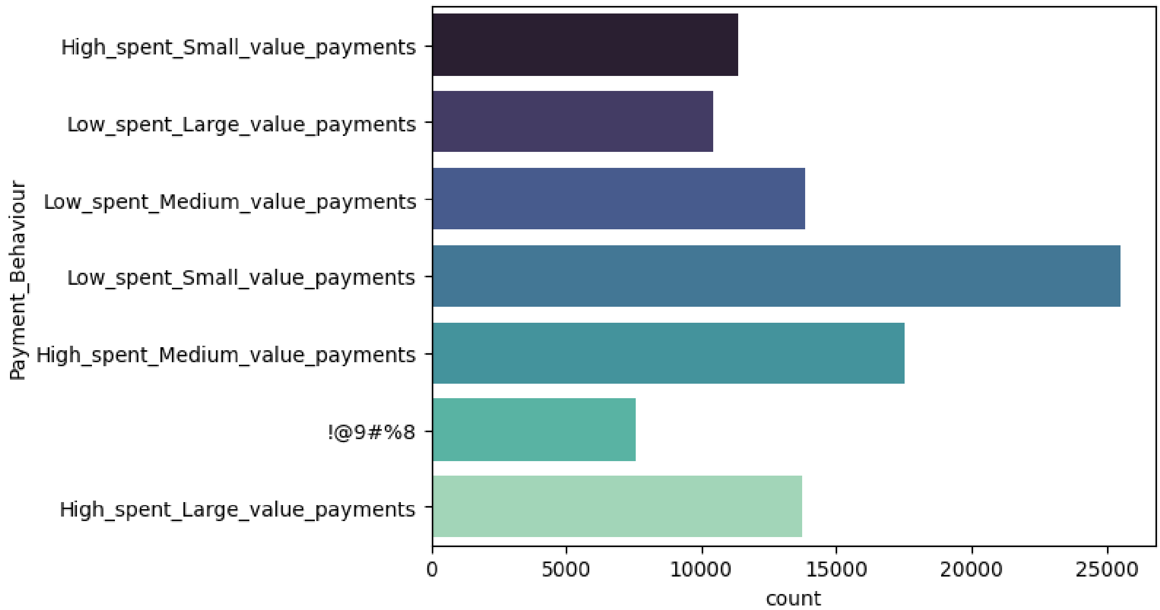
```
In [ ]: #create a for loop to plot a countplot of all categorical column in the train da
        for i in categ_cols:
            sns.countplot(train[i], palette='mako')
            plt.show()
```



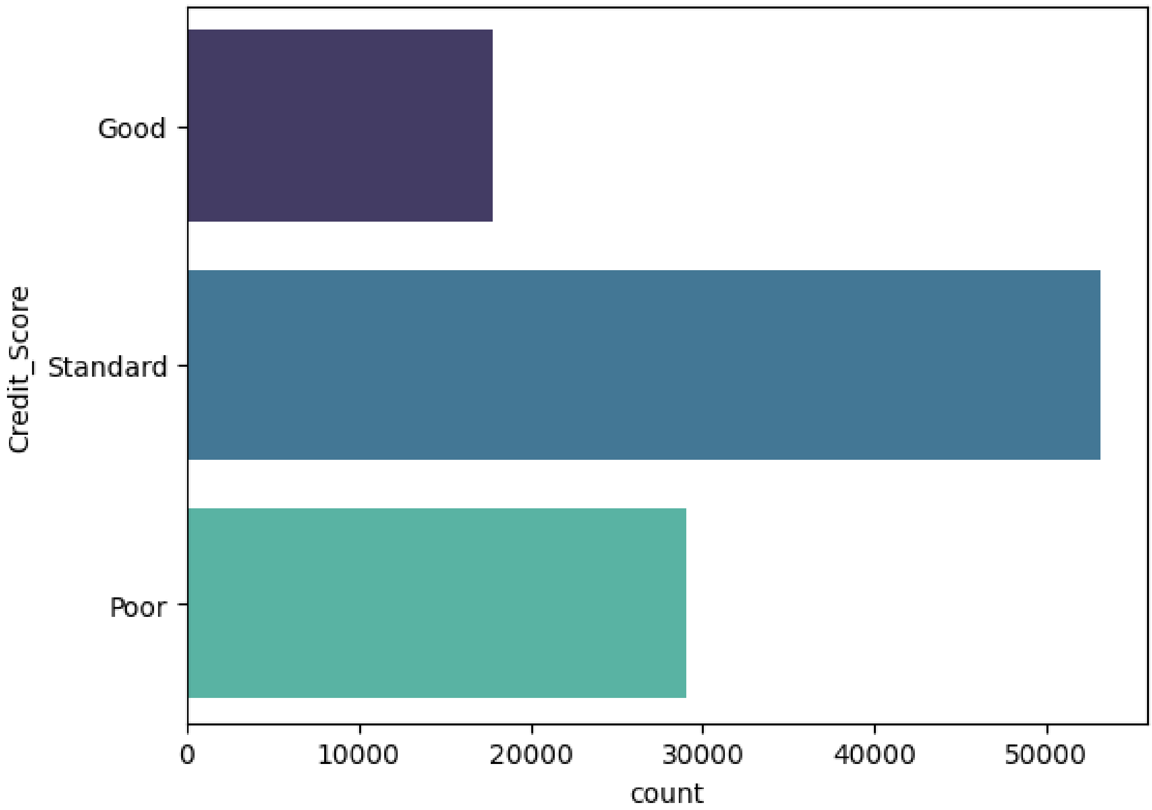

Scientist
____
Teacher
Engineer
Entrepreneur
Developer
Lawyer
Media_Manager
Doctor
Journalist
Manager
Accountant
Musician
Mechanic
Writer
Architect
Occupation
0
1000
2000
3000
4000
5000
6000
7000
count


_
Good
Standard
Bad
Credit_Mix
0
5000
10000
15000
20000
25000
30000
35000
count





In [ ]:
```
#check the distribution of the target variable
sns.countplot(train['Credit_Score'], palette='mako')
plt.show()
```



In [ ]:
```
#check the normalized count of the target variable
train['Credit_Score'].value_counts(normalize=True)
```

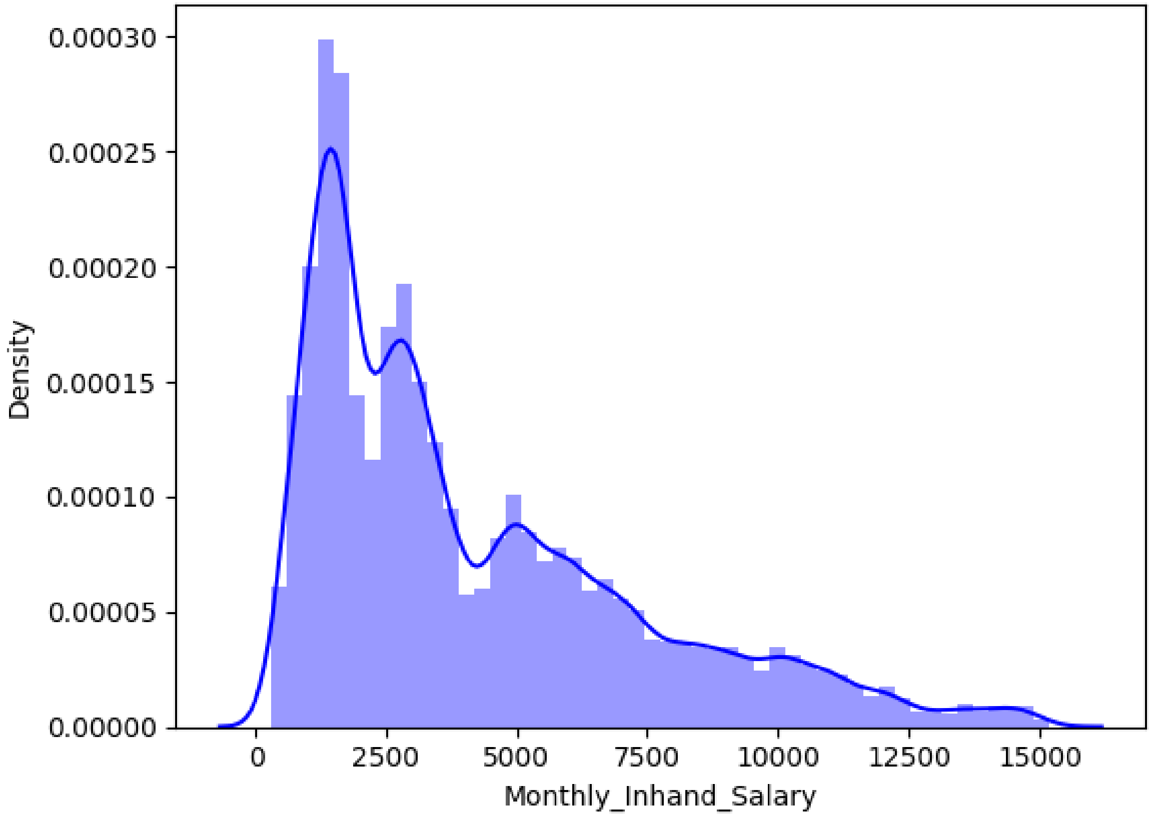
Out[ ]:
```
Credit_Score
Standard    0.53174
Poor        0.28998
Good        0.17828
Name: proportion, dtype: float64
```

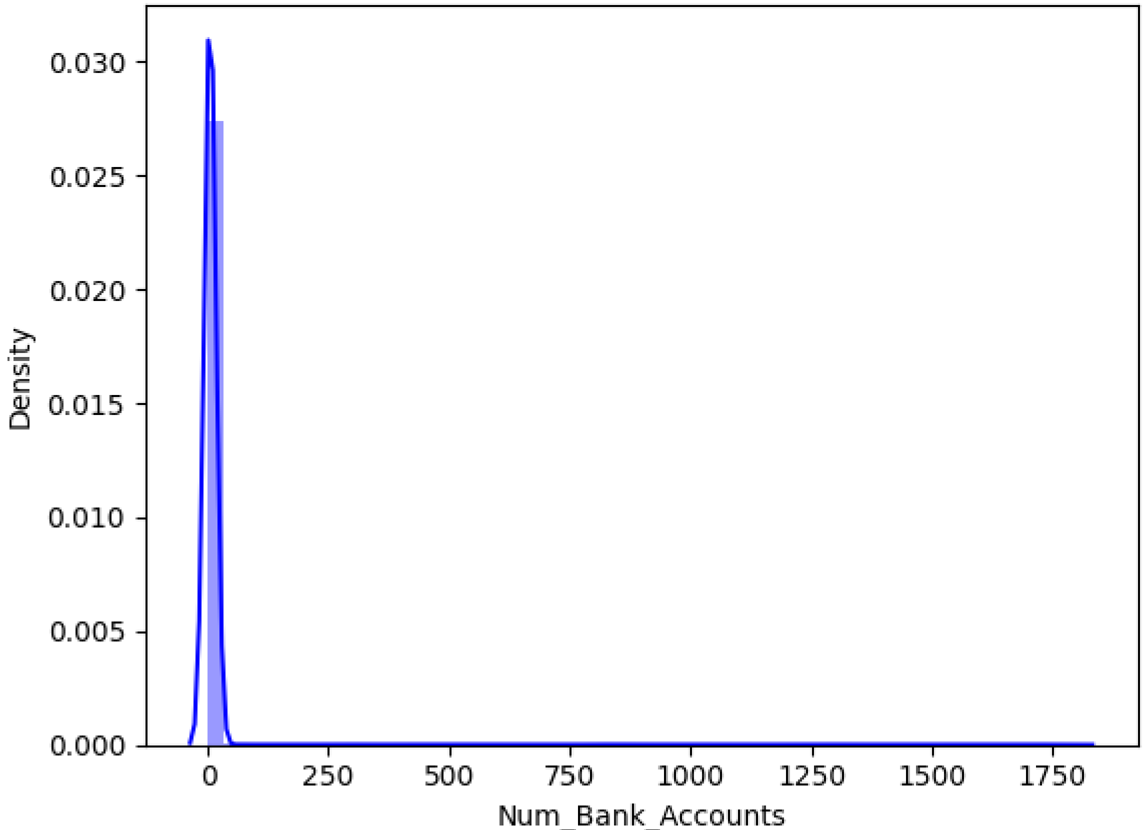
In [ ]:
```
train.columns
```

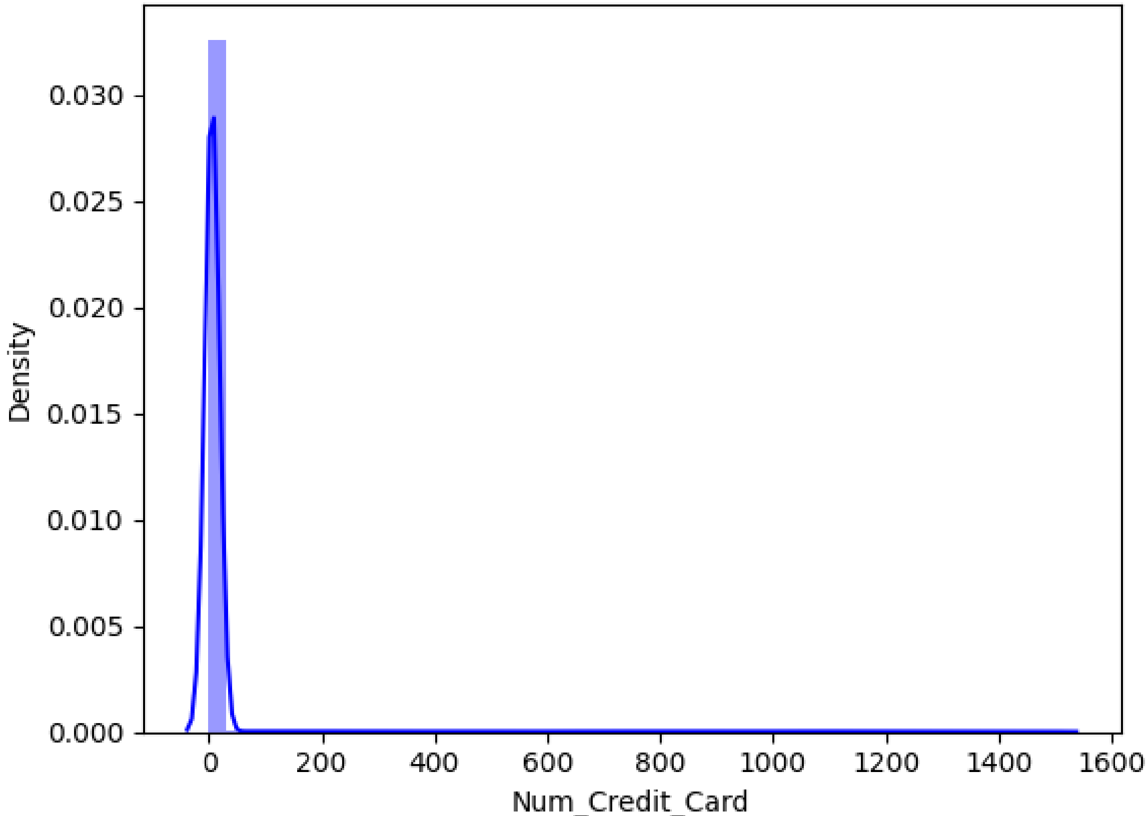
Out[ ]:
```
Index(['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       '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',
       'Credit_Score'],
      dtype='object')
```

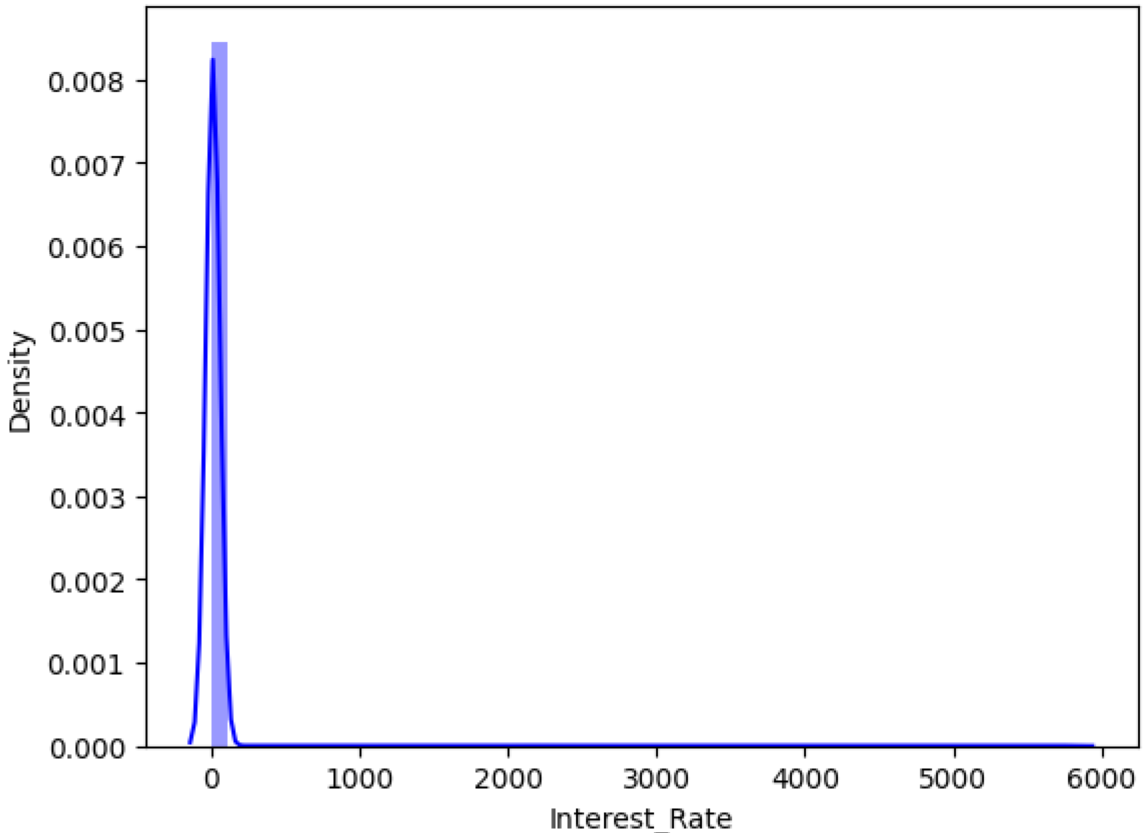
In [ ]:
```
dist_cols = ['Monthly_Inhand_Salary', 'Num_Bank_Accounts', 'Num_Credit_Card', 'I
```

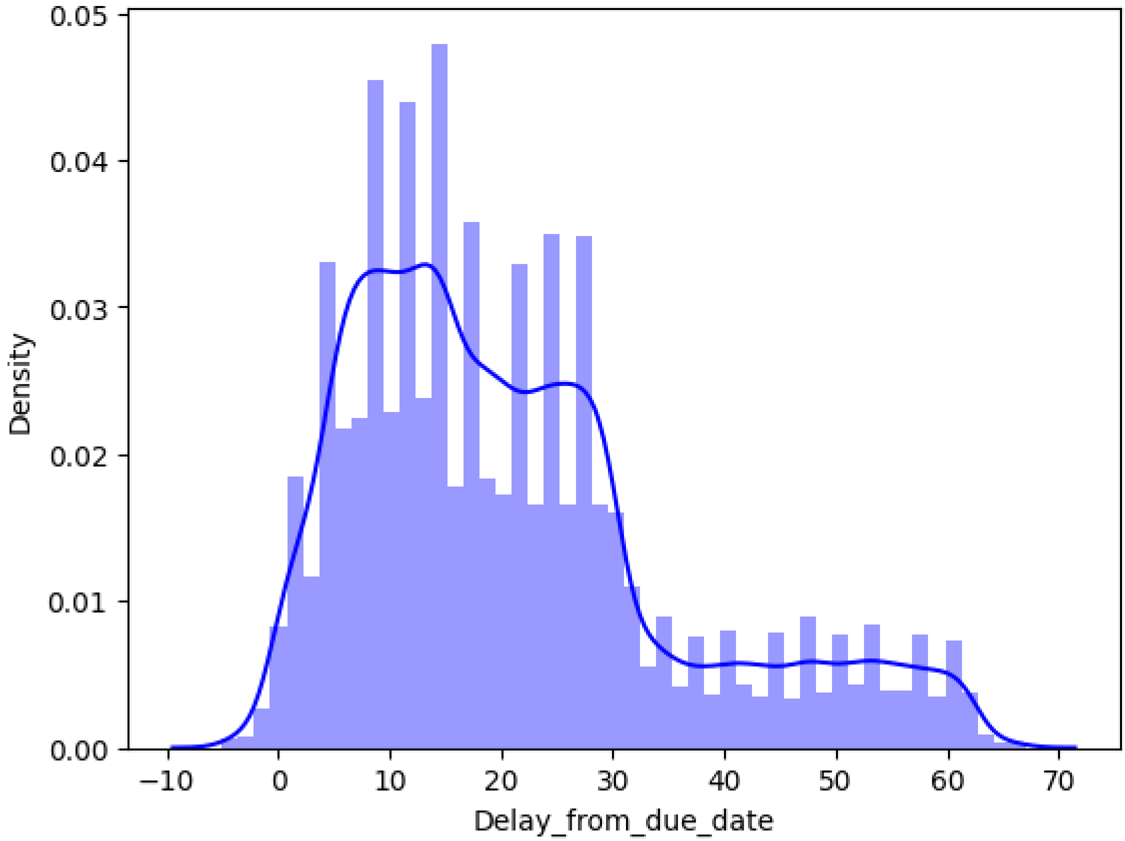
In [ ]:
```
#create a for loop to plot a distplot of all numerical column in the train data
for i in dist_cols:
    sns.distplot(train[i], color='blue')
    plt.show()
```


0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
2500
5000
7500
10000
12500
15000
Monthly_Inhand_Salary

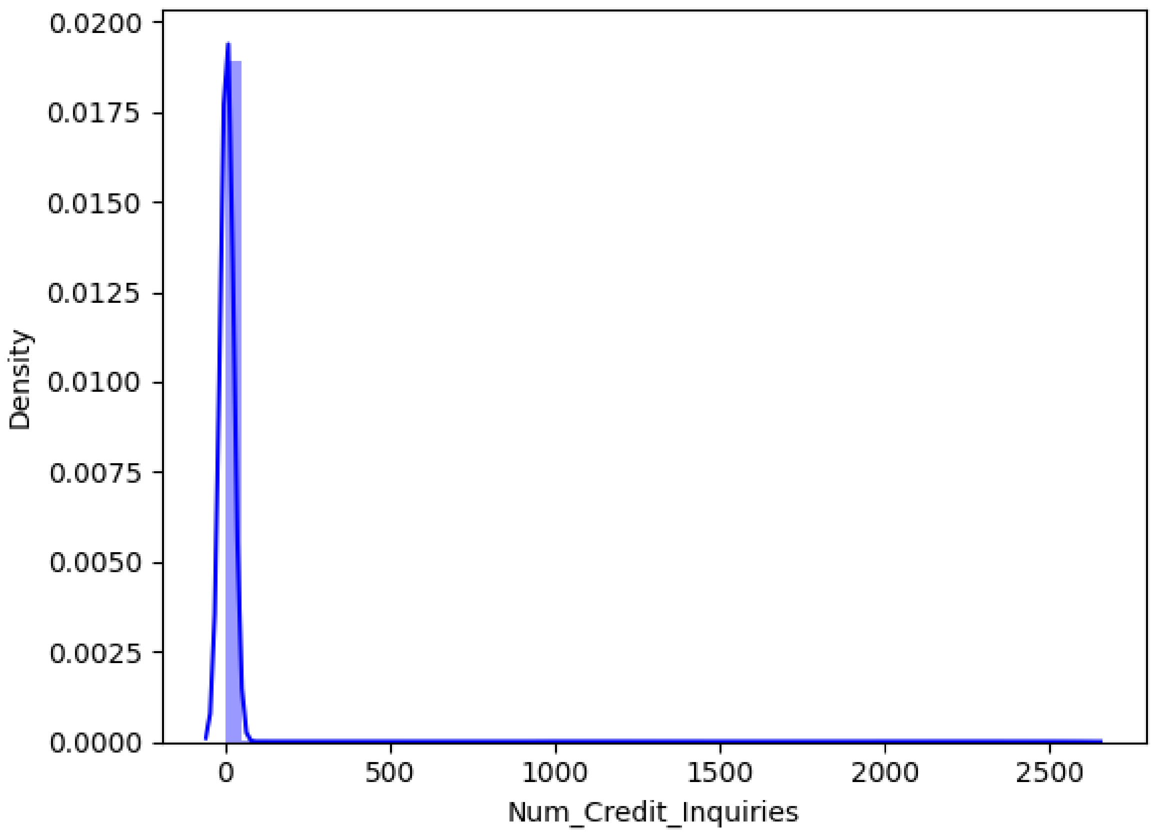

0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
0
250
500
750
1000
1250
1500
1750
Num_Bank_Accounts


0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
0
200
400
600
800
1000
1200
1400
1600
Num_Credit_Card

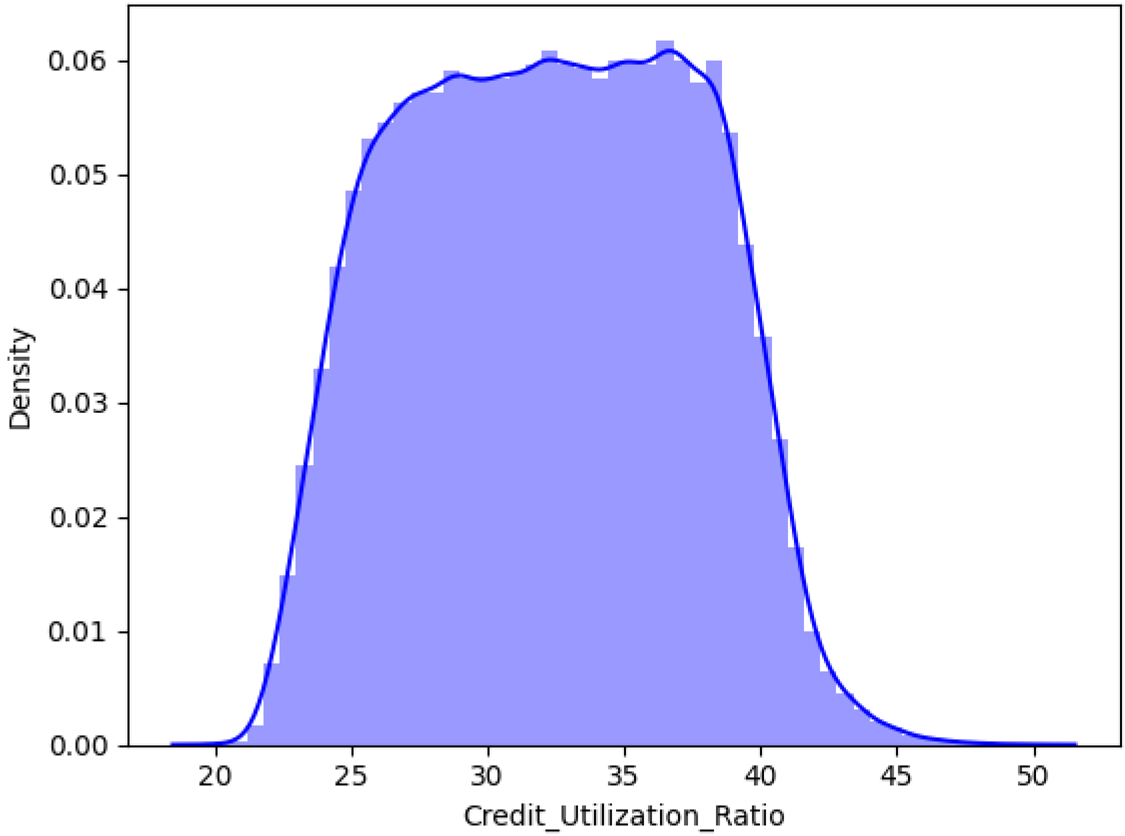

0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
0
1000
2000
3000
4000
5000
6000
Interest_Rate

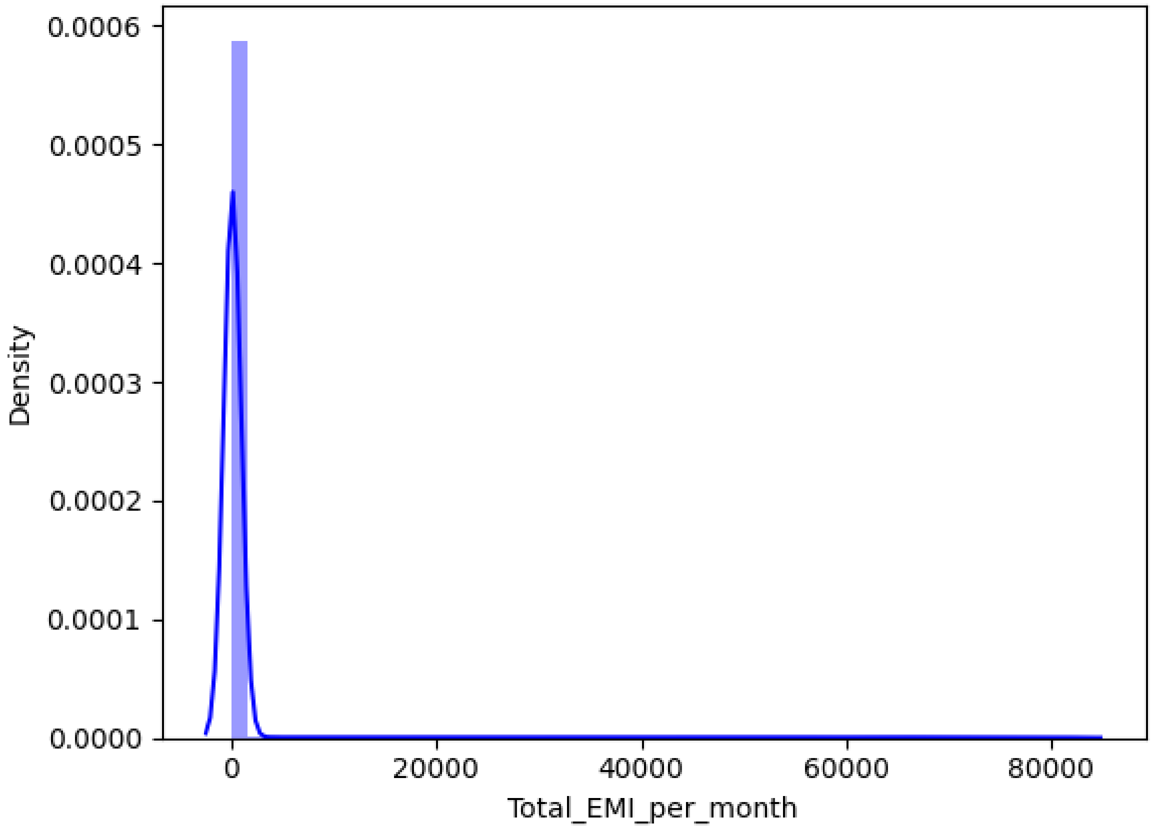


Density
Delay_from_due_date
0.00
0.01
0.02
0.03
0.04
0.05
−10
0
10
20
30
40
50
60
70



Density
Num_Credit_Inquiries
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
0
500
1000
1500
2000
2500





In [ ]:

```
#check the number of values in the 'Type_of_Loan' column
train['Type_of_Loan'].value_counts().head(10)
```

Out[ ]:
```
Type_of_Loan
Not Specified                          1408
Credit-Builder Loan                    1280
Personal Loan                          1272
Debt Consolidation Loan                1264
Student Loan                           1240
Payday Loan                            1200
Mortgage Loan                          1176
Auto Loan                              1152
Home Equity Loan                       1136
Personal Loan, and Student Loan         320
Name: count, dtype: int64
```

Identify issues

- ID, Name and SSN (Not useful)
- Age, Annual_Income, Num_of_Loan, Num_of_Delayed_Payment, Changed_Credit_Limit, Amount_invested_monthly, Outstanding_Debt Credit_Mix, Monthly_Balance Numerical but show as catogery (need to be fixed)
- Occupation, CreditMix has value "__"
- Data contains outliers
- Num_Credit_Card has zeros
- Type_of_Loan Need to rewrite as 8 columns
- Num_Bank_Accounts contains negative values
- Credit_History_Age,Payment_of_Min_Amount,Payment_Behaviour,'Credit_Mix' (needs Feature Engineering)
- Target Columns is Imbalanced
- A lot of missing data

Data Preprocessing

Removing the unnecessary columns (Unique Identifier)

In [ ]:
```
del train['ID'] # Identification
del train['Name'] # Name of client
del train['SSN'] # SSN (social security number of a person)
```

In [ ]:
```
train.head()
```

Out[ ]:

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

5 rows × 25 columns

Fixing the Numerical Column

- rename in a more better way
- place correct data type

In [ ]:

```
#create a list of column to be renamed and changing the datatype of the column
num_cols_to_fix = ['Age', 'Annual_Income', 'Num_of_Loan', 'Num_of_Delayed_Paymen

#create function to fix the rename the column
def fix_nums(num):
    try :
        return float(num.replace("_",""))
    except :
        return np.nan

#apply the function to the column and make a for loop to apply the function to a
for col in num_cols_to_fix :
    train[col] = train[col].apply(fix_nums)
```

In [ ]:

```
#recheck the dataset
train.head()
```

Out[ ]:

	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary
0	CUS_0xd40	January	23.0	Scientist	19114.12	1824.843333
1	CUS_0xd40	February	23.0	Scientist	19114.12	NaN
2	CUS_0xd40	March	-500.0	Scientist	19114.12	NaN
3	CUS_0xd40	April	23.0	Scientist	19114.12	NaN
4	CUS_0xd40	May	23.0	Scientist	19114.12	1824.843333

5 rows × 25 columns

Reset the type of Loan

In [ ]:

```
## Rebuild Type of Loans Columns
for i in train['Type_of_Loan'].value_counts().head(9).index[1:] :
    train[i] = train['Type_of_Loan'].str.contains(i)

#drop the 'Type_of_Loan' column
del train['Type_of_Loan']
```

In [ ]:

```
#recheck the dataset
train.head()
```

Out[ ]:

	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary
0	CUS_0xd40	January	23.0	Scientist	19114.12	1824.843333
1	CUS_0xd40	February	23.0	Scientist	19114.12	NaN
2	CUS_0xd40	March	-500.0	Scientist	19114.12	NaN
3	CUS_0xd40	April	23.0	Scientist	19114.12	NaN
4	CUS_0xd40	May	23.0	Scientist	19114.12	1824.843333

5 rows × 32 columns

Num Bank Accounts

In [ ]:

```
#apply a lambda function to the Num Bank Accounts column to change the negative
train['Num_Bank_Accounts'] = train['Num_Bank_Accounts'].apply(lambda x :abs (x))
```

resizing the type of credit card column

In [ ]:

```
train['Num_Credit_Card'].replace(0,1,inplace=True)
```

convert credit_history_age to months age

In [ ]:
```
#create a function to convert the age column to months
def History_age(age):
    try :
        years = int("".join(re.findall('[0-9]',''.join(age.split("and")[0]))))
        month = int("".join(re.findall('[0-9]',''.join(age.split("and")[1]))))
        return years*12 + month
    except :
        return np.nan
#apply the function to the column
train['Credit_History_Age'] = train['Credit_History_Age'].apply(History_age)
```

Binary categorizing of Payment_Min_Amount column

In [ ]:
```
train['Payment_of_Min_Amount'].replace("NM","No",inplace=True)
```

In [ ]:
```
train['Payment_of_Min_Amount'].value_counts()
```

Out[ ]:
```
Payment_of_Min_Amount
Yes    52326
No     47674
Name: count, dtype: int64
```

Removing the error in the Payment_Behaviour column

In [ ]:
```
train['Payment_Behaviour']= train['Payment_Behaviour'].replace("!@9#%8",np.nan)
train['Payment_Behaviour'].value_counts()
```

Out[ ]:
```
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
Name: count, dtype: int64
```

Imputation of the Occupation Column

In [ ]:
```
train['Occupation'].value_counts()
```

Out[ ]:
```
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 [ ]:
```
occs = train['Occupation'].value_counts().index[1:]
occs
```

Out[ ]:
```
Index(['Lawyer', 'Architect', 'Engineer', 'Scientist', 'Mechanic',
       'Accountant', 'Developer', 'Media_Manager', 'Teacher', 'Entrepreneur',
       'Doctor', 'Journalist', 'Manager', 'Musician', 'Writer'],
      dtype='object', name='Occupation')
```

In [ ]:
```
#further cleaning of the Occupation column
id_ = "CUS_0xb891"
oc = train[train['Customer_ID'] == id_]['Occupation'].mode()[0]
train[train['Customer_ID'] == id_].replace("_______",oc)
```

Out[ ]:

	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary
24	CUS_0xb891	January	54.0	Entrepreneur	30689.89	2612.490833
25	CUS_0xb891	February	54.0	Entrepreneur	30689.89	2612.490833
26	CUS_0xb891	March	55.0	Entrepreneur	30689.89	2612.490833
27	CUS_0xb891	April	55.0	Entrepreneur	30689.89	2612.490833
28	CUS_0xb891	May	55.0	Entrepreneur	30689.89	2612.490833
29	CUS_0xb891	June	55.0	Entrepreneur	30689.89	2612.490833
30	CUS_0xb891	July	55.0	Entrepreneur	30689.89	2612.490833
31	CUS_0xb891	August	55.0	Entrepreneur	30689.89	2612.490833

8 rows × 32 columns

In [ ]:
```
#for loop to replace some specific redundant values in the Occupation column
for ID in train[train['Occupation'] == "_______"]['Customer_ID'] :
    oc = train[train['Customer_ID'] == ID]['Occupation'].mode()[0]
    train[train['Customer_ID'] == ID] = train[train['Customer_ID'] == ID].replac
```

In [ ]:
```
train['Occupation'].value_counts()
```

Out[ ]:
```
Occupation
Lawyer           7096
Engineer         6864
Architect        6824
Mechanic         6776
Scientist        6744
Accountant       6744
Developer        6720
Media_Manager    6715
Teacher          6672
Entrepreneur     6648
Doctor           6568
Journalist       6536
Manager          6432
Musician         6352
Writer           6304
_______             5
Name: count, dtype: int64
```

In [ ]:
```
#replace the '_______' values in the Occupation column with the mode of the colu
train['Occupation'] = train['Occupation'].replace("_______",train['Occupation'].
train['Occupation'].value_counts()
```

Out[ ]:
```
Occupation
Lawyer           7101
Engineer         6864
Architect        6824
Mechanic         6776
Scientist        6744
Accountant       6744
Developer        6720
Media_Manager    6715
Teacher          6672
Entrepreneur     6648
Doctor           6568
Journalist       6536
Manager          6432
Musician         6352
Writer           6304
Name: count, dtype: int64
```

recategorize the credit mix column

In [ ]:
```
train['Credit_Mix'].value_counts()
```

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

In [ ]:
```
#apply custom catgorical encoding to the 'Credit_Mix' column
m = {
    "Bad":0,
    "Standard":1,
    "Good":2,
```

```
    "_":np.nan
}
train['Credit_Mix'] = train['Credit_Mix'].apply(lambda x : m[x])
```

handle missing values

In [ ]:
```
# Edit Columns from bool to int
for col in list(train.columns[-8:]):
    train[col] = train[col].astype(float)
```

In [ ]:
```
#replacing unique ID with a simple integer that increments for each unique ID
IDs = 1
for ID in train['Customer_ID'].unique() :
    train['Customer_ID'] = train['Customer_ID'].replace(ID,IDs)
    IDs += 1
```

In [ ]:
```
#apply KNN imputer to the dataset to fill the missing values
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=1)
```

In [ ]:
```
Numericals = train.select_dtypes(exclude='object').columns[1:]
Numericals
```

Out[ ]:
```
Index(['Age', 'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_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', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Monthly_Balance', 'Credit-Builder Loan',
       'Personal Loan', 'Debt Consolidation Loan', 'Student Loan',
       'Payday Loan', 'Mortgage Loan', 'Auto Loan', 'Home Equity Loan'],
      dtype='object')
```

In [ ]:
```
#for loop to apply the imputer to all the numerical columns in the dataset:
for col in Numericals[1:]:
    imputer.fit(train[['Customer_ID',col]])
    train[['Customer_ID',col]] = imputer.transform(train[['Customer_ID',col]])
```

In [ ]:
```
#recheck the dataset
train
```

Out[ ]:

	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Sala
0	1.0	January	23.0	Scientist	19114.12	1824.8433
1	1.0	February	23.0	Scientist	19114.12	1824.8433
2	1.0	March	-500.0	Scientist	19114.12	1824.8433
3	1.0	April	23.0	Scientist	19114.12	1824.8433
4	1.0	May	23.0	Scientist	19114.12	1824.8433
...	...	...	...	...	...	
99995	12500.0	April	25.0	Mechanic	39628.99	3359.4158
99996	12500.0	May	25.0	Mechanic	39628.99	3359.4158
99997	12500.0	June	25.0	Mechanic	39628.99	3359.4158
99998	12500.0	July	25.0	Mechanic	39628.99	3359.4158
99999	12500.0	August	25.0	Mechanic	39628.99	3359.4158

100000 rows × 32 columns

In [ ]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 32 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Customer_ID               100000 non-null  float64
 1   Month                     100000 non-null  object 
 2   Age                       100000 non-null  float64
 3   Occupation                100000 non-null  object 
 4   Annual_Income             100000 non-null  float64
 5   Monthly_Inhand_Salary     100000 non-null  float64
 6   Num_Bank_Accounts         100000 non-null  float64
 7   Num_Credit_Card           100000 non-null  float64
 8   Interest_Rate             100000 non-null  float64
 9   Num_of_Loan               100000 non-null  float64
 10  Delay_from_due_date       100000 non-null  float64
 11  Num_of_Delayed_Payment    100000 non-null  float64
 12  Changed_Credit_Limit      100000 non-null  float64
 13  Num_Credit_Inquiries      100000 non-null  float64
 14  Credit_Mix                100000 non-null  float64
 15  Outstanding_Debt          100000 non-null  float64
 16  Credit_Utilization_Ratio  100000 non-null  float64
 17  Credit_History_Age        100000 non-null  float64
 18  Payment_of_Min_Amount     100000 non-null  object 
 19  Total_EMI_per_month       100000 non-null  float64
 20  Amount_invested_monthly   100000 non-null  float64
 21  Payment_Behaviour         92400 non-null   object 
 22  Monthly_Balance           100000 non-null  float64
 23  Credit_Score              100000 non-null  object 
 24  Credit-Builder Loan       100000 non-null  float64
 25  Personal Loan             100000 non-null  float64
 26  Debt Consolidation Loan   100000 non-null  float64
 27  Student Loan              100000 non-null  float64
 28  Payday Loan               100000 non-null  float64
 29  Mortgage Loan             100000 non-null  float64
 30  Auto Loan                 100000 non-null  float64
 31  Home Equity Loan          100000 non-null  float64
dtypes: float64(27), object(5)
memory usage: 24.4+ MB
```

The column Payment_behaviour is yet to be filled, so we can fit impute function on it

In [ ]:

```
#refit the imputer to the Payment Behaviour column specifically to fill its miss
imputer = SimpleImputer(strategy="most_frequent")
imputer.fit(train[['Payment_Behaviour']])
train[['Payment_Behaviour']] = imputer.transform(train[['Payment_Behaviour']])
```

In [ ]:

```
train.isnull().sum()
```

Out[ ]:
```
Customer_ID                 0
Month                       0
Age                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_Score                0
Credit-Builder Loan         0
Personal Loan               0
Debt Consolidation Loan     0
Student Loan                0
Payday Loan                 0
Mortgage Loan               0
Auto Loan                   0
Home Equity Loan            0
dtype: int64
```

Outlier Detection and Handling

In [ ]:
```python
## replace Outliers with median
for col in Numericals :
    outliers_indecies = detect_outliers(train,0,[col])
    median = train[col].median()
    train[col].iloc[outliers_indecies] = median
```

Data Preprocessing

Handling Categorical Columns

In [ ]:
```python
#check for categorical columns and there values
train.select_dtypes(include="object")
```

Out[ ]:

	Month	Occupation	Payment_of_Min_Amount	Payment_Behavioı
0	January	Scientist	No	High_spent_Small_value_paymen
1	February	Scientist	No	Low_spent_Large_value_paymen
2	March	Scientist	No	Low_spent_Medium_value_paymen
3	April	Scientist	No	Low_spent_Small_value_paymen
4	May	Scientist	No	High_spent_Medium_value_paymen
...	...	...	...	
99995	April	Mechanic	No	High_spent_Large_value_paymen
99996	May	Mechanic	No	High_spent_Medium_value_paymen
99997	June	Mechanic	No	High_spent_Large_value_paymen
99998	July	Mechanic	No	Low_spent_Large_value_paymen
99999	August	Mechanic	No	Low_spent_Small_value_paymen

100000 rows × 5 columns

In [ ]:
```
#check for the count of the target columns
train['Credit_Score'].value_counts()
```

Out[ ]:
```
Credit_Score
Standard    53174
Poor        28998
Good        17828
Name: count, dtype: int64
```

In [ ]:
```
#create a dictionary to map the target column
m = {
    "Poor":0,
    "Standard":1,
    "Good":2
}
```

In [ ]:
```
#apply the dictionary to the target column
train['Credit_Score'] = train['Credit_Score'].map(m)
```

In [ ]:
```
#drop the 'Customer_ID' column
del train['Customer_ID']
```

In [ ]:
```
train = pd.get_dummies(train,drop_first=True)
```

In [ ]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 54 columns):
 #   Column                                          Non-Null Count   Dtype  
---  ------                                          --------------   -----  
 0   Age                                             100000 non-null  float64
 1   Annual_Income                                   100000 non-null  float64
 2   Monthly_Inhand_Salary                           100000 non-null  float64
 3   Num_Bank_Accounts                               100000 non-null  float64
 4   Num_Credit_Card                                 100000 non-null  float64
 5   Interest_Rate                                   100000 non-null  float64
 6   Num_of_Loan                                     100000 non-null  float64
 7   Delay_from_due_date                             100000 non-null  float64
 8   Num_of_Delayed_Payment                          100000 non-null  float64
 9   Changed_Credit_Limit                            100000 non-null  float64
 10  Num_Credit_Inquiries                            100000 non-null  float64
 11  Credit_Mix                                      100000 non-null  float64
 12  Outstanding_Debt                                100000 non-null  float64
 13  Credit_Utilization_Ratio                        100000 non-null  float64
 14  Credit_History_Age                              100000 non-null  float64
 15  Total_EMI_per_month                             100000 non-null  float64
 16  Amount_invested_monthly                         100000 non-null  float64
 17  Monthly_Balance                                 100000 non-null  float64
 18  Credit_Score                                    100000 non-null  int64  
 19  Credit-Builder Loan                             100000 non-null  float64
 20  Personal Loan                                   100000 non-null  float64
 21  Debt Consolidation Loan                         100000 non-null  float64
 22  Student Loan                                    100000 non-null  float64
 23  Payday Loan                                     100000 non-null  float64
 24  Mortgage Loan                                   100000 non-null  float64
 25  Auto Loan                                       100000 non-null  float64
 26  Home Equity Loan                                100000 non-null  float64
 27  Month_August                                    100000 non-null  bool   
 28  Month_February                                  100000 non-null  bool   
 29  Month_January                                   100000 non-null  bool   
 30  Month_July                                      100000 non-null  bool   
 31  Month_June                                      100000 non-null  bool   
 32  Month_March                                     100000 non-null  bool   
 33  Month_May                                       100000 non-null  bool   
 34  Occupation_Architect                            100000 non-null  bool   
 35  Occupation_Developer                            100000 non-null  bool   
 36  Occupation_Doctor                               100000 non-null  bool   
 37  Occupation_Engineer                             100000 non-null  bool   
 38  Occupation_Entrepreneur                         100000 non-null  bool   
 39  Occupation_Journalist                           100000 non-null  bool   
 40  Occupation_Lawyer                               100000 non-null  bool   
 41  Occupation_Manager                              100000 non-null  bool   
 42  Occupation_Mechanic                             100000 non-null  bool   
 43  Occupation_Media_Manager                        100000 non-null  bool   
 44  Occupation_Musician                             100000 non-null  bool   
 45  Occupation_Scientist                            100000 non-null  bool   
 46  Occupation_Teacher                              100000 non-null  bool   
 47  Occupation_Writer                               100000 non-null  bool   
 48  Payment_of_Min_Amount_Yes                       100000 non-null  bool   
 49  Payment_Behaviour_High_spent_Medium_value_payments  100000 non-null  bool   
 50  Payment_Behaviour_High_spent_Small_value_payments   100000 non-null  bool   
 51  Payment_Behaviour_Low_spent_Large_value_payments    100000 non-null  bool   
 52  Payment_Behaviour_Low_spent_Medium_value_payments   100000 non-null  bool   
 53  Payment_Behaviour_Low_spent_Small_value_payments    100000 non-null  bool   
```

```
dtypes: bool(27), float64(26), int64(1)
memory usage: 23.2 MB
```

All required preprocessing done, safe to save the preprocessed data now

In [ ]:
```
#save the cleaned train data
train.to_csv('train_cleaned.csv', index=False)
```

In [ ]:
```
#load the cleaned train data
train = pd.read_csv('train_cleaned.csv', low_memory=False)
train.head()
```

Out[ ]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card
0	23.0	19114.12	1824.843333	3.0	4.0
1	23.0	19114.12	1824.843333	3.0	4.0
2	33.0	19114.12	1824.843333	3.0	4.0
3	23.0	19114.12	1824.843333	3.0	4.0
4	23.0	19114.12	1824.843333	3.0	4.0

5 rows × 54 columns

Data Splitting

In [ ]:
```
# define dataset
X, y = train.drop("Credit_Score",axis=1).values , train["Credit_Score"]
```

Since we observed that our target variable distribution is imbalanced, Hence Oversampling or Resampling techniques will be carried out to improve it.

In [ ]:
```
#check the normalized count of the target variable
y.value_counts(normalize=True)
```

Out[ ]:
```
Credit_Score
1    0.53174
0    0.28998
2    0.17828
Name: proportion, dtype: float64
```

In [ ]:
```
#apply SMOTE to the dataset to balance the target column
from imblearn.over_sampling import SMOTE
rus = SMOTE(sampling_strategy='auto')
X_data_rus, y_data_rus = rus.fit_resample(X, y)
```

In [ ]:
```
y_data_rus.value_counts(normalize=True)
```

Out[ ]:
```
Credit_Score
2    0.333333
1    0.333333
0    0.333333
Name: proportion, dtype: float64
```

In [ ]:
```
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_data_rus, y_data_rus, test
```

Handling Numerical Variables

PowerTransformer is employed here to avoid data skewness

In [ ]:
```
#call the PowerTransformer function to transform the dataset
scalar = PowerTransformer(method='yeo-johnson', standardize=True).fit(X_train)
```

In [ ]:
```
#transform the train and test data
X_train = scalar.transform(X_train)
X_test = scalar.transform(X_test)
```

Modeling and Evaluation

Model Building

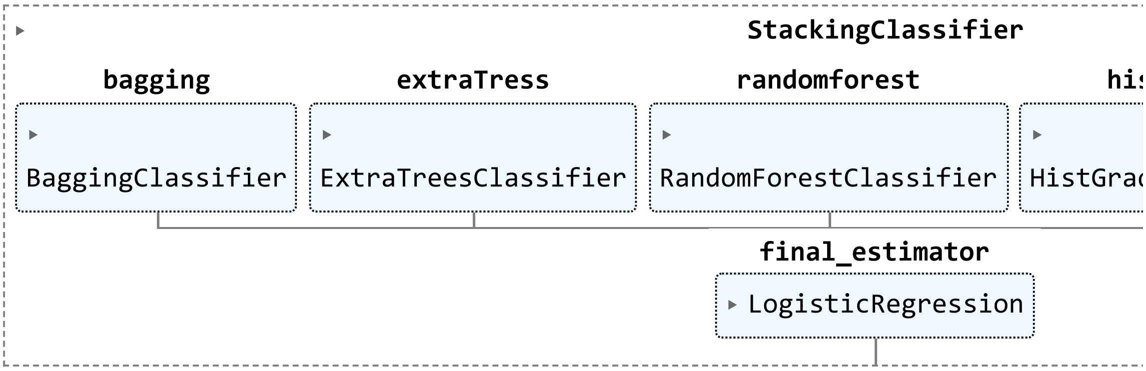
In [ ]:
```
bagging = BaggingClassifier(n_jobs=-1)
extraTrees = ExtraTreesClassifier(max_depth=10, n_jobs=-1)
randomForest = RandomForestClassifier(n_jobs=-1)
histGradientBoosting = HistGradientBoostingClassifier()
XGB = XGBClassifier(n_jobs=-1)

model = StackingClassifier([
    ('bagging', bagging),
    ('extraTress', extraTrees),
    ('randomforest', randomForest),
    ('histGradientBoosting', histGradientBoosting),
    ('XGB', XGB)
], n_jobs=-1)
```

Model Fitting/Training

In [ ]:
```
model.fit(X_train, y_train)
```

Out[ ]:



Model Evaluation

In [ ]:
```
print("Train Score: ",model.score(X_train, y_train))
print("Test Score: ",model.score(X_test, y_test))
```

```
Train Score:  0.9993731249720145
Test Score:  0.8493846250287315
```

In [ ]:
```
y_pred = model.predict(X_test)
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.86     15473
           1       0.81      0.79      0.80     16321
           2       0.89      0.89      0.89     16063

    accuracy                           0.85     47857
   macro avg       0.85      0.85      0.85     47857
weighted avg       0.85      0.85      0.85     47857
```

Saving the model

In [ ]:
```
#save the model
joblib.dump(model, 'model.pkl')
```

Out[ ]:
```
['model.pkl']
```