greatlearning

DSE FT B Jun 23 G2 Capstone Final Report

Batch details	PGPDSE-FT BANGALORE JUN23
Team members	Shobhraj Mistry Sneha Prasad Suhas Akshay Vardhan R J Vikram Venkat
Domain of Project	Sales Analytics
Proposed project title	Multi-Class Classification Problem
Group Number	Group - 2
Team Leader	Sneha Prasad
Mentor Name	Chandran Venkatesan

Date: 18th January, 2024

Problem Statement

In the dynamic landscape of company named global finance company, a substantial repository of basic bank details and extensive credit-related information has been amassed over the years. Recognizing the need for efficiency and accuracy in evaluating creditworthiness, the management has articulated the necessity for an intelligent system. The primary objective is to develop a robust solution that can autonomously categorize individuals into distinct credit score brackets. This initiative is aimed at mitigating the reliance on manual efforts, streamlining the credit assessment process, and enhancing the overall efficiency of credit-related decision-making within the organization. The challenge lies in designing and implementing an intelligent system that leverages the available data to provide accurate and reliable credit score assignments, contributing to a more streamlined and data-driven approach in our financial operations.

Data & Findings

The dataset at our disposal is a comprehensive repository of basic bank details and credit-related information. It includes a multitude of variables such as:

1. **Personal Information such as** Name, Age, Gender and Address
2. **Financial Details such as** Income, Employment status, Debt-to-income ratio and Current financial obligations
3. **Credit History such as** Previous loans and their status, Credit card usage and payment history and Outstanding debts
4. **Risk Factors such as** External economic indicators affecting credit risk, Industry-specific risk factors
5. **Behavioural Patterns such as** Spending habits, Savings patterns and Transaction history

The dataset is diverse, covering a wide range of demographics and financial behaviours. The challenge lies in extracting meaningful patterns and relationships from this data to build an intelligent credit scoring system.

Preliminary analysis of the dataset reveals the complexity and variability in credit-related factors. Some initial observations include:

1. **High Dimensionality:** The dataset exhibits high dimensionality with numerous features influencing creditworthiness, necessitating careful feature selection and extraction.
2. **Data Discrepancies:** Inconsistencies and missing values are prevalent, requiring data preprocessing techniques to ensure the robustness of the intelligent system.
3. **Correlation Patterns:** Certain variables exhibit strong correlations with credit scores, offering valuable insights into potential key determinants.
4. **Behavioural Insights:** Behavioural patterns, such as spending habits and transaction history, emerge as potential indicators of credit risk.

These initial findings underscore the importance of a thorough exploration and analysis of the dataset to inform the development of an intelligent credit scoring system that accurately reflects the creditworthiness of individuals while minimizing manual efforts.

Industry Review

Current Practices:

1. Machine Learning and Predictive Analytics:

The use of machine learning algorithms for credit rating is growing. To increase risk assessment, predictive analytics models such as gradient boosting, random forests, and neural networks are being used.

2. **Dynamic Credit Scoring Models:**

The development of dynamic credit scoring algorithms that can react to changes in an individual's financial condition in real-time is gaining traction. This is especially important when analyzing credit risk in a fast-changing economic context.

3. Real-time Data and IoT Integration:

Some financial organizations are investigating the incorporation of real-time data and Internet of Things (IoT) data into credit rating algorithms. This enables for a more complete and up-to-date assessment of an individual's financial behavior.

Background Research:

1. Incorporation of Alternative Data:

Financial organizations are experimenting with data sources other than typical credit bureau data. Utility payments, rental history, and even social media behavior are examples of such issues.

2. Fairness and Bias Mitigation:

To promote fair lending practices, researchers and practitioners are working hard to overcome flaws in credit scoring algorithms. This involves looking into ways to decrease the differential impact on different demographic groups.

3. Block chain Technology in Credit Scoring:

Exploratory study is being performed to investigate the possible application of blockchain technology in credit assessment. Blockchain can improve the security, transparency, and accuracy of credit-related data.

Literature Survey

Publications:

Title: " Multi-Class Classification Problem"

- Authors: Sudhanshu Rastogi
- Published in: https://www.kaggle.com/datasets/sudhanshu2198/processed-data-credit-score
- Year: 2023
- Summary: Pre-process the unclean and messy data from above source.

Title: "Credit Scoring and Its Applications"

- Authors: Thomas, L. C., Edelman, D. B., & Crook, J. N.
- Published in: Journal of the Operational Research Society
- Year: 2002
- Summary: This publication provides a comprehensive overview of credit scoring, discussing various modeling techniques and applications.

Title: "A Comparative Analysis of Credit Risk Models"

- Authors: Altman, E. I., Marco, G. N., & Varetto, F.
- Published in: Journal of Banking & Finance
- Year: 1994
- Summary: The paper compares different credit scoring models, including traditional statistical methods and emerging machine learning approaches.

Past and Ongoing Research:

1. Fairness and Bias Mitigation:

On going research focuses on addressing biases in credit scoring models to ensure fair lending practices. Researchers are developing techniques to reduce disparate impact on different demographic groups.

2. Behavioral Economics in Credit Scoring:

Past and current research examines the application of behavioral economics principles to credit scoring. Understanding the psychological factors influencing credit behavior contributes to more accurate models.

3. Blockchain Technology in Credit Scoring:

Description: Exploratory research is being conducted to assess the potential use of blockchain technology in credit scoring. Blockchain can enhance data security, transparency, and the accuracy of credit-related information.

4. Continuous Model Monitoring and Updates:

Description: Current research emphasizes the importance of continuous monitoring and regular updates of credit scoring models. This ensures that models remain accurate and relevant in a dynamic financial landscape.

Overview:

Dataset and Domain

1. **Data Dictionary:**
   - Age: Represents the age of the person
   - Annual_Income: Represents the annual income of the person
   - Monthly_Inhand_Salary: Represents the monthly base salary of a person
   - Num_Bank_Accounts:Represents the number of bank accounts a person holds
   - Num_Credit_Card: Represents the number of other credit cards held by a person
   - Interest_Rate: Represents the interest rate on credit card
   - Num_of_Loan: Represents the number of loans taken from the bank
   - Delay_from_due_date: Represents the average number of days delayed from the payment date
   - Num_of_Delayed_Payment: Represents the average number of payments delayed by a person
   - Changed_Credit_Limit: Represents the percentage change in credit card limit
   - Num_Credit_Inquiries: Represents the number of credit card inquiries
   - Credit_Mix: Represents the classification of the mix of credits
   - Outstanding_Debt: Represents the remaining debt to be paid (in USD)
   - Credit_Utilization_Ratio: Represents the utilization ratio of credit card
   - Credit_History_Age: Represents the age of credit history of the person
   - Payment_of_Min_Amount: Represents whether only the minimum amount was paid by the person
   - Total_EMI_per_month: Represents the monthly EMI payments (in USD)
   - Amount_invested_monthly: Represents the monthly amount invested by the customer (in USD)
   - Monthly_Balance: Represents the monthly balance amount of the customer (in USD)
   - Credit_Score: Represents the bracket of credit score (Poor, Standard, Good)

2. **Variable categorization (count of numeric and categorical):**

In the given dataset, there are 17 numerical features and 5 categorical features.

```
Numeric_cols = df.select_dtypes(include=np.number).columns

Categoric_cols = df.select_dtypes(exclude=np.number).columns[:-1]

Numeric_cols,Categoric_cols
```

```
(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', 'Outstanding_Debt', 'Credit_Utilization_Ratio',
        'Credit_History_Age', 'Total_EMI_per_month', 'Amount_invested_monthly',
        'Monthly_Balance'],
       dtype='object'),
 Index(['Month', 'Occupation', 'Credit_Mix', 'Payment_of_Min_Amount',
        'Payment_Behaviour'],
       dtype='object'))
```

3. **Reading the data:** We begin by first reading the data into the system and then start exploring the count of actual data available with us. Steps are mentioned below for reference.

Reading Data

```
In [163]: df = pd.read_csv("train.csv")
```

```
In [164]: df.head()
```

Out[164]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthl
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	

4. **Data exploration:** There are 100000 rows of data with total 28 columns, which means there are total of 28 variables or parameters and the data is of 100000 customers.

```
In [165]: df.shape
Out[165]: (100000, 28)
```

```
In [166]: df.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 [167]: df.describe().T
```

Out[167]:

	count	mean	std	min	25%	
Monthly_Inhand_Salary	84998.000000	4194.170850	3183.686167	303.645417	1625.568229	3
Num_Bank_Accounts	100000.000000	17.091280	117.404834	-1.000000	3.000000	
Num_Credit_Card	100000.000000	22.474430	129.057410	0.000000	4.000000	
Interest_Rate	100000.000000	72.466040	466.422621	1.000000	8.000000	
Delay_from_due_date	100000.000000	21.068780	14.860104	-5.000000	10.000000	
Num_Credit_Inquiries	98035.000000	27.754251	193.177339	0.000000	3.000000	
Credit_Utilization_Ratio	100000.000000	32.285173	5.116875	20.000000	28.052567	
Total_EMI_per_month	100000.000000	1403.118217	8306.041270	0.000000	30.306660	

```
In [168]: df.select_dtypes(exclude=np.number).describe().T
```

Out[168]:

	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	4384	_	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	98792	__-333333333333333333333333333__	9
Credit_Score	100000	3	Standard	53174

```
In [169]: df.duplicated().sum()
```

Out[169]: 0

As it can be observed from above data, that we have tried to identify if the data is object type or float type, what are the total number of data available for each header, mean, mode, std and top 25% data.

```
In [170]: Null_Percent = (df.isnull().sum()/df.shape[0]*100).to_frame().sort_values(b
          Null_Percent.rename(columns={0: 'Percentage'},inplace=True)
          Null_Percent[Null_Percent.Percentage>0]
```

Out[170]:

	Percentage
Monthly_Inhand_Salary	15.002000
Type_of_Loan	11.408000
Name	9.985000
Credit_History_Age	9.030000
Num_of_Delayed_Payment	7.002000
Amount_invested_monthly	4.479000
Num_Credit_Inquiries	1.965000
Monthly_Balance	1.200000

Step by Step walk through the solution:

Post understanding the details of the input variables as per the dataset, we will now clean the data and use the data explorations technique to analyze in-depth about the patterns of data so that we can bring out the accurate solutions.

1. **Data cleaning:** This essential step ensures that the data is accurate, reliable, and well-structured, laying a solid foundation for meaningful analysis and decision-making. Common data cleaning tasks in Python involve handling missing values, removing duplicates, standardizing formats, and addressing outliers, ultimately contributing to improved data quality and the overall effectiveness of data-driven processes. Python provides a versatile set of libraries, such as Pandas and NumPy, making it a popular choice for implementing efficient and scalable data cleaning workflows.

Removing Un-useful Columns

```
In [172]: del df['ID'] # Identification
          del df['Name'] # Name of client
          del df['SSN'] # SSN (social security number of a person)
```

Fixing data error in Columns

```
In [173]: for col in df.iloc[:, 1:]:
              print("==================================================")
              print(f"Unique Values of {col}")
              print("==================================================")
              print(df[col].value_counts())
```

```
==================================================
Unique Values of Month
==================================================
January     12500
February    12500
March       12500
April       12500
May         12500
June        12500
July        12500
August      12500
Name: Month, dtype: int64
==================================================
Unique Values of Age
==================================================
38       2833
28       2829
31       2806
26       2792
```

Fix underscore & other data error

In [174]:
```
df = df.applymap(lambda x: x.replace('_','') if isinstance(x, str) else x)
df.replace(['', 'nan', '!@9#%8', '#F%$D@*&8'], np.nan, inplace=True)
```

Fix Data type

In [175]:
```
df['Age']                    = df.Age.astype(int)
df['Annual_Income']          = df.Annual_Income.astype(float)
df['Num_of_Loan']            = df.Num_of_Loan.astype(int)
df['Num_of_Delayed_Payment'] = df.Num_of_Delayed_Payment.astype(float)
df['Changed_Credit_Limit']   = df.Changed_Credit_Limit.astype(float)
df['Outstanding_Debt']       = df.Outstanding_Debt.astype(float)
df['Amount_invested_monthly'] = df.Amount_invested_monthly.astype(float)
df['Monthly_Balance']        = df.Monthly_Balance.astype(float)
```

In [176]:
```
df.columns
```

Out[176]:
```
Index(['Customer_ID', 'Month', 'Age', 'Occupation', 'Annual_Income',
       'Monthly_Inhand_Salary', 'Num_Bank_Accounts', 'Num_Credit_Card',
       'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan', 'Delay_from_due_date',
       'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score'],
      dtype='object')
```

Month

In [177]:
```
df['Month'].value_counts(dropna=False)
```

Out[177]:
```
January     12500
February    12500
March       12500
April       12500
May         12500
June        12500
July        12500
August      12500
Name: Month, dtype: int64
```

Age

In [178]:
```
df['Age'].value_counts(dropna=False)
```

Out[178]:
```
38    2994
28    2968
31    2955
26    2945
32    2884
36    2868
35    2866
25    2861
27    2859
39    2846
34    2837
44    2824
19    2793
22    2785
41    2785
20    2744
37    2742
29    2735
43    2734
```

Occupation

In [180]:
```
df['Occupation'].value_counts(dropna=False)
```

Out[180]:
```
NaN             7062
Lawyer          6575
Architect       6355
Engineer        6350
Scientist       6299
Mechanic        6291
Accountant      6271
Developer       6235
MediaManager    6232
Teacher         6215
Entrepreneur    6174
Doctor          6087
Journalist      6085
Manager         5973
Musician        5911
Writer          5885
Name: Occupation, dtype: int64
```

Num_Bank_Accounts

In [183]:
```
df['Num_Bank_Accounts'].value_counts(dropna=False)
```

Out[183]:
```
 6       13001
 7       12823
 8       12765
 4       12186
 5       12118
 3       11950
 9        5443
 10       5247
 1        4490
 0        4328
 2        4304
-1          21
 11          9
 803         7
 1668        5
 791         5
 105         5
 210         4
 1139        4
```

In [184]:
```
df['Num_Bank_Accounts'] = np.where((df['Num_Bank_Accounts'] > 10) | (df['Num_Bank_Accounts'] < 0), np.nan, df['Num_Bank_Accounts'
df['Num_Bank_Accounts'].value_counts(dropna=False)
```

Out[184]:
```
6.000000     13001
7.000000     12823
8.000000     12765
4.000000     12186
5.000000     12118
3.000000     11950
9.000000      5443
10.000000     5247
1.000000      4490
0.000000      4328
2.000000      4304
NaN           1345
Name: Num_Bank_Accounts, dtype: int64
```

Num_Credit_Card

In [185]:
```
df['Num_Credit_Card'].value_counts(dropna=False)
```

Out[185]:
```
5       18459
7       16615
6       16559
4       14030
3       13277
8        4956
10       4860
9        4643
2        2149
1        2132
11         36
0          13
849         8
852         7
183         6
106         6
1420        6
958         6
159         6
```

In [186]:
```
df['Num_Credit_Card'] = np.where((df['Num_Credit_Card'] > 10) | (df['Num_Credit_Card'] < 0), np.nan, df['Num_Credit_Card'])
df['Num_Credit_Card'].value_counts(dropna=False)
```

Out[186]:
```
5.000000     18459
7.000000     16615
6.000000     16559
4.000000     14030
3.000000     13277
8.000000      4956
10.000000     4860
9.000000      4643
NaN           2307
2.000000      2149
1.000000      2132
0.000000        13
Name: Num_Credit_Card, dtype: int64
```

Credit_Mix

In [200]:
```
df['Credit_Mix'].value_counts(dropna=False)
```

Out[200]:
```
Standard    36479
Good        24337
NaN         20195
Bad         18989
Name: Credit_Mix, dtype: int64
```

Credit_Utilization_Ratio

In [202]:
```
df['Credit_Utilization_Ratio'].value_counts(dropna=False)
```

Out[202]:
```
26.822620    1
28.327949    1
30.016576    1
25.478841    1
33.933755    1
            ..
30.687138    1
38.730069    1
30.017515    1
27.279794    1
34.192463    1
Name: Credit_Utilization_Ratio, Length: 100000, dtype: int64
```

Credit_History_Age

In [203]:
```
df['Credit_History_Age'].value_counts(dropna=False)
```

Out[203]:
```
NaN                        9030
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
```

In [204]:
```
def month_conv(x):
    if type(x)==float:
        return x
    else:
        a=x.split()
        b=int(a[0])*12+int(a[3])
        return b
```

In [205]:
```
df['Credit_History_Age'] = df['Credit_History_Age'].apply(month_conv)
df['Credit_History_Age'].value_counts(dropna=False)
```

Out[205]:
```
NaN           9030
191.000000     446
232.000000     445
233.000000     444
215.000000     443
231.000000     441
213.000000     438
190.000000     436
214.000000     435
189.000000     432
219.000000     428
220.000000     426
218.000000     426
237.000000     422
212.000000     419
188.000000     415
227.000000     414
194.000000     412
221.000000     410
```

Payment_of_Min_Amount

In [206]:
```
df['Payment_of_Min_Amount'].value_counts(dropna=False)
```

Out[206]:
```
Yes    52326
No     35667
NM     12007
Name: Payment_of_Min_Amount, dtype: int64
```

Monthly_Balance

In [210]:
```
df['Monthly_Balance'].value_counts(dropna=False)
```

Out[210]:
```
NaN                                   1200
-333333333333333314856026112.000000      9
312.494089                               1
347.413890                               1
254.970922                               1
                                      ...
366.289038                               1
151.188270                               1
306.750279                               1
278.872026                               1
393.673696                               1
Name: Monthly_Balance, Length: 98793, dtype: int64
```

In [211]:
```
df['Monthly_Balance'] = np.where(df['Monthly_Balance'] < 0, np.nan, df['Monthly_Balance'])
df['Monthly_Balance'].value_counts(dropna=False)
```

Out[211]:
```
NaN           1209
312.494089       1
589.699342       1
250.093168       1
289.755075       1
              ...
366.289038       1
151.188270       1
306.750279       1
278.872026       1
393.673696       1
Name: Monthly_Balance, Length: 98792, dtype: int64
```

Credit_Score

In [212]:
```
df['Credit_Score'].value_counts(dropna=False)
```

Out[212]:
```
Standard    53174
Poor        28998
Good        17828
Name: Credit_Score, dtype: int64
```

Null values before imputing

In [213]:
```
Null_Percent = (df.isnull().sum()/df.shape[0]*100).to_frame().sort_values(by=0,ascending=False)
Null_Percent.rename(columns={0: 'Percentage'},inplace=True)
Null_Percent[Null_Percent.Percentage>0]
```

Out[213]:

	Percentage
Credit_Mix	20.195000
Monthly_Inhand_Salary	15.002000
Num_of_Loan	11.408000
Type_of_Loan	11.408000
Credit_History_Age	9.030000
Num_of_Delayed_Payment	7.726000
Payment_Behaviour	7.600000
Occupation	7.062000
Amount_invested_monthly	4.479000
Num_Credit_Inquiries	3.599000
Age	2.776000
Num_Credit_Card	2.307000
Changed_Credit_Limit	2.091000
Interest_Rate	2.034000
Num_Bank_Accounts	1.345000
Monthly_Balance	1.209000

2. **Data Preprocessing:** We are using KNN method for data processing as K-Nearest Neighbors (KNN) imputation is a valuable technique in the realm of data pre-processing, which addresses the challenge of missing values within a dataset. This method involves filling in missing data points by considering the values of their nearest neighbors. KNN imputation stands out for its ability to preserve local patterns, making it particularly suitable for datasets with spatial or temporal dependencies. Its non-parametric nature ensures adaptability to various data distributions and complex relationships between variables. With its dynamic adaptation to local density and the absence of a dedicated training phase, KNN imputation serves as a flexible and effective solution for enhancing the completeness and quality of datasets during the crucial data pre-processing stage.

Imputing using KNN

```
In [214]: Numeric_cols = df.select_dtypes(include=np.number).columns
          Numeric_cols
```

```
Out[214]: 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', 'Outstanding_Debt', 'Credit_Utilization_Ratio',
                 'Credit_History_Age', 'Total_EMI_per_month', 'Amount_invested_monthly',
                 'Monthly_Balance'],
                dtype='object')
```

```
In [215]: from sklearn.impute import KNNImputer

          imputer = KNNImputer(n_neighbors=5)
          df[Numeric_cols] = imputer.fit_transform(df[Numeric_cols])
```

Null values after imputing

```
In [216]: Null_Percent = (df[Numeric_cols].isnull().sum()/df[Numeric_cols].shape[0]*100).to_frame().sort_values(by=0,ascending=False)
          Null_Percent.rename(columns={0: 'Percentage'},inplace=True)
          Null_Percent
```

Out[216]:

	Percentage
Age	0.000000
Changed_Credit_Limit	0.000000
Amount_invested_monthly	0.000000
Total_EMI_per_month	0.000000
Credit_History_Age	0.000000
Credit_Utilization_Ratio	0.000000
Outstanding_Debt	0.000000
Num_Credit_Inquiries	0.000000
Num_of_Delayed_Payment	0.000000
Annual_Income	0.000000
Delay_from_due_date	0.000000
Num_of_Loan	0.000000
Interest_Rate	0.000000
Num_Credit_Card	0.000000
Num_Bank_Accounts	0.000000
Monthly_Inhand_Salary	0.000000
Monthly_Balance	0.000000

Filling missing values in Categorical columns

In [217]:
```
Categoric_cols = df.iloc[:, 1:].select_dtypes(exclude=np.number).columns
Categoric_cols
```

Out[217]:
```
Index(['Month', 'Occupation', 'Type_of_Loan', 'Credit_Mix',
       'Payment_of_Min_Amount', 'Payment_Behaviour', 'Credit_Score'],
      dtype='object')
```

In [218]:
```
Null_Percent = (df.isnull().sum()/df.shape[0]*100).to_frame().sort_values(by=0,ascending=False)
Null_Percent.rename(columns={0: 'Percentage'},inplace=True)
Null_Percent[Null_Percent.Percentage>0]
```

Out[218]:

	Percentage
Credit_Mix	20.195000
Type_of_Loan	11.408000
Payment_Behaviour	7.600000
Occupation	7.062000

In [219]:
```
df['Credit_Mix']=df.groupby('Customer_ID')['Credit_Mix'].transform(lambda x:x.fillna(x.mode()[0]))
```

In [220]:
```
df['Payment_Behaviour']=df.groupby('Customer_ID')['Payment_Behaviour'].transform(lambda x:x.fillna(x.mode()[0]))
```

In [221]:
```
df['Occupation']=df.groupby('Customer_ID')['Occupation'].transform(lambda x:x.fillna(x.mode()[0]))
```

In [222]:
```
Null_Percent = (df.isnull().sum()/df.shape[0]*100).to_frame().sort_values(by=0,ascending=False)
Null_Percent.rename(columns={0: 'Percentage'},inplace=True)
Null_Percent[Null_Percent.Percentage>0]
```

Out[222]:

	Percentage
Type_of_Loan	11.408000

Removing non usefulll columns

In [223]:
```
del df['Customer_ID'] # Identification
del df['Type_of_Loan'] # Already we have this data in No of Loans column
del df['Month']         # Data Doesnt have much variation with varying months
```

Pre-Processing Data Analysis (count of missing/ null values, redundant columns, etc.):

- After closely observing raw data, the conclusion we arrived is, data contains 8 consecutive months data for 12500 customers, while much information remains same for 8 months data like Name,Annual_Income,Num_Bank_Accounts, Interest_Rate,Num_of_Loan, Outstanding_Debt,some variables changes evry monthly Amount_invested_monthly, Monthly_Balance while there are some variables that dependent on previous values and increments like Num_of_Delayed_Payment, Num_Credit_Inquiries, Credit_History_Age.

- There are also outliers and wrong information present like Payment_Behaviour, negative values for Num_Bank_Accounts, Num_of_Loan and extremely high value for Amount_invested_monthly etc.

- For variables where information remain same throughout 8 months, we will calculate mode and replace null and wrong values with mode.

- For variables that vary monthly, we will replace outliers and null values using the mode calculated for each customer_id.

- For variables that increments, we will use past and future values to impute missing values using forward and backward fill.

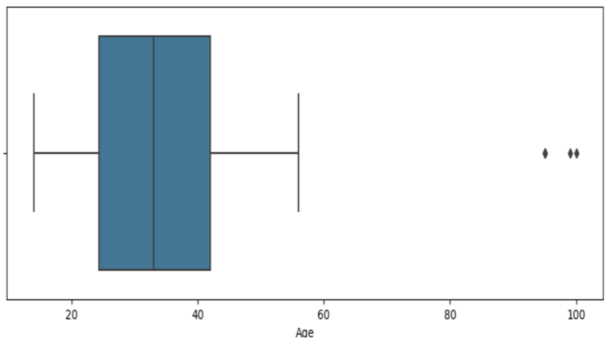
- There are many variables that should be numerical type but present as object type like Num_of_Loan, Num_of_Delayed_Payment, Changed_Credit_Limit, Outstanding_Debt, Amount_invested_monthly. These variables contain mixed types some instances as int/float while others as string.

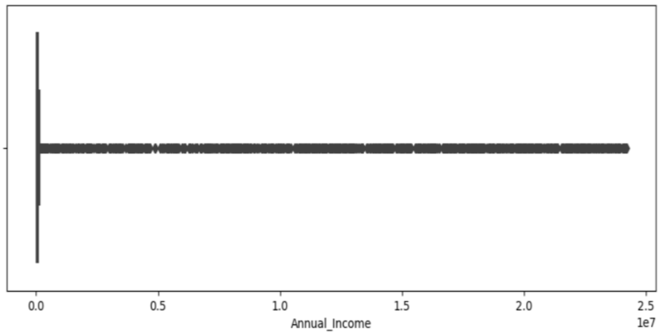
- ID, Customer_ID, Month, Name, SSN, Occupation, Type_of_Loan will be dropped as they are not useful for classification task.

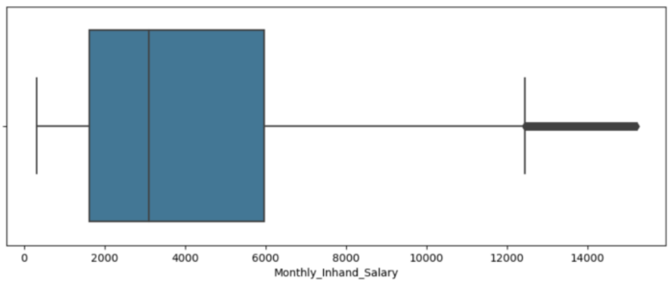
3 Data Exploratory Analysis (EDA)

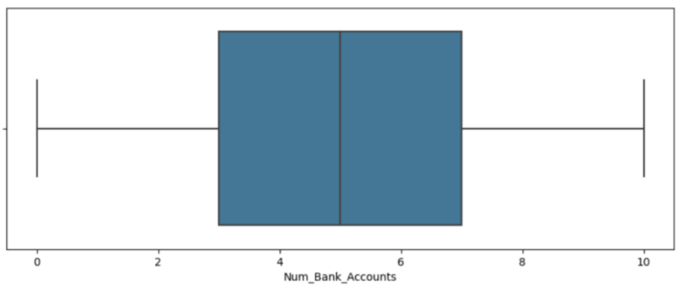
Univariate Analysis:

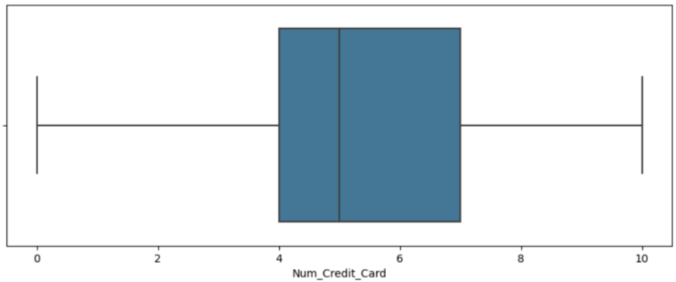
- **Boxplot**

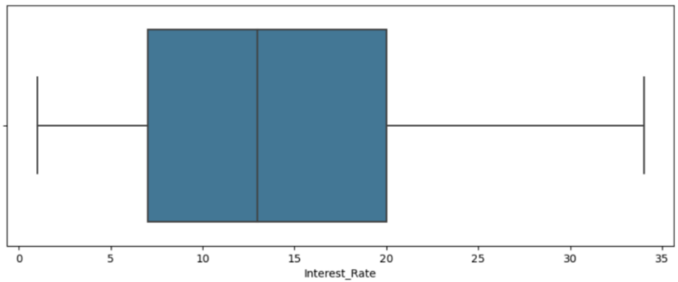


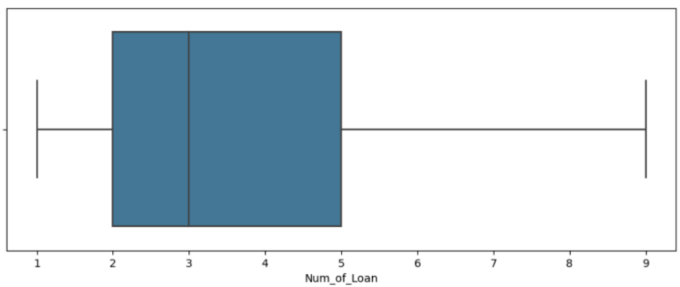


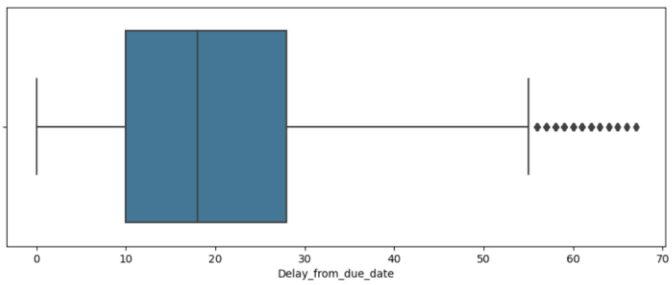


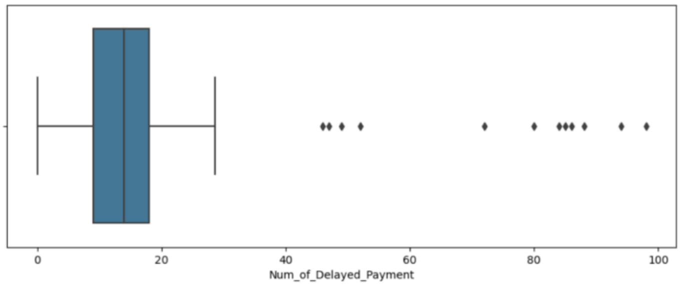


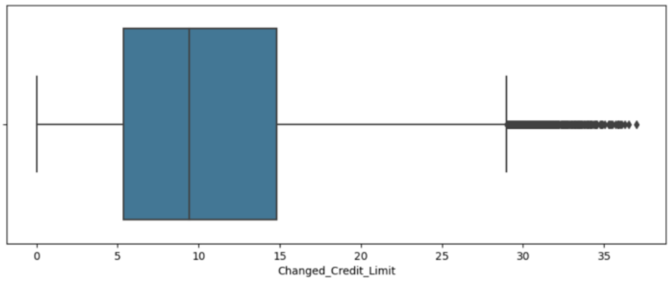


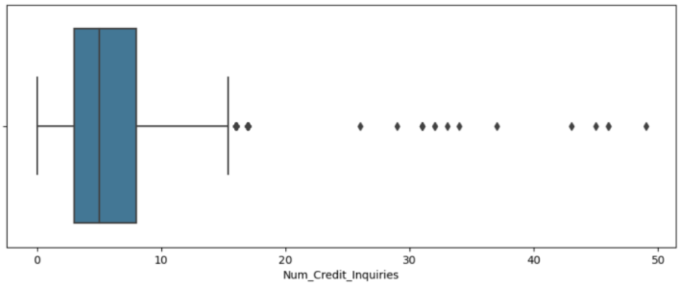


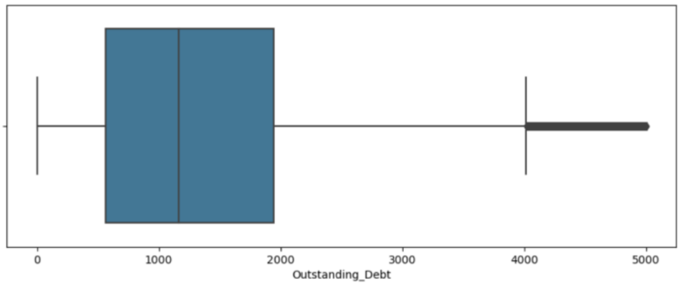


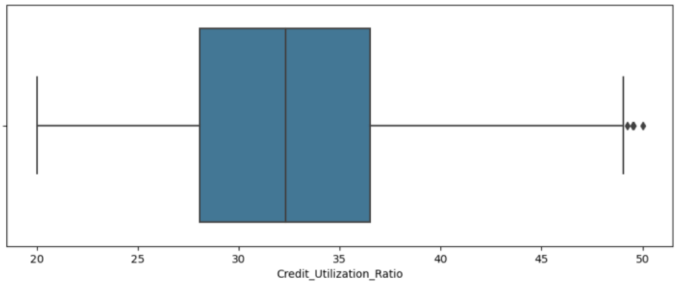


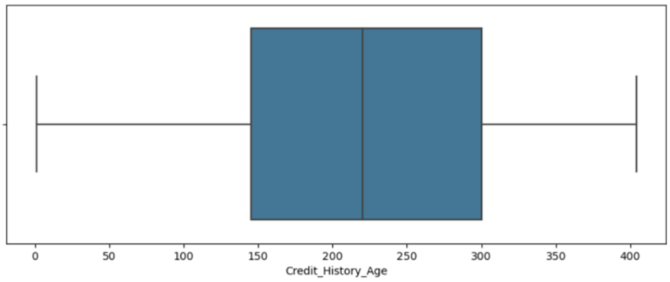


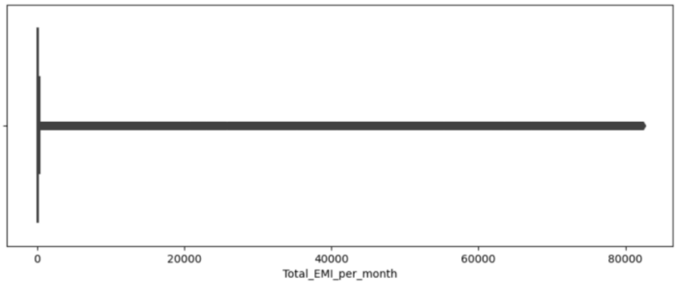


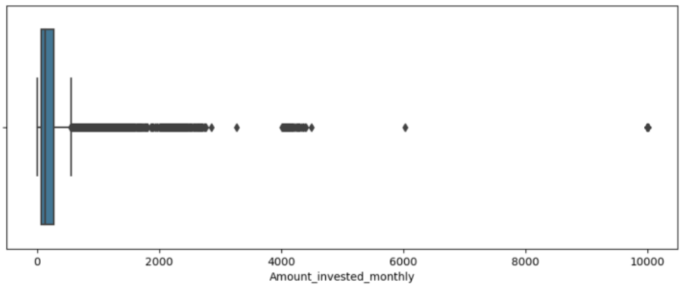


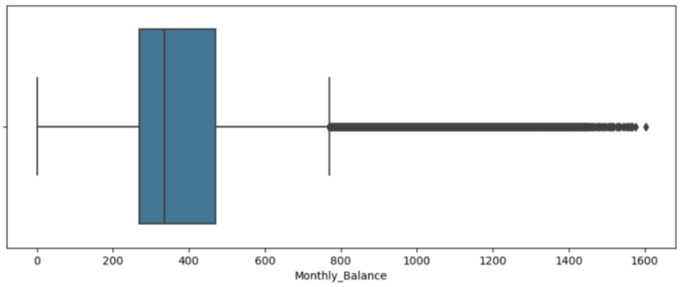






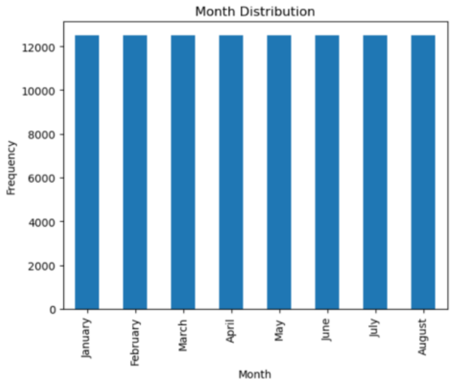


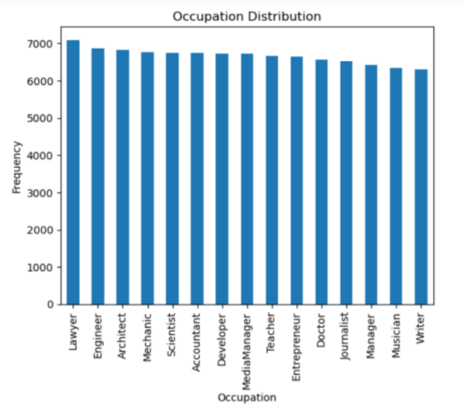


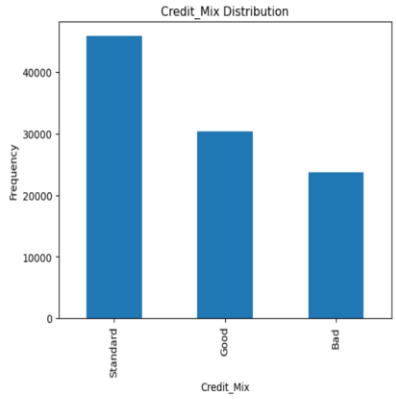


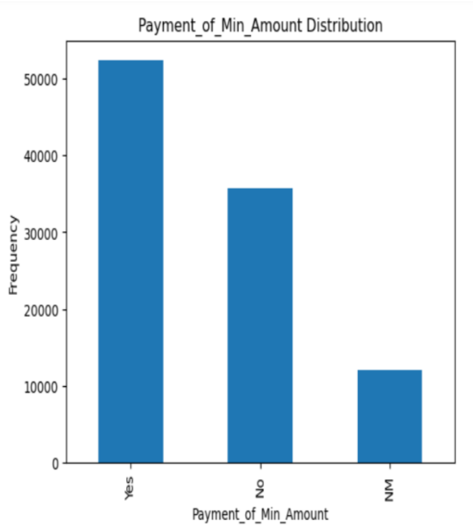
- Features that have positive skew are Age, Annual_Income, Monthly_Inhand_Salary, Num_Credit_Card, Interest_Rate, Num_of_Loan, Delay_from_due_date, Changed_Credit_Limit, Num_Credit_Inquiries, Outstanding_Debt, Total_EMI_per_month, Amount_invested_monthly, Monthly_Balance
- Features that have negetive skew is Num_Bank_Accounts
- Features that have normal distribution are Num_of_Delayed_Payment, Credit_Utilization_Ratio, Credit_History_Age
- Total_EMI_per_month and annual_income have extreme values.

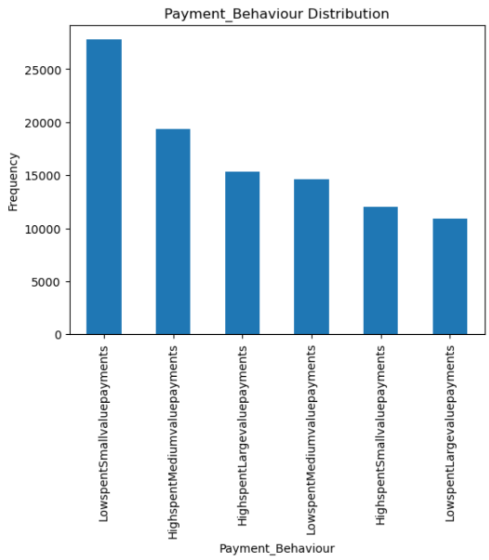
Countplot







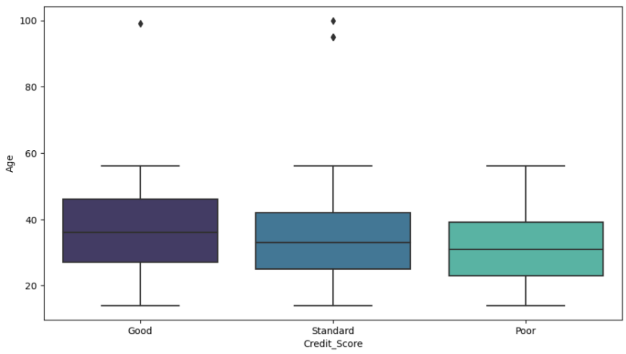


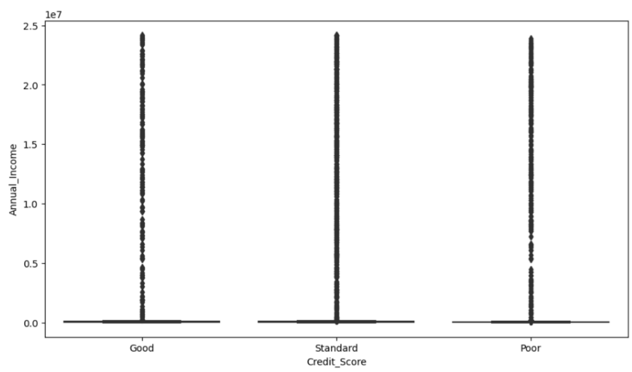


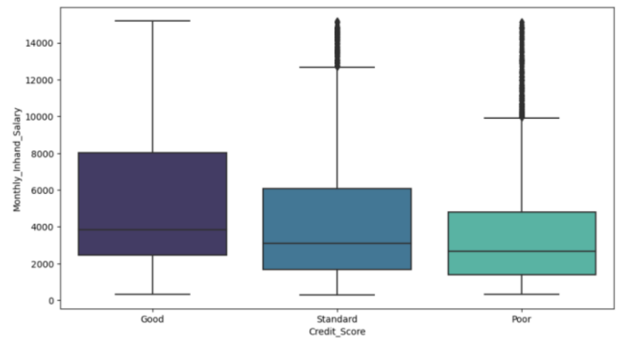
- The data of each customer for all the 6 months are available.
- The data of lawyers are most in the data and the data of musicians and writers are least.
- The people with credit score as Standard is the most and Bad is the least.
- Most of them have done the minimum payment and less number of people have not mentioned whether they did or not.
- Most of the people have the behavior of Low_spent_small_value_payments and very less people does low_spent_large _value_payments.

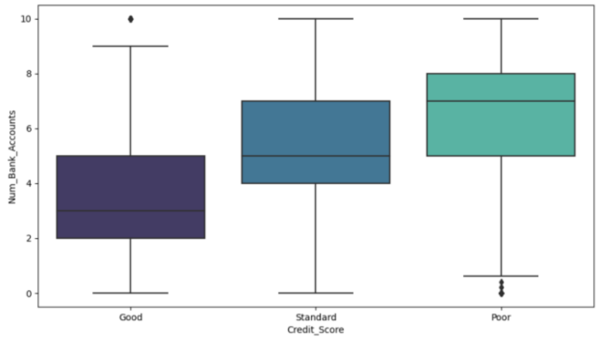
Bivariate Analysis:

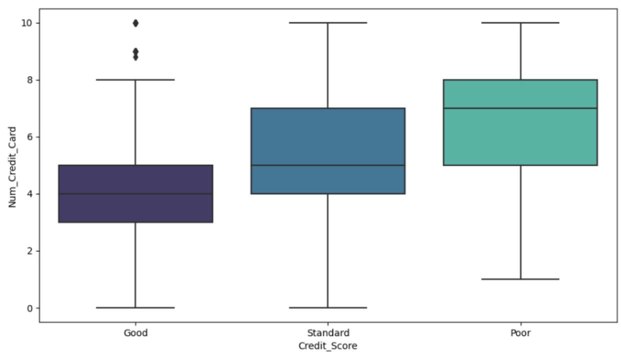
- **Boxplot**

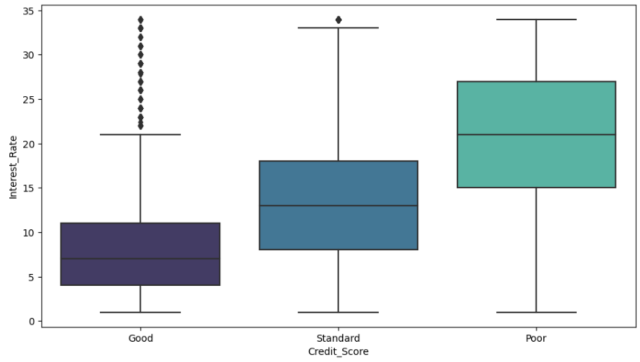


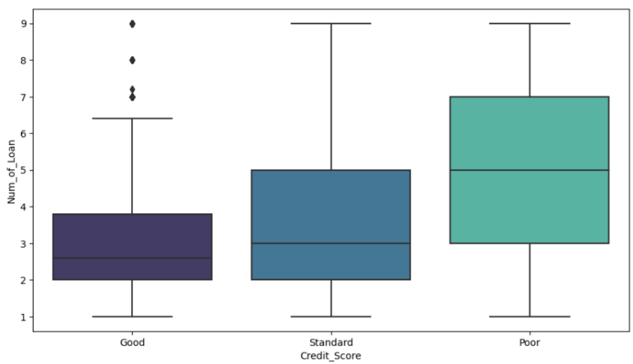


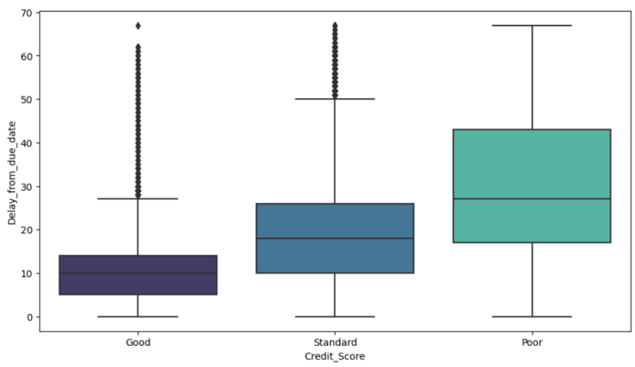

Monthly_Inhand_Salary
0
2000
4000
6000
8000
10000
12000
14000
Good
Standard
Poor
Credit_Score

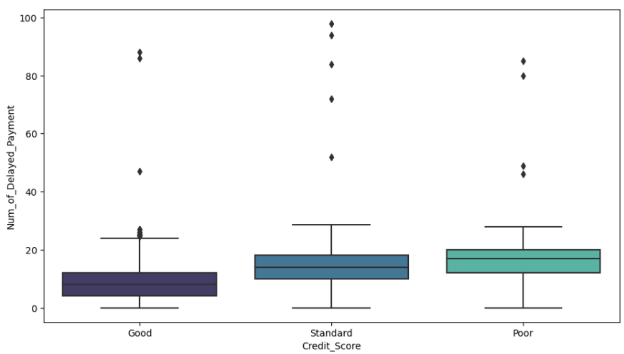

Num_Bank_Accounts
0
2
4
6
8
10
Good
Standard
Poor
Credit_Score

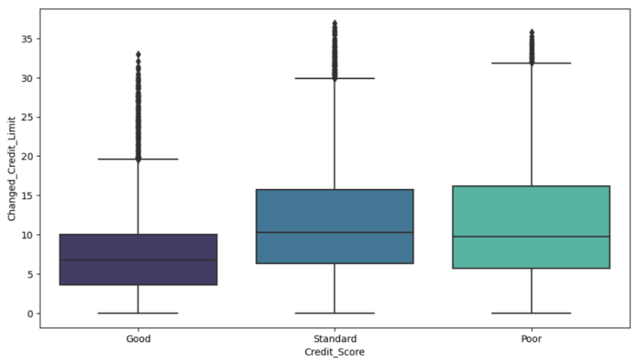

Num_Credit_Card
0
2
4
6
8
10
Good
Standard
Poor
Credit_Score

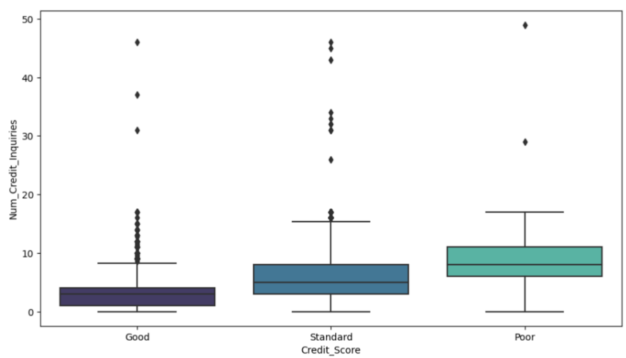

Interest_Rate
0
5
10
15
20
25
30
35
Good
Standard
Poor
Credit_Score

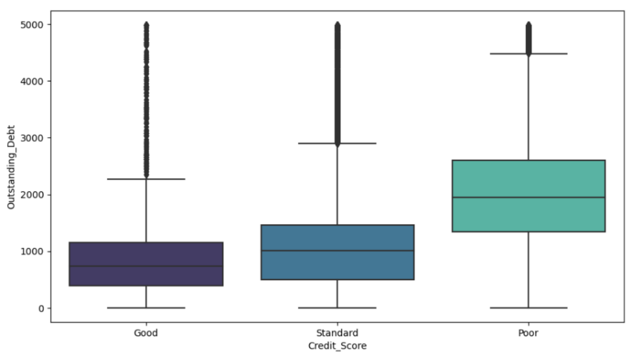

Num_of_Loan
1
2
3
4
5
6
7
8
9
Good
Standard
Poor
Credit_Score

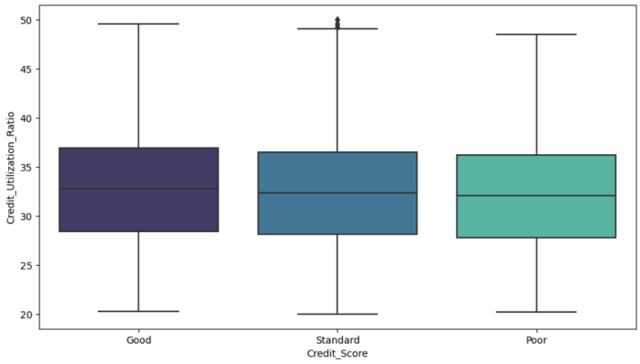

Delay_from_due_date
0
10
20
30
40
50
60
70
Good
Standard
Poor
Credit_Score

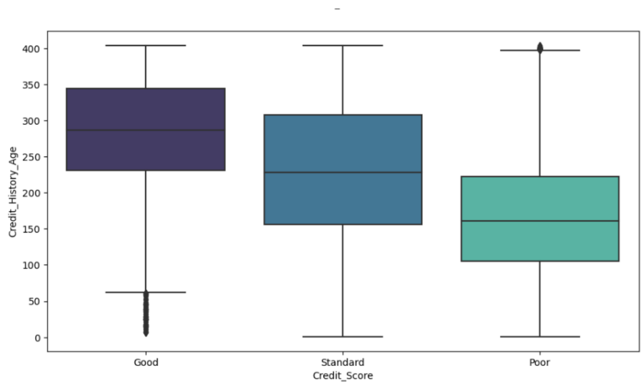

Num_of_Delayed_Payment
0
20
40
60
80
100
Good
Standard
Poor
Credit_Score

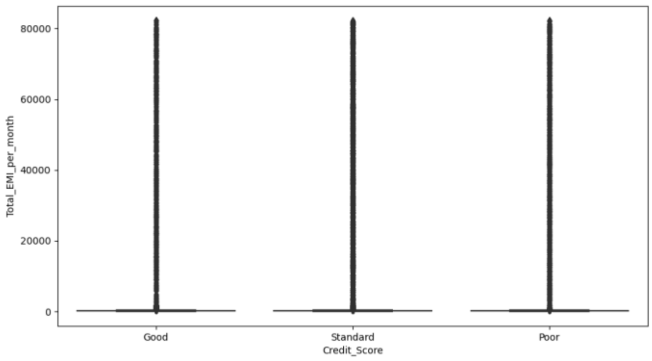

Changed_Credit_Limit
0
5
10
15
20
25
30
35
Good
Standard
Poor
Credit_Score

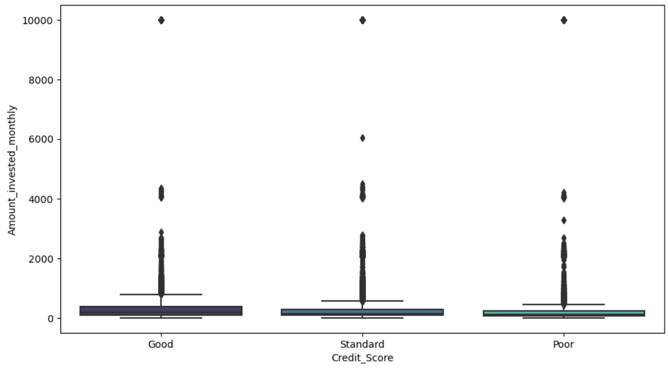

Num_Credit_Inquiries
0
10
20
30
40
50
Good
Standard
Poor
Credit_Score

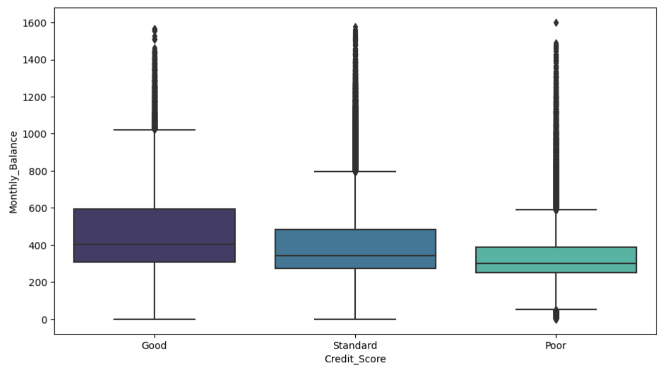

Outstanding_Debt
0
1000
2000
3000
4000
5000
Good
Standard
Poor
Credit_Score





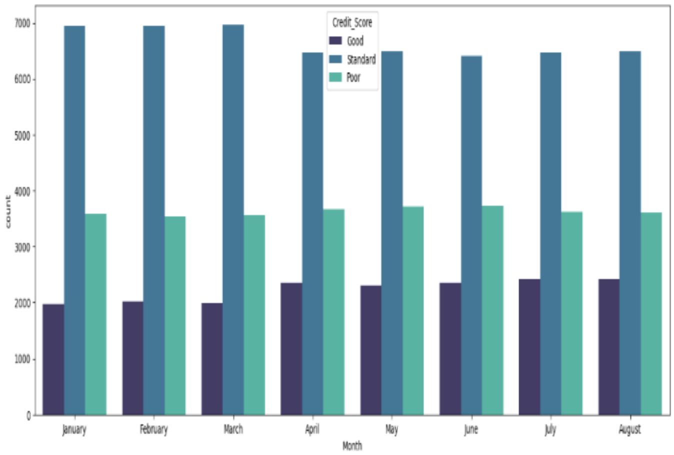


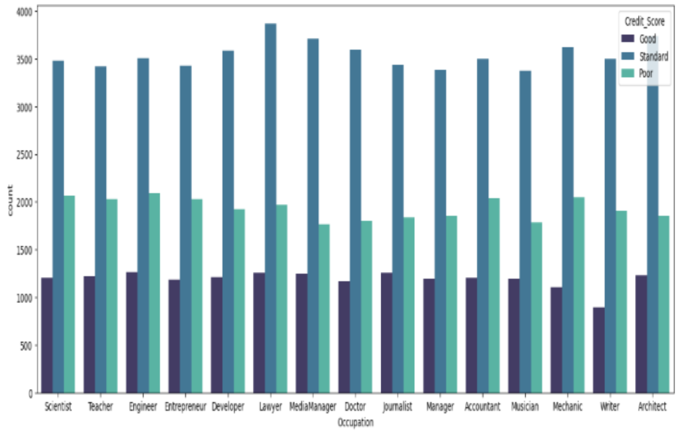


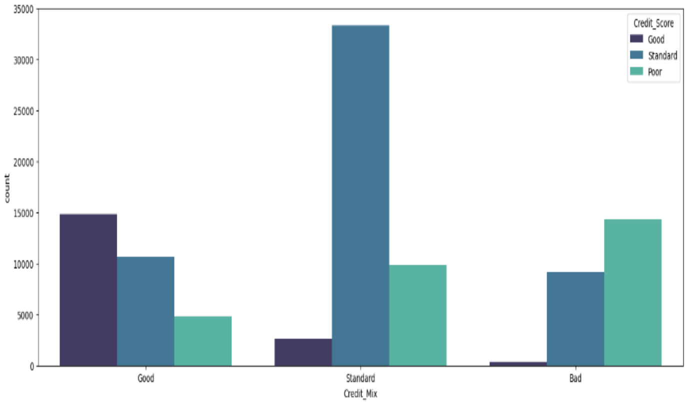


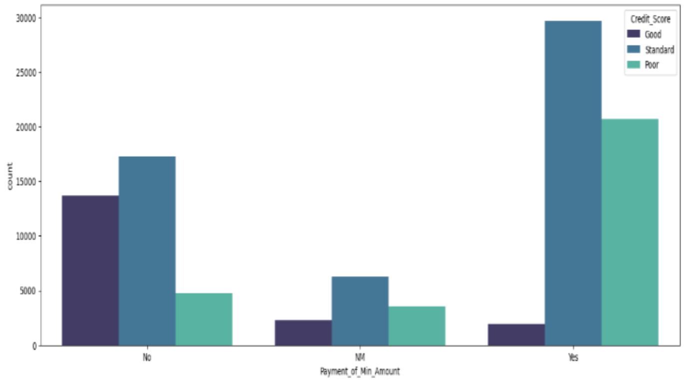
- All the credit score features are equally shown by the people who comes under the age group of 10 – 60.
- Those having the tree types of credit score have annual incomes in very different ranges.
- The standard and poor credit score have outliers in monthly inhand salary.
- Those having good credit score have more number of bank account, number of credit cards, number of loans, interest rate, interest rate, delay from due date, compare to others.
- Number of delay payment is almost similar for all the credit score.
- The median of changed credit limit is almost same for standard and poor credit score and all of those have positive outliers.
- The maximum and minimum value of number of credit inquiries are same for each credit score.
- The minimum, maximum and median of credit utilization ratio for credit score types are same.
- Age has negative outliers in good credit score and poor have positive outliers.
- Total EMI per month have a large number of outliers in each category.
- The amount invested monthly are ranging to very different values for all credit score categories.
- Those with bad credit score are having very different monthly balance amounts and the IQR is less compared to others.

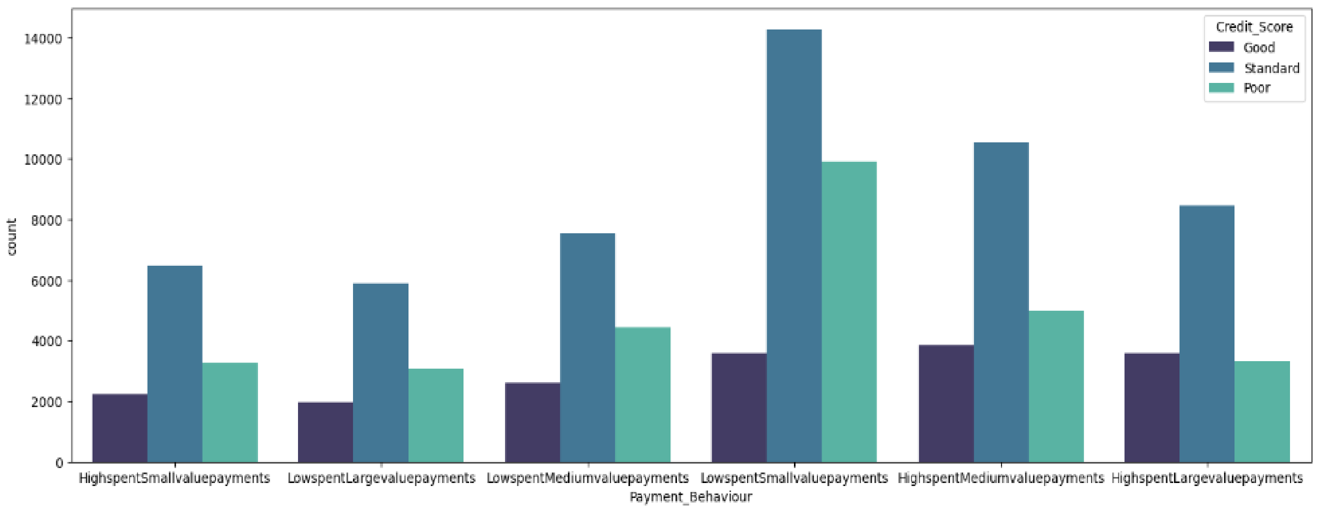
Countplot





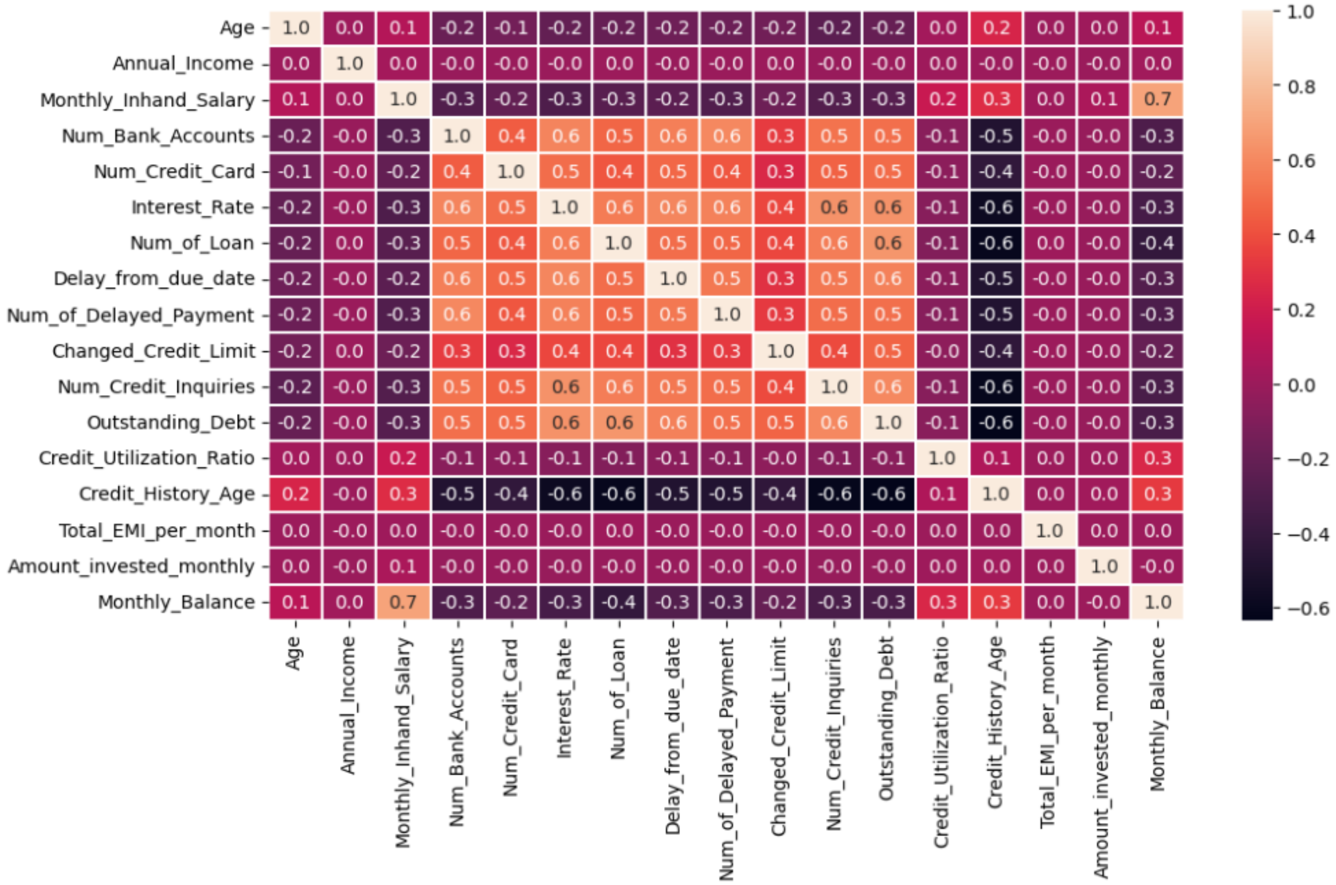






- Scientists, teachers, engineers, developers, lawyers, media manager, doctors, journalists, architects have a chance to get good credit score.
- Lawyers, mechanics, architects, writers have a average or standard credit score.
- Mechanics, accountants, entrepreneurs have bad credit score.
- Those who have not done minimum amount payment have a good credit score.
- People who did minimum payment amount have a poor credit score.
- Those whose behavior is High spending medium value payments have a good credit score.
- Those whose behavior is Low spending small value payments have standard credit score.
- People having bad credit score have those who low spending in small value payments.

Multivariate Analysis:



- The data is not showing much multi-collinearity.
- Monthly balance and monthly In-hand salary are showing high positive correlation.

Outliers

In [230]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = ((df < lower_bound) | (df > upper_bound)).any(axis=1)
outliers
```

Out[230]:
```
0        False
1        False
2        False
3        False
4        False
         ...
99995    False
99996    False
99997    False
99998    False
99999    False
Length: 100000, dtype: bool
```

4.Data transformation

Filter numerical and categorical variables.

In [231]:
```
df_target = df['Credit_Score']

df_feature = df.drop('Credit_Score', axis = 1)
```

In [232]:
```
df_num = df_feature.select_dtypes(include = [np.number])

df_num.columns
```

Out[232]:
```
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', 'Outstanding_Debt', 'Credit_Utilization_Ratio',
       'Credit_History_Age', 'Total_EMI_per_month', 'Amount_invested_monthly',
       'Monthly_Balance'],
      dtype='object')
```

In [233]:
```
df_cat = df_feature.select_dtypes(include = [np.object])

df_cat.columns
```

Out[233]:
```
Index(['Occupation', 'Credit_Mix', 'Payment_of_Min_Amount',
       'Payment_Behaviour'],
      dtype='object')
```

Scaling

In [234]:
```
print("Original DataFrame:")
print(df_num)

scaler = PowerTransformer(method='yeo-johnson', standardize=True)

df_num = pd.DataFrame(scaler.fit_transform(df_num), columns=df_num.columns)

print("\nDataFrame after Standard Scaling:")
print(df_num)
```

```
Original DataFrame:
             Age  Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  \
0      23.000000    19114.120000            1824.843333           3.000000
1      23.000000    19114.120000            1576.640583           3.000000
2      23.000000    19114.120000            1659.374833           3.000000
3      23.000000    19114.120000            1493.906333           3.000000
4      23.000000    19114.120000            1824.843333           3.000000
...          ...             ...                    ...                ...
99995  25.000000    39628.990000            3359.415833           4.000000
99996  25.000000    39628.990000            3359.415833           4.000000
99997  25.000000    39628.990000            3359.415833           4.000000
99998  25.000000    39628.990000            3359.415833           4.000000
99999  25.000000    39628.990000            3359.415833           4.000000

       Num_Credit_Card  Interest_Rate  Num_of_Loan  Delay_from_due_date  \
0             4.000000       3.000000     4.000000             3.000000
1             4.000000       3.000000     4.000000             1.000000
2             4.000000       3.000000     4.000000             3.000000
3             4.000000       3.000000     4.000000             5.000000
```

Encoding

In [235]:
```
dummy_var = pd.get_dummies(data = df_cat, drop_first = True)
```

In [236]:
```
from sklearn.preprocessing import LabelEncoder
```

In [237]:
```
lb=LabelEncoder()
df_target=lb.fit_transform(df_target)
```

Concatenate numerical and dummy encoded categorical variables.

In [238]:
```
X = pd.concat([df_num, dummy_var], axis = 1)

X.head()
```

Out[238]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date	Num_of_Delay
0	-0.951949	-0.798469	-0.690990	-0.918537	-0.717452	-1.566584	0.300885	-1.643007	
1	-0.951949	-0.798469	-0.864984	-0.918537	-0.717452	-1.566584	0.300885	-2.132256	
2	-0.951949	-0.798469	-0.804360	-0.918537	-0.717452	-1.566584	0.300885	-1.643007	
3	-0.951949	-0.798469	-0.928591	-0.918537	-0.717452	-1.566584	0.300885	-1.296232	
4	-0.951949	-0.798469	-0.690990	-0.918537	-0.717452	-1.566584	0.300885	-1.150832	

Train-Test Split

Before applying various classification techniques to predict the admission status of the student, let us split the dataset in train and test set.

In [239]:
```
X_train, X_test, y_train, y_test = train_test_split(X, df_target, random_state = 4, test_size = 0.3)

print('X_train', X_train.shape)
print('y_train', y_train.shape)

print('X_test', X_test.shape)
print('y_test', y_test.shape)
```

```
X_train (70000, 40)
y_train (70000,)
X_test (30000, 40)
y_test (30000,)
```

Feature Engineering

1. Transformation using Power Transformer with Yeo-Johnson method:

Power transformations, such as the Yeo-Johnson transformation, is an effective in handling skewed data and outliers. They make the distribution of the features more symmetric and bring them closer to a normal distribution.

2. Base Model with All Features:

We have used all the features without feature selection for building base model. This is a good approach to understand how well the model performs with the entire feature set before considering feature selection techniques.

3. Feature Selection:

This step involves choosing a subset of the most relevant features for the model. It can help improve model interpretability, reduce overfitting, and potentially enhance model performance. We have dropped some features that are not having significant relationship with the target.

4. Dimension Reduction Methods:

If needed, we are open to exploring dimension reduction methods. Dimension reduction techniques, such as visualization techniques can be used to reduce the number of features. They are very useful when dealing with a high-dimensional dataset.

Modeling and Evaluation

Using certain modelling techniques such as KNN Neighbor classifier, Logistic regression, we will classify data.

In [240]:
```
perf_score=pd.DataFrame(columns=['Model','Accuracy','Recall','Precision','F1 Score'])
```

In [241]:
```
pd.options.display.max_columns = None
pd.options.display.max_rows = None

pd.options.display.float_format = '{:.6f}'.format

from sklearn.model_selection import train_test_split


import statsmodels.api as sm

from sklearn.preprocessing import StandardScaler

from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score

from sklearn.feature_selection import RFE
```

In [242]:
```
pd.options.display.max_columns = None
pd.options.display.max_rows = None

pd.options.display.float_format = '{:.6f}'.format



from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import classification_report,f1_score,recall_score,precision_score,accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay  #****


import pydoc
from IPython.display import Image
import plotly.graph_objects as go
from sklearn.tree import DecisionTreeClassifier
```

In [243]:
```
def per_measures(model,test,pred):
    accuracy=accuracy_score(test,pred)
    f1score=f1_score(test,pred, average='weighted')
    recall=recall_score(test,pred, average='weighted')
    precision=precision_score(test,pred, average='weighted')
    return(accuracy,recall,precision,f1score)
```

In [244]:
```
def update_performance (name, model,test,pred):
    global perf_score

    perf_score = perf_score.append({'Model'      : name,
                                    'Accuracy'   : per_measures(model,test,pred)[0],
                                    'Recall'     : per_measures(model,test,pred)[1],
                                    'Precision'  : per_measures(model,test,pred)[2],
                                    'F1 Score'   : per_measures(model,test,pred)[3]

                                     },
                                   ignore_index=True)
```

In [245]:
```
lr=LogisticRegression()
lr.fit(X_train,y_train)
```

Out[245]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [246]:
```
coef_logit=lr.coef_
coef_logit
```

Out[246]:
```
array([[ 3.50706445e-02, -1.16668125e-02,  5.14927041e-02,
         6.29797382e-02, -3.52709417e-01, -1.58317034e-01,
        -5.20382637e-02, -2.20179425e-01,  1.15440796e-01,
         7.96572487e-02, -6.78699968e-02, -1.20524923e-01,
        -1.66686118e-02,  9.33983245e-02,  2.66560553e-02,
         8.09328954e-03, -1.47298173e-01, -2.02834342e-02,
         3.15693655e-02,  8.37918010e-02,  8.04614409e-02,
        -2.98114350e-02,  1.31584741e-01,  5.14768022e-03,
         5.01041802e-02, -5.97123535e-02,  5.67355794e-02,
         5.86753658e-02, -3.88724932e-02,  6.47296514e-02,
        -1.15853865e-01,  1.33526488e+00,  1.53393315e-01,
         4.79115886e-02, -2.15725852e-01, -7.11381845e-02,
        -1.51538669e-01, -2.49883752e-01, -3.13402161e-01,
        -4.82583591e-01],
       [-4.29714639e-02, -1.55824691e-02, -5.28701328e-02,
         1.23965477e-02,  2.85591150e-01,  3.67109005e-01,
         5.86123132e-02,  2.91782067e-01, -4.29160759e-02,
        -2.47717257e-01,  2.49891635e-01,  2.36592603e-01,
         9.09564455e-03, -1.05008952e-01,  2.03646867e-02,
         8.76419048e-03,  1.34785697e-01, -5.23658126e-02,
        -8.91950511e-02, -1.43494658e-01, -6.02085156e-02,
        -1.29173114e-02, -1.24007026e-01, -6.66611388e-02,
        -8.61874872e-02, -1.94605877e-02, -1.37148750e-01,
        -1.00071511e-01,  2.67864404e-02, -2.85562793e-02,
         3.46290471e-02,  7.91493263e-01,  1.75008354e-01,
        -2.82126172e-01,  1.91264687e-01,  8.31588214e-02,
         1.40836663e-01,  2.45093970e-01,  3.39969851e-01,
         4.51874035e-01],
       [ 7.90081937e-03,  2.72492816e-02,  1.37742863e-03,
        -7.53762860e-02,  6.71182667e-02, -2.08791972e-01,
        -6.57404947e-03, -7.16026429e-02, -7.25247200e-02,
```

In [247]:
```
coef_odd=np.exp(coef_logit)
```

In [248]:
```
lr.get_params()
```

Out[248]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [250]:
```
ypred_lr_tr=lr.predict(X_train)
print(classification_report(y_train,ypred_lr_tr))
print(confusion_matrix(y_train,ypred_lr_tr))
```

```
              precision    recall  f1-score   support

           0       0.56      0.66      0.61     12346
           1       0.64      0.56      0.60     20315
           2       0.71      0.71      0.71     37339

    accuracy                           0.66     70000
   macro avg       0.64      0.65      0.64     70000
weighted avg       0.66      0.66      0.66     70000

[[ 8176   517  3653]
 [ 1672 11448  7195]
 [ 4830  5916 26593]]
```

In [251]:
```
ypred_lr=lr.predict(X_test)
print(classification_report(y_test,ypred_lr))
print(confusion_matrix(y_test,ypred_lr))
```

```
              precision    recall  f1-score   support

           0       0.56      0.66      0.60      5482
           1       0.64      0.57      0.60      8683
           2       0.71      0.71      0.71     15835

    accuracy                           0.66     30000
   macro avg       0.64      0.65      0.64     30000
weighted avg       0.66      0.66      0.66     30000

[[ 3600   216  1666]
 [  769  4942  2972]
 [ 2053  2533 11249]]
```

In [252]:
```
update_performance(name='LogisticReg-Base',
                  model=lr,
                  test=y_test,
                  pred=ypred_lr)
perf_score
```

Out[252]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590

In [253]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
ypred_knn=knn.predict(X_test)
```

In [254]:
```
knn.get_params()
```

Out[254]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [255]:
```
update_performance(name='KNeighborsClassifier',
                  model=knn,
                  test=y_test,
                  pred=ypred_knn)
perf_score
```

Out[255]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304

In [256]:
```
ypred_knn_tr=knn.predict(X_train)
```

In [257]:
```
f1_score(y_train, ypred_knn_tr, average='weighted')
```

Out[257]:
```
0.8339491792489809
```

In [258]:
```
dt=DecisionTreeClassifier(random_state=10)
dt.fit(X_train,y_train)
ypred_dt=dt.predict(X_test)
```

In [259]:
```
dt.get_params()
```

Out[259]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 10,
 'splitter': 'best'}
```

In [260]:
```
update_performance(name='Decision Tree-Gini',
                  model=dt,
                  test=y_test,
                  pred=ypred_dt)
perf_score
```

Out[260]:

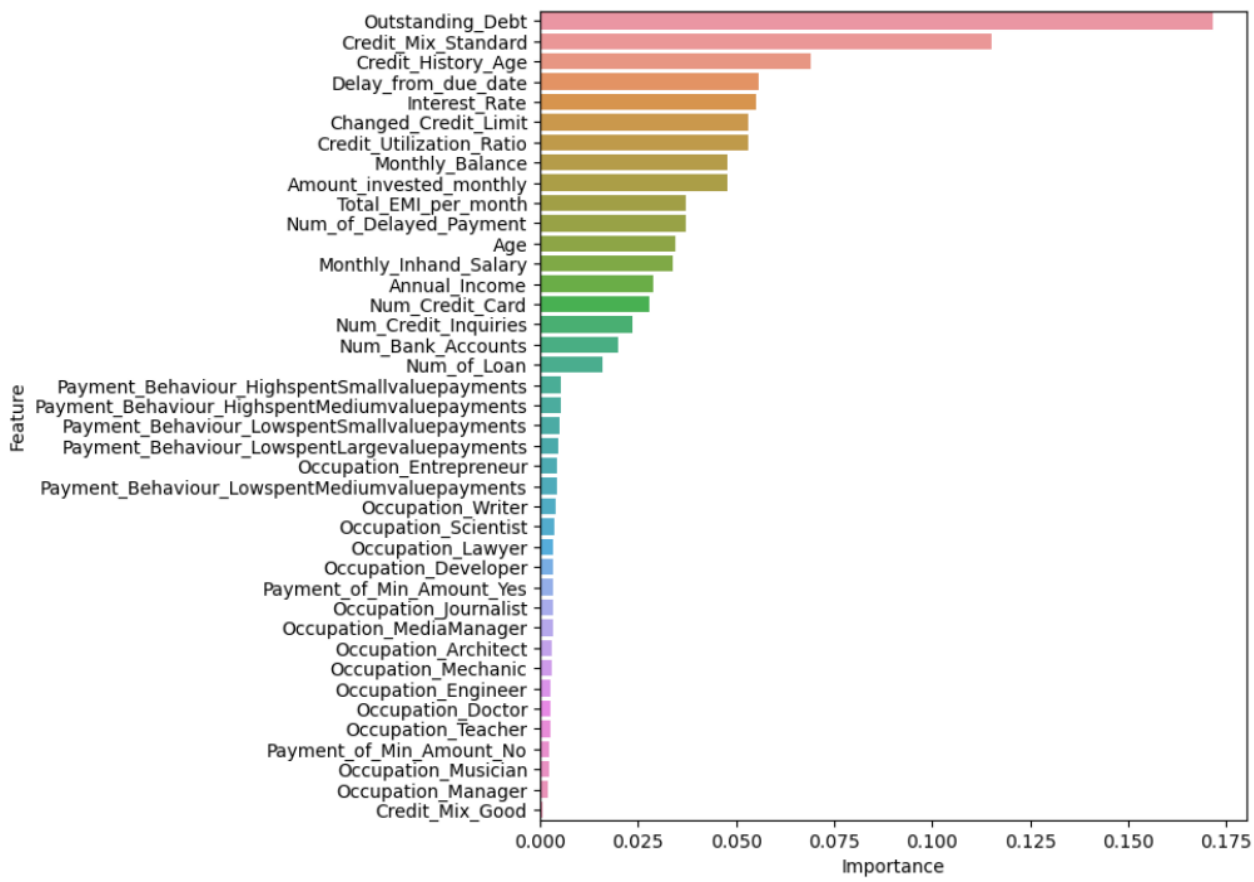
	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342

In [261]:
```
ypred_dt_tr=dt.predict(X_train)
f1_score(y_train, ypred_dt_tr, average='weighted')
```

Out[261]:
```
1.0
```

In [262]:
```
feature_imp=pd.DataFrame()
feature_imp['Feature']=X_train.columns
feature_imp['Importance']=dt.feature_importances_
```

In [264]:
```
plt.figure(figsize=(7,8))
feature_imp=feature_imp.sort_values('Importance',ascending=False)
sns.barplot(x='Importance',y='Feature',data=feature_imp)
plt.show()
```



In [265]:
```
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(X_train,y_train)
ypred_gnb=gnb.predict(X_test)
```

In [266]:
```
gnb.get_params()
```

Out[266]: {'priors': None, 'var_smoothing': 1e-09}

In [267]:
```
update_performance(name='Gaussian NB',
                   model= gnb,
                   test=y_test,
                   pred=ypred_gnb)
perf_score
```

Out[267]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885

In [269]:
```
ypred_gnb_tr=gnb.predict(X_train)
f1_score(y_train, ypred_gnb_tr, average='weighted')
```

Out[269]: 0.6522402023053747

In [270]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [273]:
```
rf=RandomForestClassifier(random_state=10)
rf.fit(X_train,y_train)
ypred_rf=rf.predict(X_test)
```

In [274]:
```
rf.get_params()
```

Out[274]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 10,
 'verbose': 0,
 'warm_start': False}
```

In [275]:
```
update_performance(name='Random Forest',
                   model=rf,
                   test=y_test,
                   pred=ypred_rf)
perf_score
```

Out[275]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588

In [276]:
```
ypred_rf_tr=rf.predict(X_train)
f1_score(y_train, ypred_rf_tr, average='weighted')
```

Out[276]: 1.0

In [277]:
```
from sklearn.ensemble import BaggingClassifier
dt=DecisionTreeClassifier(random_state=10)
bc=BaggingClassifier(dt)
bc.fit(X_train,y_train)
ypred_bc=bc.predict(X_test)
```

In [278]:
```
bc.get_params()
```

Out[278]:
```
{'base_estimator': 'deprecated',
 'bootstrap': True,
 'bootstrap_features': False,
 'estimator__ccp_alpha': 0.0,
 'estimator__class_weight': None,
 'estimator__criterion': 'gini',
 'estimator__max_depth': None,
 'estimator__max_features': None,
 'estimator__max_leaf_nodes': None,
 'estimator__min_impurity_decrease': 0.0,
 'estimator__min_samples_leaf': 1,
 'estimator__min_samples_split': 2,
 'estimator__min_weight_fraction_leaf': 0.0,
 'estimator__random_state': 10,
 'estimator__splitter': 'best',
 'estimator': DecisionTreeClassifier(random_state=10),
 'max_features': 1.0,
 'max_samples': 1.0,
 'n_estimators': 10,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [279]:
```
update_performance(name='Bagging Classifier-dt',
                   model=bc,
                   test=y_test,
                   pred=ypred_bc)
perf_score
```

Out[279]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617

In [280]:
```
ypred_bc_tr=bc.predict(X_train)
f1_score(y_train, ypred_bc_tr, average='weighted')
```

Out[280]:
```
0.9856073057922181
```

In [281]:
```
from sklearn.ensemble import AdaBoostClassifier
abcl=AdaBoostClassifier(dt,random_state=10)
abcl.fit(X_train,y_train)
ypred_abcl=abcl.predict(X_test)
```

In [283]:
```
abcl.get_params()
```

Out[283]:
```
{'algorithm': 'SAMME.R',
 'base_estimator': 'deprecated',
 'estimator__ccp_alpha': 0.0,
 'estimator__class_weight': None,
 'estimator__criterion': 'gini',
 'estimator__max_depth': None,
 'estimator__max_features': None,
 'estimator__max_leaf_nodes': None,
 'estimator__min_impurity_decrease': 0.0,
 'estimator__min_samples_leaf': 1,
 'estimator__min_samples_split': 2,
 'estimator__min_weight_fraction_leaf': 0.0,
 'estimator__random_state': 10,
 'estimator__splitter': 'best',
 'estimator': DecisionTreeClassifier(random_state=10),
 'learning_rate': 1.0,
 'n_estimators': 50,
 'random_state': 10}
```

In [284]:
```
update_performance(name='Adaboost-dt',
                   model=abcl,
                   test=y_test,
                   pred=ypred_abcl)
perf_score
```

Out[284]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502

In [285]:
```
ypred_abcl_tr=abcl.predict(X_train)
f1_score(y_train, ypred_abcl_tr, average='weighted')
```

Out[285]: 1.0

In [286]:
```
from sklearn.ensemble import StackingClassifier
```

In [287]:
```
lr=LogisticRegression()
knn=KNeighborsClassifier()
dt=DecisionTreeClassifier()
```

In [288]:
```
base_learners=[('lr_model',lr),('knn_model',knn),('DT_model',dt)]

stack=StackingClassifier(estimators=base_learners,final_estimator=GaussianNB())

stack.fit(X_train,y_train)
ypred_stack=stack.predict(X_test)
print(accuracy_score(y_test,ypred_stack))
```

0.7651

In [290]:
```
update_performance(name='stacking model',
                   model=stack,
                   test=y_test,
                   pred=ypred_stack)
perf_score
```

Out[290]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587

In [291]:
```
ypred_stack_tr=stack.predict(X_train)
f1_score(y_train, ypred_stack_tr, average='weighted')
```

Out[291]: 0.8771922315462581

In [292]:
```
from xgboost import XGBClassifier
```

In [293]:
```
xgb=XGBClassifier(random_state=10)
xgb.fit(X_train,y_train)
ypred_xgb=xgb.predict(X_test)
print(accuracy_score(y_test,ypred_xgb))
```

0.7684

In [295]:
```
ypred_xgb_tr=xgb.predict(X_train)
f1_score(y_train, ypred_xgb_tr, average='weighted')
```
Out[295]:
```
0.8326495575077902
```

In [296]:
```
update_performance(name='XGBoost',
                   model=xgb,
                   test=y_test,
                   pred=ypred_xgb)
perf_score
```
Out[296]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676

Choosing XGBoost as the best model before SMOTE. Weighted F1 score 76% in test data and 83% in training data.

In [297]:
```
from sklearn.model_selection import GridSearchCV,KFold
```

In [298]:
```
kf=KFold(n_splits=5,
         shuffle=True,
         random_state=0)
```

In [299]:
```
param_xg = {
    'learning_rate': [0.01, 0.1],
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator=xgb, param_grid=param_xg, scoring='accuracy', cv=kf, n_jobs=-1)

# Fit the model to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
```
```
Best Hyperparameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
```

In [302]:
```
xgb_tuned=XGBClassifier(random_state=10, colsample_bytree=1.0, learning_rate=0.1, max_depth= 5, n_estimators= 200, subsample=0.8)
xgb_tuned.fit(X_train,y_train)
ypred_xgb_tuned=xgb.predict(X_test)
```

In [304]:
```
update_performance(name='XGBoost_tuned',
                   model=xgb_tuned,
                   test=y_test,
                   pred=ypred_xgb_tuned)
perf_score
```
Out[304]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377

Logistic regression:

Logistic Regression is a valuable tool for credit score classification, offering numerous benefits in the financial industry. One key advantage is its interpretability, as the coefficients of the model provide clear insights into the impact of various factors on creditworthiness. The probabilistic output of Logistic Regression allows for a nuanced understanding of the likelihood of a borrower being creditworthy, enabling financial institutions to set appropriate risk thresholds. The model is adept at handling both linear and non-linear relationships between input features and credit outcomes, providing flexibility in capturing complex patterns. Logistic Regression's simplicity and ease of implementation make it a practical choice for credit scoring applications, particularly when dealing with large datasets. Additionally, its ability to naturally handle categorical variables and the option to apply regularization techniques contribute to its effectiveness in building accurate and reliable credit scoring models.

K-Neighbor Classifier:

The K-Nearest Neighbors (KNN) classifier offers distinct advantages in the realm of credit score classification. One notable benefit is its simplicity and ease of implementation. KNN makes minimal assumptions about the underlying data distribution, allowing it to handle complex relationships without relying on predefined models. The flexibility of KNN is particularly advantageous in credit scoring, where patterns of creditworthiness may not follow linear trends. Another notable advantage is its adaptability to varying dataset characteristics, making it suitable for datasets with mixed types of variables and different scales. The probabilistic nature of KNN can be valuable for assessing the uncertainty associated with credit score predictions. Additionally, KNN is effective in capturing local patterns and anomalies, providing a holistic view of credit risk. While the choice of the number of neighbors (k) is crucial, KNN stands out as a versatile and powerful tool for credit score classification, especially in scenarios where the underlying data relationships are intricate and nonlinear.

Decision Tree:

Decision Trees offer several advantages in the context of credit score classification. One notable benefit is their interpretability, as the decision-making process is represented graphically in a tree structure, making it easy to understand and explain to stakeholders. Decision Trees are adept at handling both categorical and numerical data, a common characteristic in credit scoring datasets. They naturally capture non-linear relationships and interactions between various features, providing a more nuanced representation of creditworthiness factors. Decision Trees are resilient to outliers and can automatically select relevant features, contributing to their robustness in diverse credit scoring scenarios. Their ability to handle missing values without the need for extensive data preprocessing is another practical advantage. Moreover, Decision Trees facilitate transparent risk assessment, enabling financial institutions to make informed lending decisions based on the explicit criteria defined by the tree's branches. Overall, Decision Trees are a valuable tool for credit score classification, combining interpretability, flexibility, and robust performance.

Random Forest

Random Forest, an ensemble learning method based on Decision Trees, offers compelling benefits for credit score classification. One of its primary advantages is high predictive accuracy. By constructing multiple Decision Trees and aggregating their predictions, Random Forest mitigates the risk of overfitting and provides a more robust and accurate model. The ensemble nature of Random Forest enables it to capture complex relationships and interactions within credit scoring data, offering improved performance compared to individual Decision Trees. Additionally, Random Forest handles both categorical and numerical features effortlessly, making it suitable for diverse credit scoring datasets. The algorithm is resilient to outliers and missing values, requiring minimal data preprocessing. Moreover, Random Forest provides valuable insights into feature importance, aiding in the identification of key factors influencing creditworthiness. Its scalability and parallelization capabilities make it suitable for handling large datasets efficiently. Overall, Random Forest is a powerful and versatile tool for credit score classification, offering a balance of accuracy, interpretability, and robust performance.

Bagging Claddifier DT:

The Bagging Classifier with Decision Trees (Bagging-DT) offers notable benefits in the realm of credit score classification. By leveraging an ensemble of Decision Trees trained on bootstrap samples of the data, Bagging-DT enhances predictive accuracy and robustness. The bagging technique helps mitigate overfitting by aggregating the diverse predictions of multiple trees. This results in a more stable and reliable credit scoring model. The combined decision-making power of individual trees enables Bagging-DT to capture intricate relationships within credit datasets, accommodating both categorical and numerical features effectively. The algorithm's ability to handle outliers and missing values contributes to its versatility, requiring minimal data preprocessing. Moreover, Bagging-DT provides insights into feature importance, aiding in the identification of key factors influencing creditworthiness. Its simplicity and ease of implementation make it a practical choice for credit scoring applications, offering a balanced trade-off between interpretability and predictive performance.

XG Boost DT:

XGBoost (Extreme Gradient Boosting) with Decision Trees (XGBoost-DT) is a powerful algorithm that brings several advantages to credit score classification. One key benefit is its exceptional predictive performance. XGBoost optimizes the strengths of ensemble learning, combining the predictive power of multiple Decision Trees in a boosting framework. This results in a highly accurate and robust credit scoring model. XGBoost handles both numerical and categorical features seamlessly, making it well-suited for the diverse nature of credit scoring datasets. Its regularization techniques help prevent overfitting, ensuring the model's generalizability to new data. Additionally, XGBoost provides insights into feature importance, aiding in the identification of key creditworthiness factors. The algorithm is computationally efficient and scalable, enabling it to handle large datasets with ease. Its flexibility, efficiency, and state-of-the-art performance make XGBoost-DT a popular and effective choice for credit score classification tasks.

Stacking Model:

The Stacking Model stands out as an advanced ensemble learning technique with compelling benefits for credit score classification. By combining the predictions of multiple base models, such as Decision Trees, Random Forests, or Gradient Boosting, the Stacking Model leverages the diverse strengths of each constituent model. This results in improved predictive accuracy and robustness, surpassing the performance of individual models. The stacking process allows the algorithm to adaptively weigh the contributions of each base model, optimizing the ensemble's overall performance. Moreover, the Stacking Model excels in capturing complex relationships and interactions within credit scoring datasets, providing a comprehensive understanding of creditworthiness factors. Its flexibility to incorporate various machine learning algorithms enhances versatility. Overall, the Stacking Model represents a sophisticated approach to credit score classification, offering a fine-tuned balance between accuracy, interpretability, and adaptability to diverse data patterns.

Performing SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a popular method for addressing class imbalance in machine learning datasets by oversampling the minority class. In Python, you can use the imbalanced-learn library, commonly known as imblearn, to apply SMOTE.

```
In [305]: from imblearn.over_sampling import SMOTE

In [306]: smote = SMOTE(sampling_strategy={0: 45000, 1: 45000, 2: 45000}, random_state=1)
          Xtrain_sm, ytrain_sm = smote.fit_resample(X_train,y_train)

In [307]: Xtrain_sm.shape

Out[307]: (135000, 40)

In [308]: lr_smote=LogisticRegression()
          lr_smote.fit(Xtrain_sm,ytrain_sm)

          y_pred_lr_smote=lr_smote.predict(X_test)

In [309]: update_performance(name='LogisticReg-Base-smote',
                             model=lr_smote,
                             test=y_test,
                             pred=y_pred_lr_smote)
          perf_score
```

Out[309]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122

```
In [310]: ypred_lrsm_tr=stack.predict(Xtrain_sm)
          f1_score(ytrain_sm, ypred_lrsm_tr, average='weighted')

Out[310]: 0.8790885957809256
```

In [311]:
```
knn.fit(Xtrain_sm,ytrain_sm)
ypred_knn_sm=knn.predict(X_test)
```

In [312]:
```
update_performance(name='KNeighborsClassifier after SMOTE',
                   model=knn,
                   test=y_test,
                   pred=ypred_knn_sm)
perf_score
```

Out[312]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122
11	KNeighborsClassifier after SMOTE	0.724900	0.724900	0.758632	0.724662

In [313]:
```
dt.fit(Xtrain_sm,ytrain_sm)
ypred_dt_sm=dt.predict(X_test)
```

In [314]:
```
update_performance(name='Decision Tree after SMOTE',
                   model=dt,
                   test=y_test,
                   pred=ypred_dt_sm)
perf_score
```

Out[314]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122
11	KNeighborsClassifier after SMOTE	0.724900	0.724900	0.758632	0.724662
12	Decision Tree after SMOTE	0.705800	0.705800	0.707683	0.706410

In [315]:
```
ypred_dtsm_tr=dt.predict(Xtrain_sm)
f1_score(ytrain_sm, ypred_dtsm_tr, average='weighted')
```

Out[315]: 1.0

In [316]:
```
gnb.fit(Xtrain_sm,ytrain_sm)
ypred_gnb_sm=gnb.predict(X_test)
```

In [317]:
```
update_performance(name='GaussianNB after SMOTE',
                   model=gnb,
                   test=y_test,
                   pred=ypred_gnb_sm)
perf_score
```

Out[317]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122
11	KNeighborsClassifier after SMOTE	0.724900	0.724900	0.758632	0.724662
12	Decision Tree after SMOTE	0.705800	0.705800	0.707683	0.706410
13	GaussianNB after SMOTE	0.641867	0.641867	0.704518	0.644041

In [318]:
```
ypred_gnsm_tr=gnb.predict(Xtrain_sm)
f1_score(ytrain_sm, ypred_gnsm_tr, average='weighted')
```

Out[318]: 0.6944268612937413

In [319]:
```
rf=RandomForestClassifier(random_state=10)
rf.fit(Xtrain_sm,ytrain_sm)
ypred_rf_sm=rf.predict(X_test)
```

In [320]:
```
update_performance(name='Random Forest after SMOTE',
                   model=rf,
                   test=y_test,
                   pred=ypred_rf_sm)
perf_score
```

Out[320]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122
11	KNeighborsClassifier after SMOTE	0.724900	0.724900	0.758632	0.724662
12	Decision Tree after SMOTE	0.705800	0.705800	0.707683	0.706410
13	GaussianNB after SMOTE	0.641867	0.641867	0.704518	0.644041
14	Random Forest after SMOTE	0.803200	0.803200	0.809339	0.803859

In [321]:
```python
ypred_rfsm_tr=gnb.predict(Xtrain_sm)
f1_score(ytrain_sm, ypred_rfsm_tr, average='weighted')
```
Out[321]: 0.6944268612937413

In [322]:
```python
xgb.fit(Xtrain_sm,ytrain_sm)
ypred_xgb_sm=xgb.predict(X_test)
print(accuracy_score(y_test,ypred_xgb_sm))
```
0.7296666666666667

In [323]:
```python
update_performance(name='XGBoost after SMOTE',
                   model=xgb,
                   test=y_test,
                   pred=ypred_xgb_sm)
perf_score
```
Out[323]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122
11	KNeighborsClassifier after SMOTE	0.724900	0.724900	0.758632	0.724662
12	Decision Tree after SMOTE	0.705800	0.705800	0.707683	0.706410
13	GaussianNB after SMOTE	0.641867	0.641867	0.704518	0.644041
14	Random Forest after SMOTE	0.803200	0.803200	0.809339	0.803859
15	XGBoost after SMOTE	0.729667	0.729667	0.749076	0.732567

In [329]:
```python
update_performance(name='Adaboost-dt after SMOTE',
                   model=abcl,
                   test=y_test,
                   pred=ypred_abclsm)
perf_score
```
Out[329]:

	Model	Accuracy	Recall	Precision	F1 Score
0	LogisticReg-Base	0.659700	0.659700	0.662156	0.659590
1	KNeighborsClassifier	0.749867	0.749867	0.751004	0.750304
2	Decision Tree-Gini	0.721400	0.721400	0.721314	0.721342
3	Gaussian NB	0.650767	0.650767	0.708344	0.653885
4	Random Forest	0.805600	0.805600	0.806000	0.805588
5	Bagging Classifier-dt	0.789400	0.789400	0.791126	0.789617
6	Adaboost-dt	0.721533	0.721533	0.721519	0.721502
7	stacking model	0.765100	0.765100	0.775801	0.765587
8	XGBoost	0.768400	0.768400	0.769101	0.768676
9	XGBoost_tuned	0.743767	0.743767	0.746084	0.744377
10	LogisticReg-Base-smote	0.659567	0.659567	0.686751	0.663122
11	KNeighborsClassifier after SMOTE	0.724900	0.724900	0.758632	0.724662
12	Decision Tree after SMOTE	0.705800	0.705800	0.707683	0.706410
13	GaussianNB after SMOTE	0.641867	0.641867	0.704518	0.644041
14	Random Forest after SMOTE	0.803200	0.803200	0.809339	0.803859
15	XGBoost after SMOTE	0.729667	0.729667	0.749076	0.732567
16	Bagging Classifier-dt after SMOTE	0.796533	0.796533	0.803017	0.796810
17	Adaboost-dt after SMOTE	0.702800	0.702800	0.704693	0.703424

```
In [330]: ypred_abclsm_tr=abcl.predict(Xtrain_sm)
          f1_score(ytrain_sm, ypred_abclsm_tr, average='weighted')

Out[330]: 1.0

In [331]: lr=LogisticRegression()
          knn=KNeighborsClassifier()
          dt=DecisionTreeClassifier()

In [332]: base_learners=[('lr_model',lr),('knn_model',knn),('DT_model',dt)]

          stack=StackingClassifier(estimators=base_learners,final_estimator=GaussianNB())

          stack.fit(Xtrain_sm,ytrain_sm)
          ypred_stack_sm=stack.predict(X_test)
          print(accuracy_score(y_test,ypred_stack_sm))

          0.7674666666666666
```

Model Comparison

#	Model Name	Parameters Used	Before Applying SMOTE on Training Dataset: Measure for Training Dataset	Before Applying SMOTE on Training Dataset: Measure for Test Dataset	After Applying SMOTE on Training Dataset: Measure for Training Dataset	After Applying SMOTE on Training Dataset: Measure for Test Dataset
1	LogisticReg-Base	'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': None, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False	0.659590	0.60	0.87908	0.663122
2	KNeighborsClassifier	'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'	0.833949	0.750304		0.724662
3	Decision Tree-Gini	'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': 10, 'splitter': 'best'	1.0	0.721342	1.0	0.706410
4	Gaussian NB	'priors': None, 'var_smoothing': 1e-09	0.6522402	0.653885	0.69442	0.644041
5	Random Forest	'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 10, 'verbose': 0, 'warm_start': False	1.0	0.805588	0.69442	0.803859
6	Bagging Classifier-dt	'base_estimator': 'deprecated', 'bootstrap': True, 'bootstrap_features': False,	0.98560	0.789617	0.99094	0.796810

		'estimator__ccp_alpha': 0.0, 'estimator__class_weight': None, 'estimator__criterion': 'gini', 'estimator__max_depth': None, 'estimator__max_features': None, 'estimator__max_leaf_nodes': None, 'estimator__min_impurity_decrease': 0.0, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 2, 'estimator__min_weight_fraction_leaf': 0.0, 'estimator__random_state': 10, 'estimator__splitter': 'best', 'estimator': DecisionTreeClassifier(random_state=10), 'max_features': 1.0, 'max_samples': 1.0, 'n_estimators': 10, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False				
7	Adaboost-dt	'algorithm': 'SAMME.R', 'base_estimator': 'deprecated', 'estimator__ccp_alpha': 0.0, 'estimator__class_weight': None, 'estimator__criterion': 'gini', 'estimator__max_depth': None, 'estimator__max_features': None, 'estimator__max_leaf_nodes': None, 'estimator__min_impurity_decrease': 0.0, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 2, 'estimator__min_weight_fraction_leaf': 0.0, 'estimator__random_state': 10, 'estimator__splitter': 'best', 'estimator': DecisionTreeClassifier(random_state=10), 'learning_rate': 1.0, 'n_estimators': 50, 'random_state': 10	1.0	0.721502	1.0	0.703424
8	XGBoost	'objective': 'multi:softprob', 'base_score': None, 'booster': None, 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': None, 'device': None, 'early_stopping_rounds': None, 'enable_categorical': False, 'eval_metric': None, 'feature_types': None, 'gamma': None, 'grow_policy': None, 'importance_type': None, 'interaction_constraints': None, 'learning_rate': None, 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None,	0.83264	0.768676	0.84743	0.760933

		'max_depth': None, 'max_leaves': None, 'min_child_weight': None, 'missing': nan, 'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': None, 'n_jobs': None, 'num_parallel_tree': None, 'random_state': 10, 'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None, 'subsample': None, 'tree_method': None, 'validate_parameters': None, 'verbosity': None				
9	XGBoost_tuned	'objective': 'multi:softprob', 'base_score': None, 'booster': None, 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': 1.0, 'device': None, 'early_stopping_rounds': None, 'enable_categorical': False, 'eval_metric': None, 'feature_types': None, 'gamma': None, 'grow_policy': None, 'importance_type': None, 'interaction_constraints': None, 'learning_rate': 0.1, 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 5, 'max_leaves': None, 'min_child_weight': None, 'missing': nan, 'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': 200, 'n_jobs': None, 'num_parallel_tree': None, 'random_state': 10, 'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None, 'subsample': 0.8, 'tree_method': None, 'validate_parameters': None, 'verbosity': None	0.769765	0.744377	0.790979	0.75866

Project Justification

Complexity involved:

Building a machine learning model to classify credit scores involves several complexities due to the nature of the task and the data involved. Access to accurate and comprehensive credit data is crucial. However, acquiring and handling sensitive financial information requires compliance with privacy regulations. The distribution of credit scores may be imbalanced, with fewer instances of low or high scores. This can affect the model's ability to generalize well.

Evaluation:

The dataset encompasses 27 features, with Outstanding Debt, Credit Mix Standard, and Delay from Due Date identified as pivotal factors influencing the target variable. Using visualization tools, irrelevant features lacking impact were systematically excluded from consideration.

Various classification models, such as Logistic Regression, KNeighbors Classifier, Decision Tree, Gaussian Naive Bayes, Random Forest, Bagging Classifier, Adaboost, and XGBoost, underwent experimentation. Among these, the XGBoost algorithm emerged as the most effective, showcasing superior performance across the eight classification models on both training and testing data. Accuracy levels were diligently observed for each model. To enhance model performance, SMOTE was employed to address data imbalance, and hyperparameter tuning was conducted using GridSearch CV. However, as the base model demonstrated superior results, we proceeded with the XGBoost algorithm without tuning and SMOTE.

Given the dataset's multiclassification nature, the weighted F1 score was chosen as the pivotal metric for evaluating classification model accuracy and efficacy. The optimal model identified utilizes the XGBoost algorithm, achieving a weighted F1 score of 0.76 on the test data and 0.83 on the training data. This underscores a proficient model capable of accurate and effective classification.

Project Outcome:

Building a machine learning model with a credit score classification feature based on an individual's credit-related data and the global finance company's dataset. We have tried with 8 different algorithms for obtaining the best model. Also SMOTE as well as hyperparameter tuning was done. This resulted in identifying XGBoost, giving weighted F1 score 0.76 in test data and 0.83 in training data. The categorization models classify each person's credit score with a good accuracy. The features having more influence on target where identified as Outstanding Debt, Credit Mix Standard, and Delay from Due Date.

Implications

The finance company's decision to establish an intelligent method for categorizing individuals into credit score groups is a foresighted and strategic move. We can modernize operations, improve customer experiences, and even alter the way we assess and provide financial services to clients by adopting data science and artificial intelligence. This effort has the potential to deliver enormous benefits and position the organization as a leader in the emerging financial landscape with careful planning and a commitment to ethical data practices. In summary, the implications of using machine learning for credit score classification underscore the importance of ethical considerations, fairness, transparency, and ongoing vigilance in model monitoring and management. Striking the right balance between leveraging advanced technologies for improved credit assessment and safeguarding the rights and privacy of individuals is crucial for responsible and sustainable use in the financial industry.

Limitations

While machine learning can significantly enhance credit score classification, there are several limitations and challenges associated with its application in this domain:

1. **Interpretability:**
    - Many machine learning models, especially complex ones like ensemble methods or deep learning, are often considered "black-box" models. Interpreting the rationale behind credit decisions becomes challenging, which can be a concern for regulatory compliance and transparency.
2. **Data Privacy and Bias:**

    Machine learning models are sensitive to biases present in historical data. If historical data includes biased information, the model may perpetuate and even exacerbate these biases, leading to unfair or discriminatory outcomes. Ensuring fairness and avoiding biased decisions is crucial, especially in the financial sector.

3. **Dynamic Nature of Credit Markets:**

   Credit markets are dynamic, influenced by economic conditions, regulatory changes, and other external factors. Machine learning models trained on historical data may not adapt well to changes in the credit landscape, potentially leading to reduced predictive performance.

4. **Imbalanced Datasets:**

   Imbalanced datasets, where one class (e.g., good credit) significantly outnumbers the other (e.g., bad credit), can pose challenges. Standard machine learning algorithms might be biased towards the majority class, impacting the model's ability to accurately predict the minority class.

5. **Data Quality and Missing Values:**

   The quality of credit data can vary, and missing or inaccurate information can affect the model's performance. Ensuring data completeness and accuracy is crucial for reliable credit score predictions.

6. **Overfitting:**

   Overfitting occurs when a model learns the training data too well, capturing noise and idiosyncrasies that do not generalize to new, unseen data. Regularization techniques and proper model evaluation can help mitigate this issue.

Despite these challenges, ongoing research and advancements in interpretable machine learning, fairness-aware algorithms, and ethical AI practices aim to address some of these limitations and improve the reliability and fairness of credit score classification models. It is crucial for organizations to carefully consider these limitations and deploy machine learning models responsibly in the financial sector.

Closing Reflections

In closing reflections on Credit Score classification using machine learning, it is evident that while these advanced techniques hold immense potential to revolutionize credit assessment, careful consideration must be given to the ethical, regulatory, and societal implications. The responsible use of machine learning in credit scoring demands a delicate balance between harnessing predictive power and ensuring fairness, transparency, and privacy. As financial institutions increasingly adopt these technologies, ongoing efforts to address biases, enhance interpretability, and comply with evolving regulations become paramount. Moreover, fostering consumer trust through clear communication and education about the use of machine learning models in credit decisions is essential. In this dynamic landscape, continuous monitoring, model updating, and collaboration between industry stakeholders, regulators, and technologists are crucial to navigate the evolving challenges and maximize the benefits of machine learning in shaping the future of credit scoring.