Credit Score Estimation

Project Goal:

The goal of this project is to build a machine learning model that can predict the credit worthiness of customers. The model can be used by banks and other financial institutions to assess the risk of lending money to customers. The project will involve data preprocessing, test for multicollinearity, model training, and evaluation. By leveraging historical credit data, the project will provide insights into creditworthiness and assist in risk assessment for lenders.

Libraries used:

sklearn (Scikit-learn): It is a popular machine learning library in Python that provides a wide range of tools for data preprocessing, feature selection, model training, evaluation, and more. It offers various algorithms and utilities for classification, regression, clustering, dimensionality reduction, and model selection.

- preprocessing (from sklearn): This module within scikit-learn provides functions for data preprocessing and transformation. It includes methods for scaling, normalization, encoding categorical variables, handling missing values, and more.
- LogisticRegression (from sklearn.linear_model): This module implements the logistic regression algorithm, which is used for binary classification problems.
- KNeighborsClassifier (from sklearn.neighbors): This module implements the K-Nearest Neighbors (KNN) algorithm for classification. It assigns a class label to a data point based on the majority class of its K nearest neighbors in the feature space.
- DecisionTreeClassifier (from sklearn.tree): This module provides the Decision Tree algorithm for classification tasks. It builds a tree-like model of decisions and their possible consequences based on the training data.
- RandomForestClassifier (from sklearn.ensemble): This module implements the Random Forest algorithm, which is an ensemble method that combines multiple decision trees to make

predictions. It improves generalization and reduces overfitting compared to a single decision tree.

- metrics (from sklearn): This module provides various metrics to evaluate the performance of machine learning models. It includes functions for calculating accuracy, precision, recall, F1-score, and other evaluation metrics.
    i. accuracy_score (from sklearn.metrics): It is a function specifically for calculating the accuracy of classification models. It compares the predicted labels with the true labels and computes the accuracy as the ratio of correct predictions to the total number of predictions.
- train_test_split (from sklearn.model_selection): This function is used to split the dataset into training and testing subsets. It randomly shuffles the data and divides it into specified proportions, allowing for model training on the training set and evaluation on the testing set.

Getting familiar with the Dataset:

The dataset is downloaded from Kaggle. The source has not been verified by the provider. But the dataset has been viewed more than 1,11,0000 times and downloaded over 14,000 times by fellow Kaggle users. The dataset consisted of two separate CSVs, one labelled as train and the other as test. Upon close observation, the dataset contained a few inconsistencies and impurities which will be discussed later in the study.

The data has been split for January to August under training set and September to December for test set. This is a common way to split credit score estimation models. The idea is to use the data from the first 8 months of the year to train the model, and then use the data from the last 4 months of the year to test the model. This ensures that the model is not overfitting to the training data, and that it can generalize to new data.

Understanding the Dataset:

	count	mean	std	min	25%	50%	75%	max
Monthly_Inhand_Salary	84998.0	4194.170850	3183.686167	303.645417	1625.568229	3093.745000	5957.448333	15204.633333
Num_Bank_Accounts	100000.0	17.091280	117.404834	-1.000000	3.000000	6.000000	7.000000	1798.000000
Num_Credit_Card	100000.0	22.474430	129.057410	0.000000	4.000000	5.000000	7.000000	1499.000000
Interest_Rate	100000.0	72.466040	466.422621	1.000000	8.000000	13.000000	20.000000	5797.000000
Delay_from_due_date	100000.0	21.068780	14.860104	-5.000000	10.000000	18.000000	28.000000	67.000000
Num_Credit_Inquiries	98035.0	27.754251	193.177339	0.000000	3.000000	6.000000	9.000000	2597.000000
Credit_Utilization_Ratio	100000.0	32.285173	5.116875	20.000000	28.052567	32.305784	36.496663	50.000000
Total_EMI_per_month	100000.0	1403.118217	8306.041270	0.000000	30.306660	69.249473	161.224249	82331.000000

None

	count	mean	std	min	25%	50%	75%	max
Monthly_Inhand_Salary	42502.0	4182.004291	3174.109304	303.645417	1625.188333	3086.305000	5934.189094	15204.633333
Num_Bank_Accounts	50000.0	16.838260	116.396848	-1.000000	3.000000	6.000000	7.000000	1798.000000
Num_Credit_Card	50000.0	22.921480	129.314804	0.000000	4.000000	5.000000	7.000000	1499.000000
Interest_Rate	50000.0	68.772640	451.602363	1.000000	8.000000	13.000000	20.000000	5799.000000
Delay_from_due_date	50000.0	21.052640	14.860397	-5.000000	10.000000	18.000000	28.000000	67.000000
Num_Credit_Inquiries	48965.0	30.080200	196.984121	0.000000	4.000000	7.000000	10.000000	2593.000000
Credit_Utilization_Ratio	50000.0	32.279581	5.106238	20.509652	28.061040	32.280390	36.468591	48.540663
Total_EMI_per_month	50000.0	1491.304305	8595.647887	0.000000	32.222388	74.733349	176.157491	82398.000000

Table

Here we will have a look at the summary of the training and testing data. The first table is for the training dataset and the second table is for test dataset. The resulting tables have similar values for these statistics, it could mean that they are very similar. As the datasets were drawn from the same population, or because they were pre-processed in a similar way.

I combined the train and test dataset for ease of data cleaning.

Data Cleaning

In the first step of the cleaning process, redundant values like ['', 'nan', '!@9#%8', '#F%$D@*&8'] from the dataset have been replaced with np.NaN. Features that were not relevant were dropped like ['ID' ,'Name', 'Type_of_Loan', 'SSN', 'Occupation'].

Changing the data type:

Let's look at the data types for all columns-

```
 Data columns (total 24 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Customer_ID               150000 non-null  object
 1   Month                     150000 non-null  object
 2   Age                       150000 non-null  object
 3   Occupation                139500 non-null  object
 4   Annual_Income             150000 non-null  object
 5   Monthly_Inhand_Salary     127500 non-null  float64
 6   Num_Bank_Accounts         150000 non-null  int64
 7   Num_Credit_Card           150000 non-null  int64
 8   Interest_Rate             150000 non-null  int64
 9   Num_of_Loan               150000 non-null  object
 10  Delay_from_due_date       150000 non-null  int64
 11  Num_of_Delayed_Payment    139500 non-null  object
 12  Changed_Credit_Limit      146850 non-null  object
 13  Num_Credit_Inquiries      147000 non-null  float64
 14  Credit_Mix                120000 non-null  object
 15  Outstanding_Debt          150000 non-null  object
 16  Credit_Utilization_Ratio  150000 non-null  float64
 17  Credit_History_Age        136500 non-null  object
 18  Payment_of_Min_Amount     150000 non-null  object
 19  Total_EMI_per_month       150000 non-null  float64
 20  Amount_invested_monthly   143250 non-null  object
 21  Payment_Behaviour         138600 non-null  object
 22  Monthly_Balance           148238 non-null  object
 23  Credit_Score              100000 non-null  object
```

Some columns are incorrectly entered as object type. I fixed this by converting the 'Age' column to integer, the 'Annual_Income', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit', 'Outstanding_Debt', 'Amount_invested_monthly', and 'Monthly_Balance' columns to float.

- For better readability and understanding of the unique customer ID, I changed the ID number.
  `df['Customer_ID'] = df.Customer_ID.apply(lambda x: int(x[4:], 16))`
  This line of code converts the hexadecimal value in the Customer_ID column to an integer. The apply() method applies the lambda function to each cell in the Customer_ID column. The lambda function takes a hexadecimal value as input and returns the integer value of the hexadecimal value, starting from the 4th character

- `df['Month'] = pd.to_datetime(df.Month, format='%B').dt.month`

This line of code converts the Month column to a datetime object. The to_datetime() method takes a string as input and returns a datetime object. The format parameter specifies the format of the string. In this case, the format is %B, which is the standard format for months in the English language. The dt.month attribute of the datetime object returns the month number of the datetime object.

- I converted credit history age from years and month to number of months. I changed credit mix from bad, standard and good to 0,1, and 2.
- 'Payment of Min Amount' contained 3 categorical values 'No, 'yes' and 'NM'. The dataset or data source on Kaggle did not specify what 'NM' is, so I converted it to the modal value of the grouped 'Customer_ID'.

Exploratory data analysis

Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. EDA is a critical first step in the data analysis process, as it helps to understand the data and identify any potential problems.

Handling Missing Values

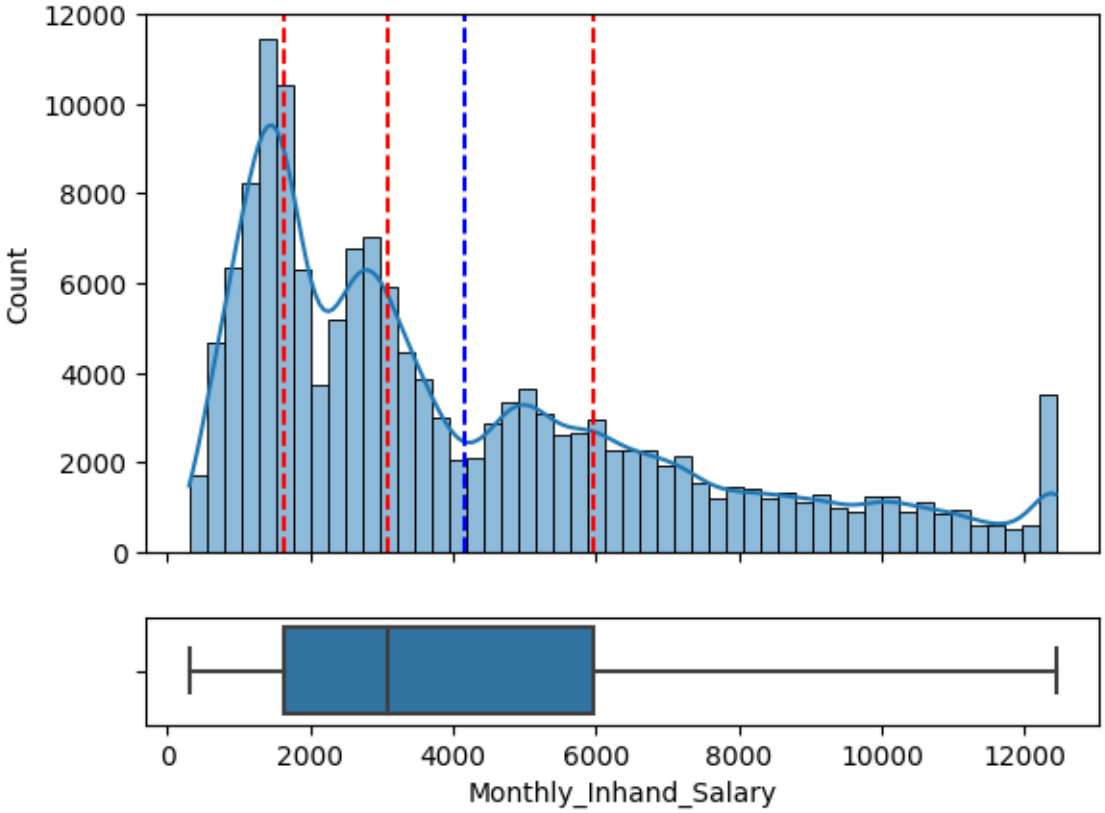
The data consists of a lot of missing values. It was challenging to impute the data with the mean or modal value of the dataset, as there were 12 monthly observations for each customer ID. This meant that the central tendency of the entire dataset could very different from an individual's central value. Hence, I first grouped the data by Customer_ID column, found out the modal value and replaced the missing value with it. This ensured that within the Customer's details too, the data remained consistent.

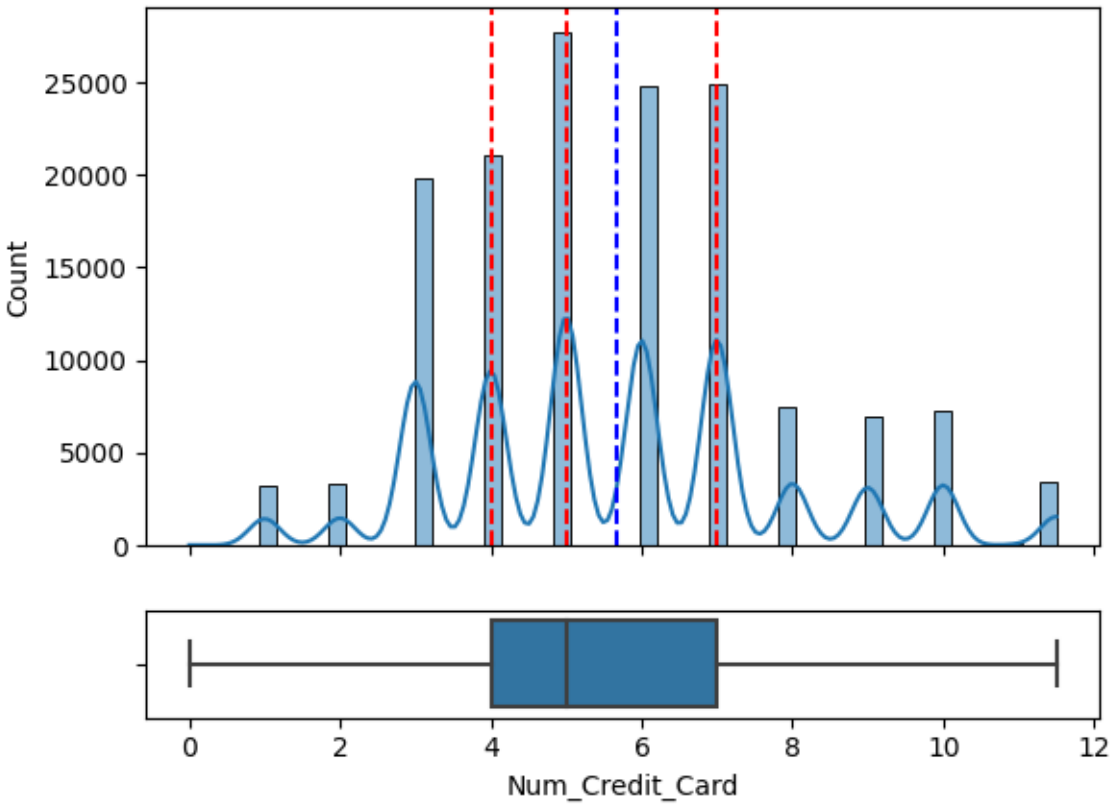
Duplicate values

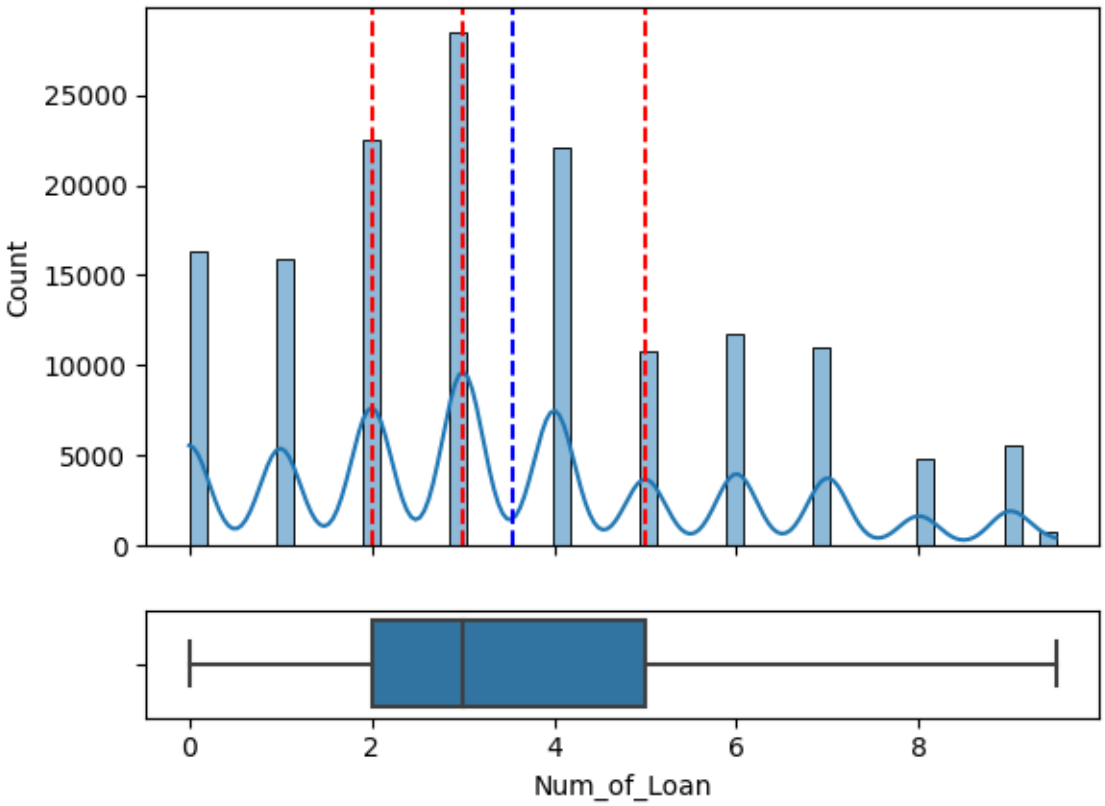
There were no duplicates found.

Correcting outliers

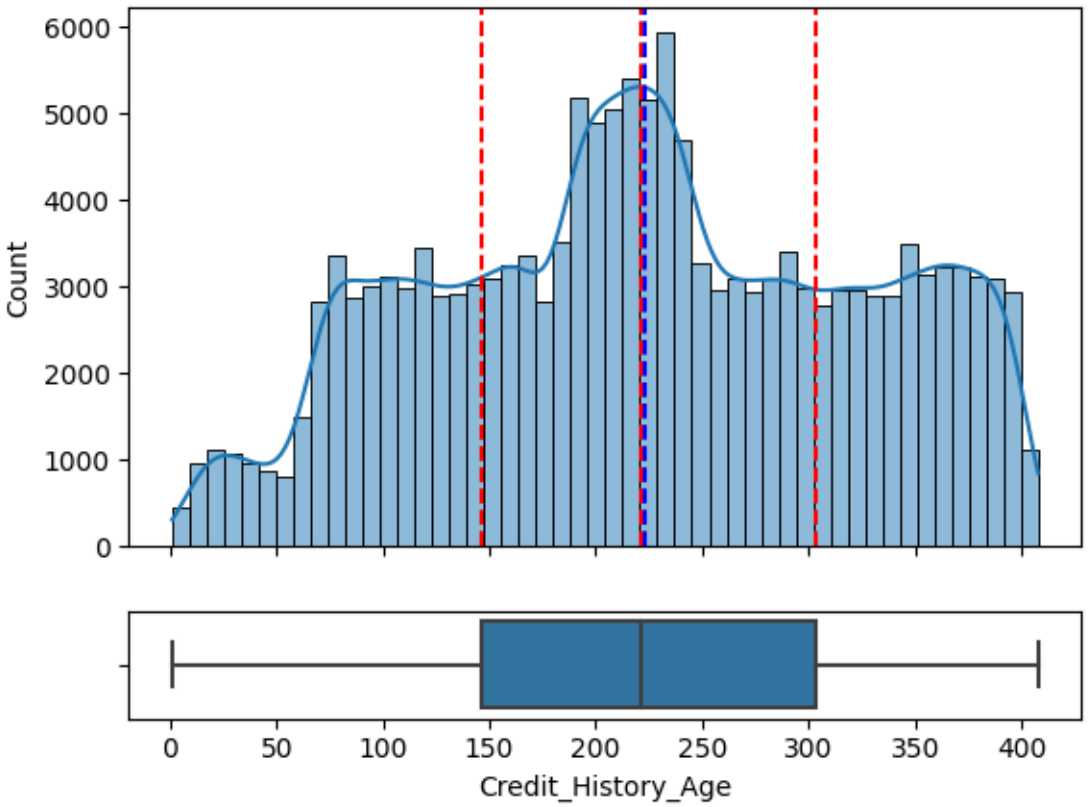
Since there were several numerical values like Age, Annual_Income, Monthly_Inhand_Salary, Num_Bank_Accounts, Num_Credit_Card, Interest_Rate, Num_of_Loan, Delay_from_due_date, Num_of_Delayed_Payment, Changed_Credit_Limit, Num_Credit_Inquiries, Credit_Mix, Outstanding_Debt, Credit_Utilization_Ratio, Credit_History_Age, Total_EMI_per_month, Amount_invested_monthly, Monthly_Balance, I defined a function to test for outliers. There were thousands of outliers. So, the data operation followed by setting aside the object values and winsorizing all the numeric columns. Winsorization is a data transformation technique used to limit the impact of extreme values (outliers) on statistical analyses. It involves replacing extreme values (outliers) in a dataset with less extreme values, specifically the minimum and maximum non-outlier values within a certain range.

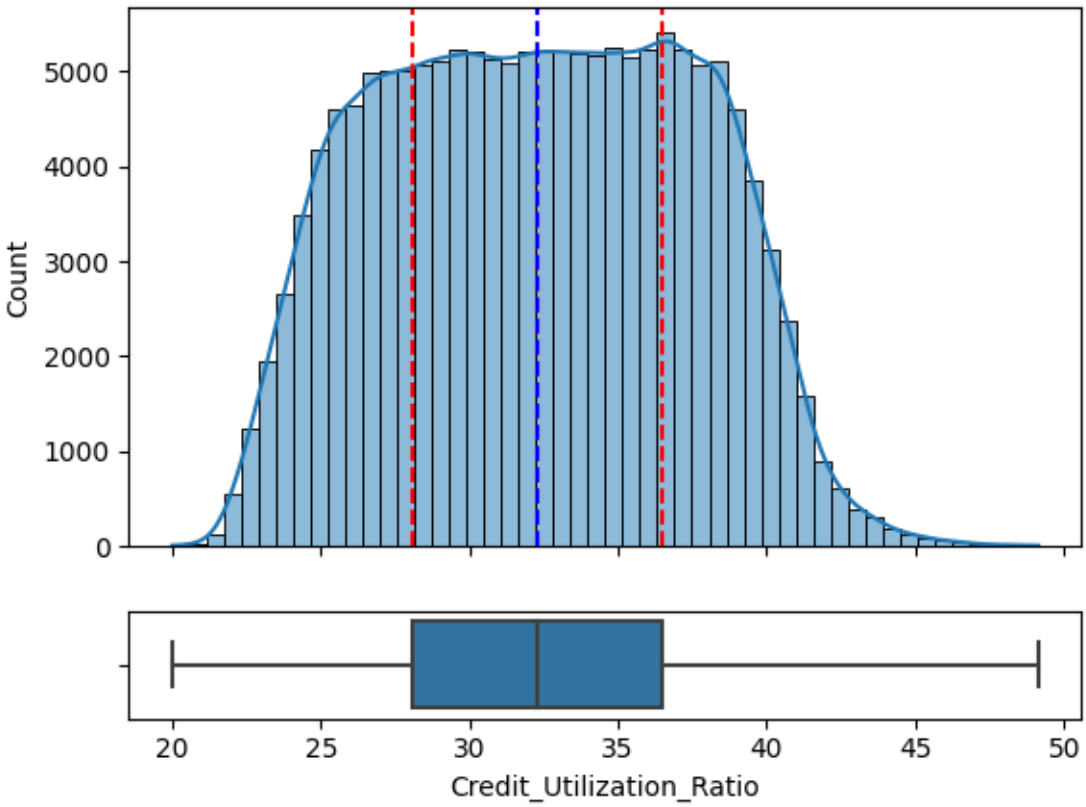

12000
10000
8000
6000
4000
2000
0
Count
0
2000
4000
6000
8000
10000
12000
Monthly_Inhand_Salary


25000
20000
15000
10000
5000
0
Count
0
2
4
6
8
10
12
Num_Credit_Card

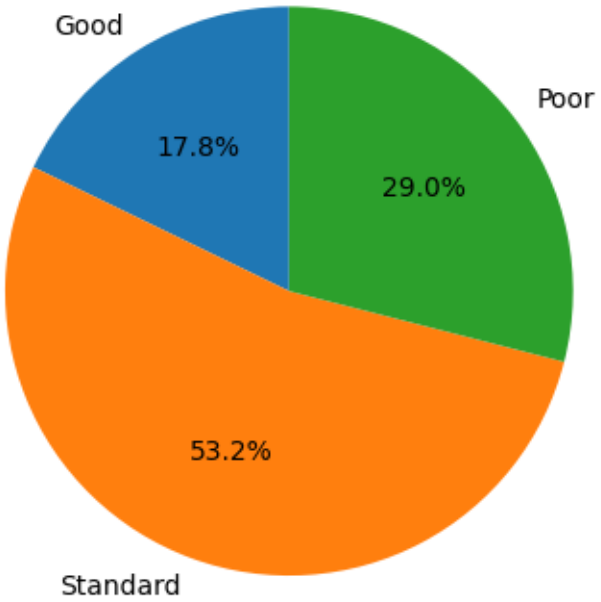

Count
25000
20000
15000
10000
5000
0
0
2
4
6
8
Num_of_Loan

\

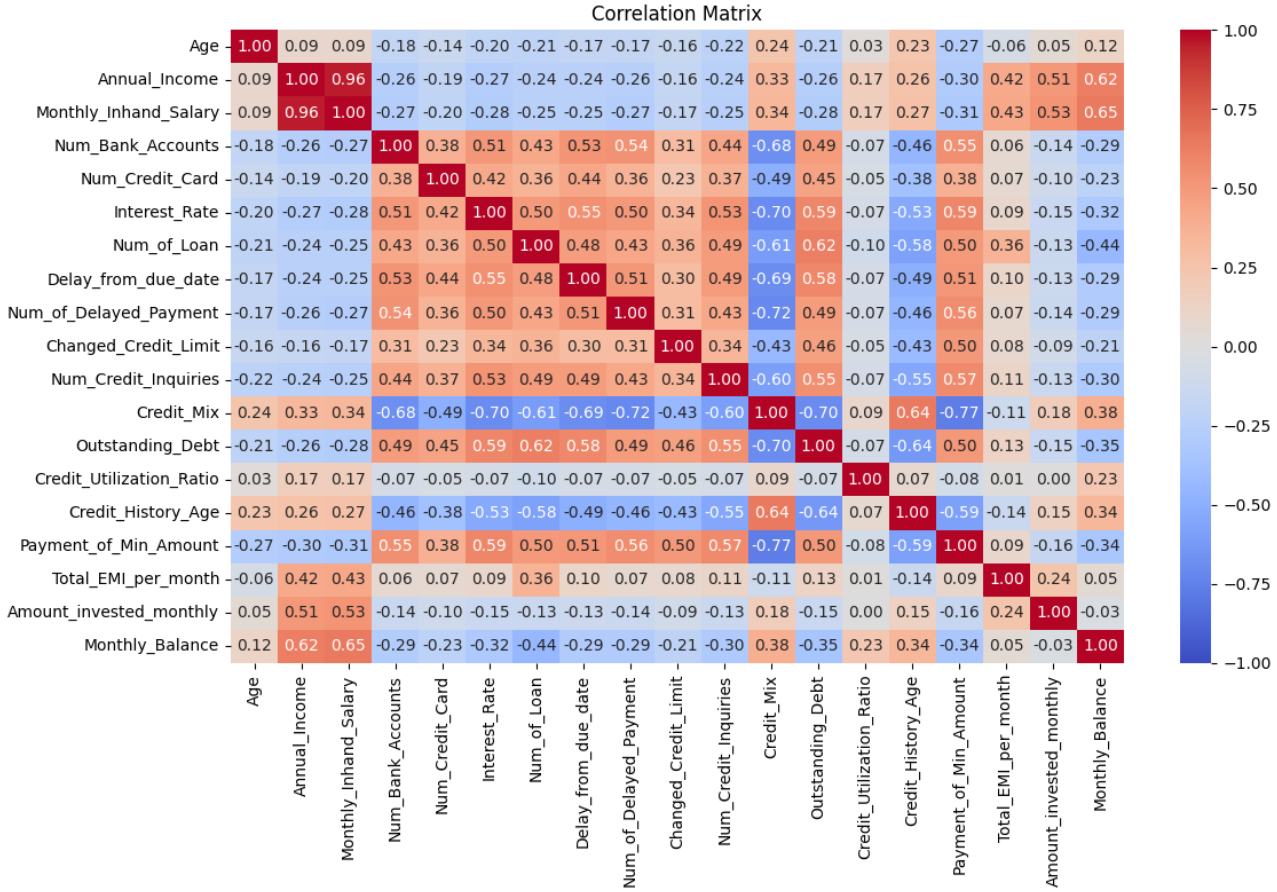

Count
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250
300
350
400
Credit_History_Age



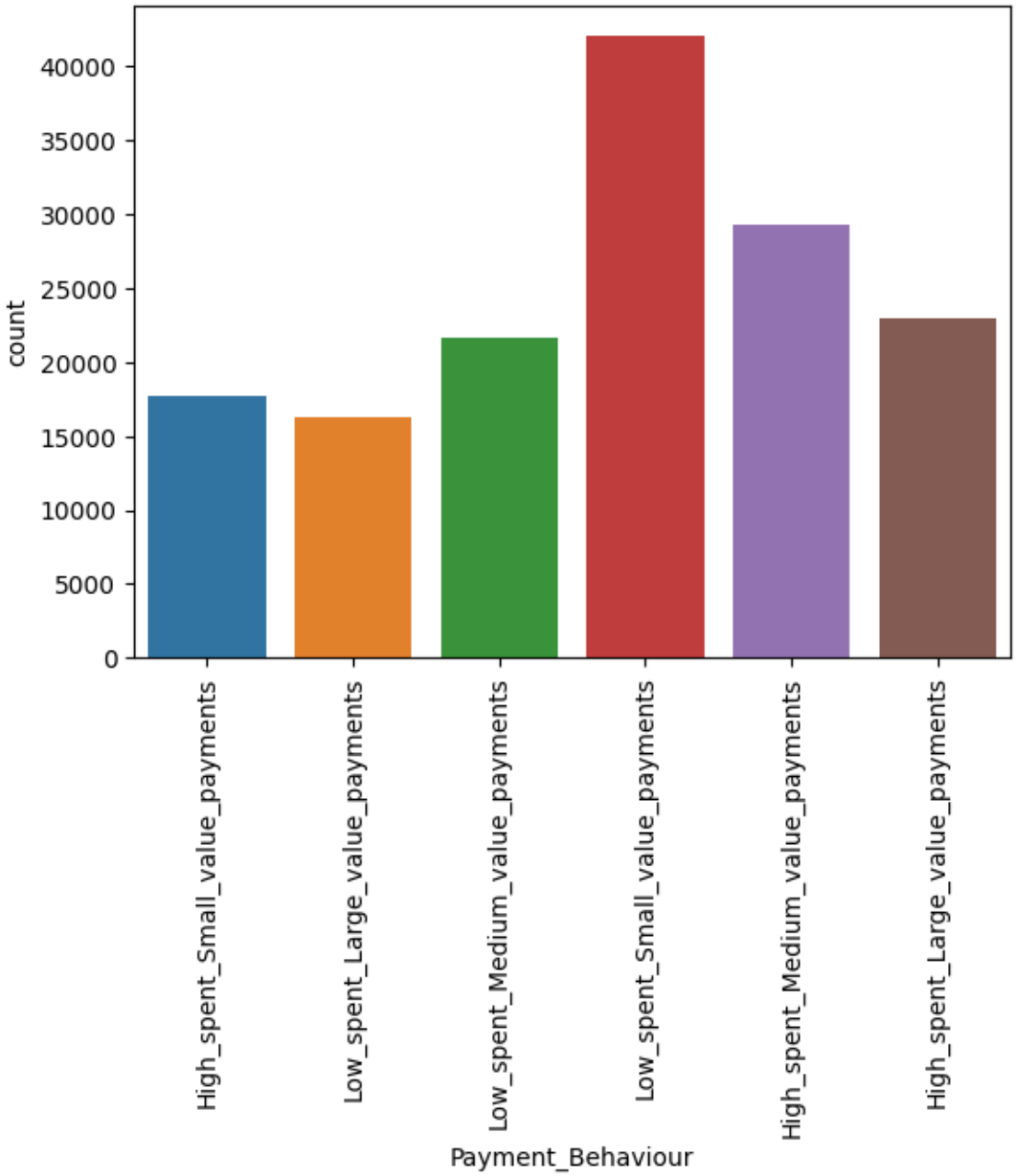
customers with Good, Standard and Poor Credit Score

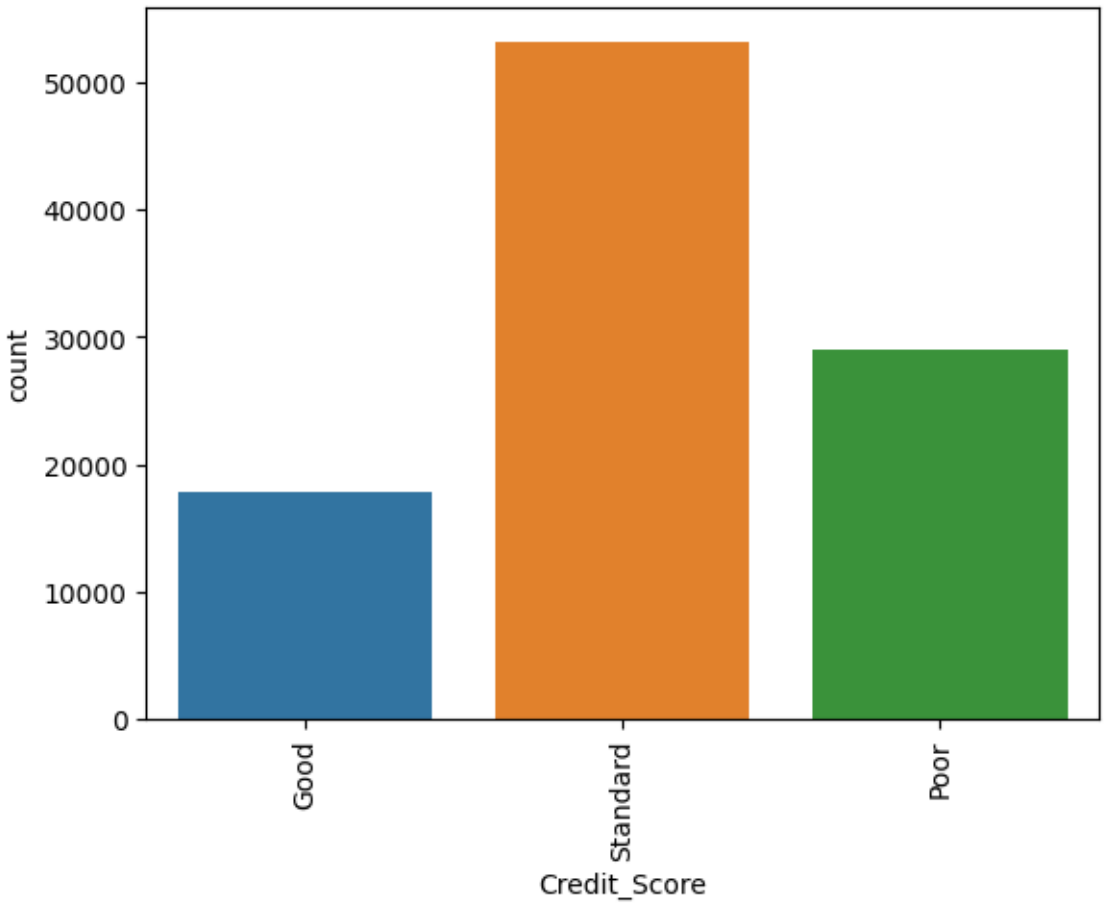


Bivariate Analysis for numerical data

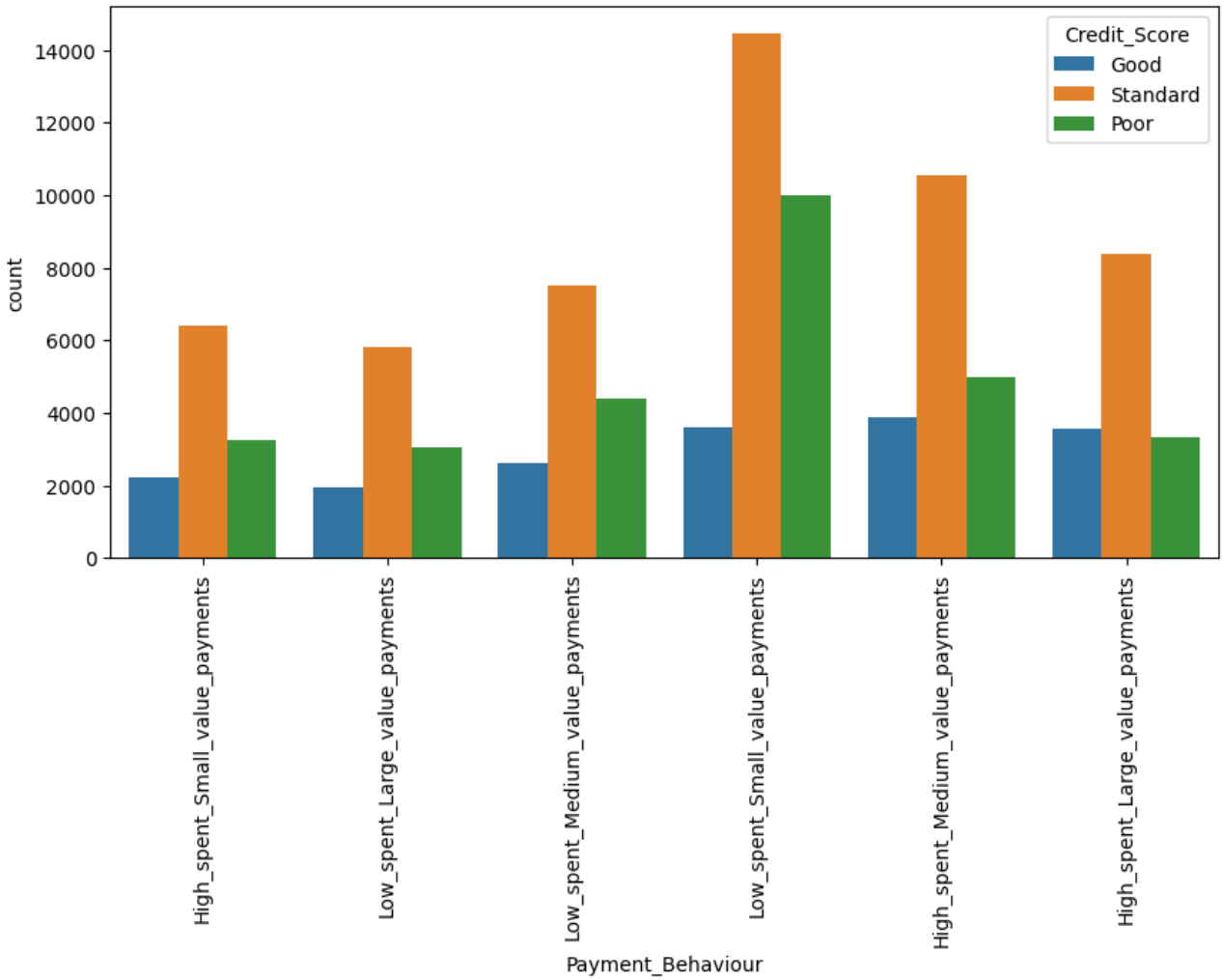


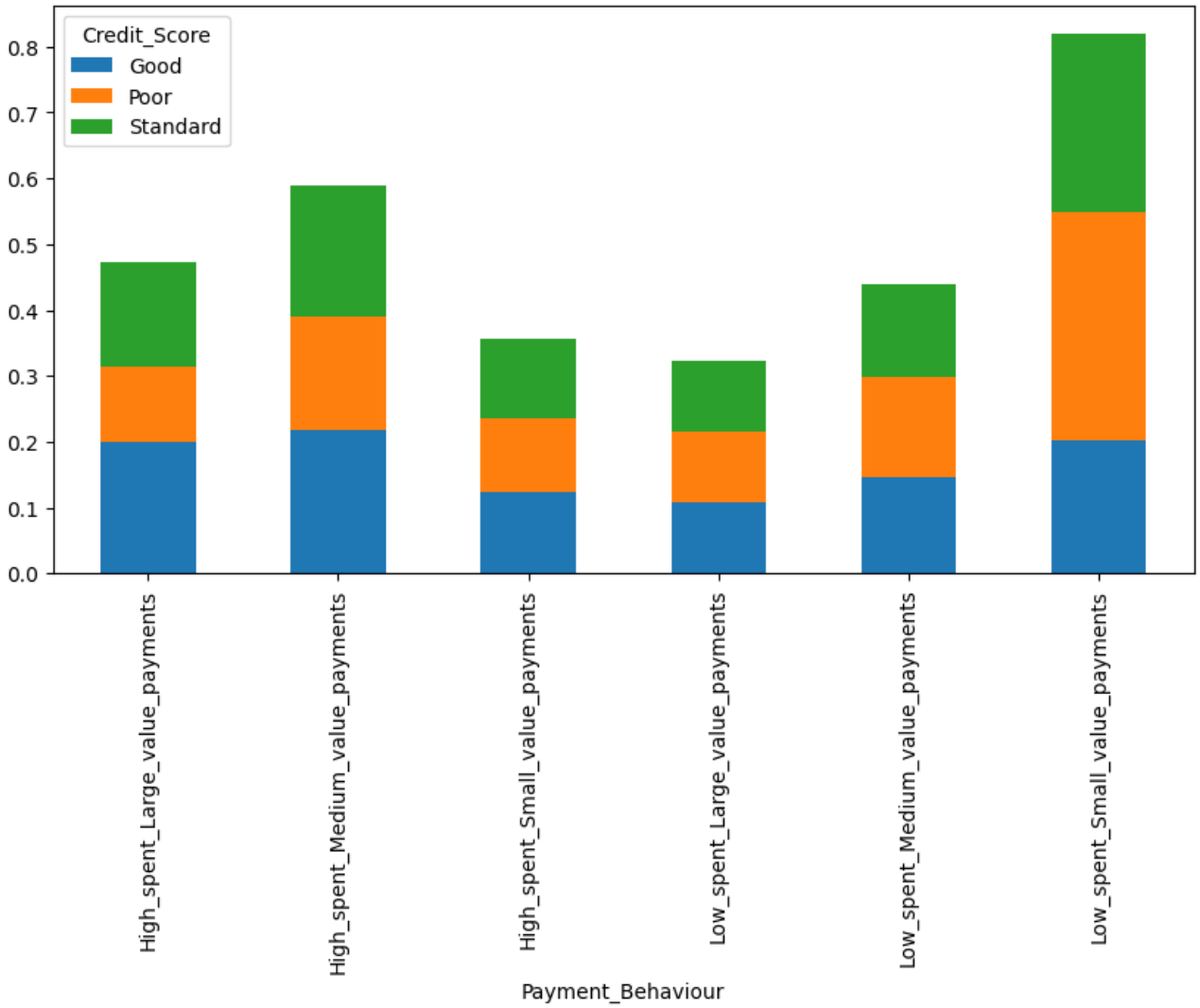
Univariate Analysis of Categorical data




50000
40000
30000
20000
10000
0
count
Good
Standard
Poor
Credit_Score

Bivariate analysis for categorical



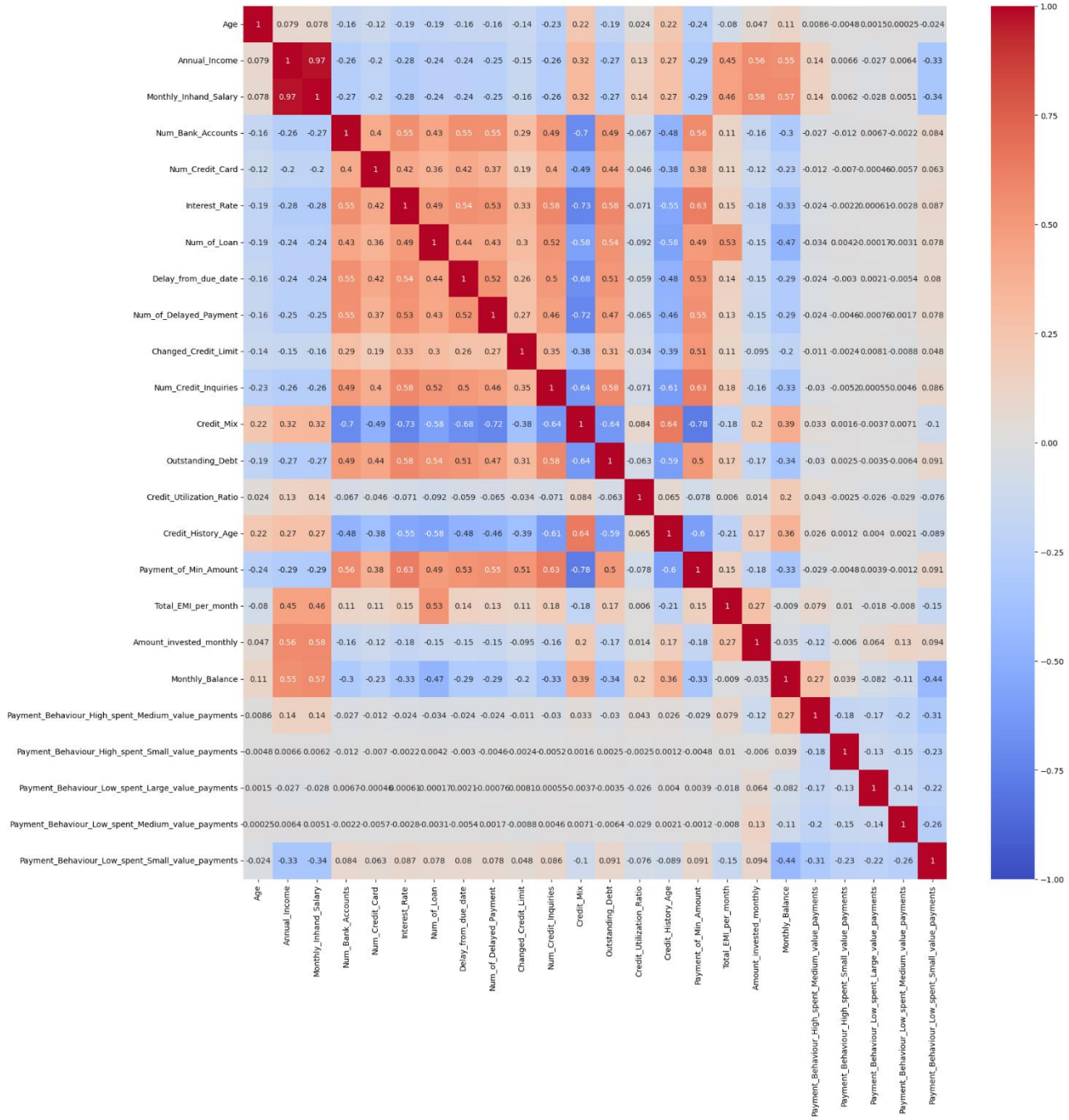

Credit_Score
Good
Poor
Standard
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
High_spent_Large_value_payments
High_spent_Medium_value_payments
High_spent_Small_value_payments
Low_spent_Large_value_payments
Low_spent_Medium_value_payments
Low_spent_Small_value_payments
Payment_Behaviour

Feature Engineering

Encoding Categorical Features

- “Payment_of_Min_Amount” column represents whether a person has made only the minimum payment amount on their credit card bill or loan instalment. It can provide insights into their ability to manage their debts and whether they are making timely and sufficient payments towards their outstanding balances. It was encoded ordinally with 0 for ‘No’ and 1 for ‘Yes’.
- “Credit_Mix” represents the classification of the mix of credits. It typically describes the types of credit accounts a person has, such as loans, credit cards, mortgages, etc. The exact definitions are not given but "Bad" may indicate a poor or unfavorable mix of credit accounts. This may imply having a higher proportion of high-risk or low-quality credit accounts in the overall credit mix. "Standard" suggests a balanced combination of different types of credit accounts without any significant negative or positive implications. "Good” suggests having a diverse range of credit accounts, which can contribute positively to one's credit profile. This was mapped in the following way - "Bad":0, "Standard":1, "Good":2 as there is some ordinal relationship present.
- “Payment_Behaviour” has 6 unique categorical values like - 'High_spent_Small_value_payments', 'Low_spent_Large_value_payments', 'Low_spent_Medium_value_payments', 'Low_spent_Small_value_payments', 'High_spent_Medium_value_payments', and 'High_spent_Large_value_payments'.
- Similarly, our target variable “Credit_Score” would be ordinally encoded "Poor":0, "Standard":1, "Good":2

Standard Scaling


Age
Annual_Income
Monthly_Inhand_Salary
Num_Bank_Accounts
Num_Credit_Card
Interest_Rate
Num_of_Loan
Delay_from_due_date
Num_of_Delayed_Payment
Changed_Credit_Limit
Num_Credit_Inquiries
Credit_Mix
Outstanding_Debt
Credit_Utilization_Ratio
Credit_History_Age
Payment_of_Min_Amount
Total_EMI_per_month
Amount_invested_monthly
Monthly_Balance
Payment_Behaviour_High_spent_Medium_value_payments
Payment_Behaviour_High_spent_Small_value_payments
Payment_Behaviour_Low_spent_Large_value_payments
Payment_Behaviour_Low_spent_Medium_value_payments
Payment_Behaviour_Low_spent_Small_value_payments
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

	Random Forest 25	Random Forest 50	KNN 15	KNN 6	KNN 2	KNN 1	Decision_tree	Random Forest 75	Random Forest 100	Logistic Reggression
Accuracy	89.57	89.94	72.68	74.98	80.40	86.38	83.70	90.06	90.12	65.16
Precision	89.57	89.95	72.66	74.98	80.98	86.39	83.76	90.06	90.12	90.12

Feature Importance