DATA DICTIONARY

ID: An identifier for each record in the dataset.
Customer_ID: Identifier for individual customers.
Month: The month associated with the data entry.
Name: Customer's name.
Age: Customer's age.
SSN: Social Security Number or some other form of identification.
Occupation: Customer's occupation or job title.
Annual_Income: Customer's annual income.
Monthly_Inhand_Salary: The amount of money the customer receives as salary on a monthly basis.
Num_Bank_Accounts: Number of bank accounts the customer has.
Num_Credit_Card: Number of credit cards the customer possesses.
Interest_Rate: The interest rate associated with some financial aspect (e.g., loans or credit cards).
Num_of_Loan: Number of loans the customer has.
Type_of_Loan: The type of loan(s) the customer has (e.g., mortgage, personal loan, etc.).
Delay_from_due_date: Delay in payments from the due date.
Num_of_Delayed_Payment: Number of delayed payments.
Changed_Credit_Limit: Indicates if the customer has changed their credit limit.
Num_Credit_Inquiries: Number of credit inquiries made by the customer.
Credit_Mix: The mix of different types of credit accounts (e.g., credit cards, loans).
Outstanding_Debt: The amount of outstanding debt.
Credit_Utilization_Ratio: The ratio of credit used to the total credit limit.
Credit_History_Age: The age of the customer's credit history.
Payment_of_Min_Amount: Payment behavior regarding minimum required payments.
Total_EMI_per_month: Total Equated Monthly Installment (EMI) payments made by the customer.
Amount_invested_monthly: The amount the customer invests on a monthly basis.
Payment_Behaviour: Behavior related to payments, possibly indicating patterns or trends.
Monthly_Balance: The customer's monthly balance in their financial accounts.
Credit_Score: The credit score associated with the customer's creditworthiness.

The Target Variable is Credit_Score which has values of {Good, Poor, Standard}.

```
1 data['Credit_Score'].value_counts()
```

```
Standard    53174
Poor        28998
Good        17828
Name: Credit_Score, dtype: int64
```

The objective of this study is to classify customers based on their information about their demographics and financial information into these 3 categories and predict if the customer is going to have good or bad credit score.

DATA TYPES

There are different data types in our raw data. As we know some of the columns corresponds to numeric values but due to presence of some missing values and discrepancy in data they are stored as string. Such as Age, Balance and son on. The below table demonstrates the initial data types in raw dataset. In the following steps we performed some data transformations in order to address this issue and convert numeric columns to numeric values.

```
1 data_types = data.dtypes
2 data_types
```

```
ID                          object
Customer_ID                 object
Month                       object
Name                        object
Age                         object
SSN                         object
Occupation                  object
Annual_Income               object
Monthly_Inhand_Salary      float64
Num_Bank_Accounts            int64
Num_Credit_Card              int64
Interest_Rate                int64
Num_of_Loan                 object
Type_of_Loan                object
Delay_from_due_date          int64
Num_of_Delayed_Payment      object
Changed_Credit_Limit       float64
Num_Credit_Inquiries       float64
Credit_Mix                  object
Outstanding_Debt            object
Credit_Utilization_Ratio   float64
Credit_History_Age          object
Payment_of_Min_Amount       object
Total_EMI_per_month        float64
Amount_invested_monthly     object
Payment_Behaviour           object
Monthly_Balance             object
Credit_Score                object
dtype: object
```

DESCRIPTIVE OF DATA

Some of the features that are initially stored as Object variables, need to be converted to Numeric as they corresponds to Numeric Values as they corresponds to actual numeric values.

For example for the below features we converted them into Numeric Values:

```
features_to_be_num=['Outstanding_Debt','Annual_Income','Amount_invested_monthly','Monthly_Balance']
for feature in features_to_be_num:
    data[feature] = data[feature].str.extract(r'(\d+\.\d+)').astype(float)
```

```
data['Num_of_Delayed_Payment'] = data['Num_of_Delayed_Payment'].str.extract('(\d+)').astype(float)
```

After Converting them to numeric values, we see below the description of columns using describe() command in python:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num
count	100000.000000	1.000000e+05	84998.000000	100000.000000	
mean	119.509700	1.764157e+05	4194.170850	17.091280	
std	684.757313	1.429618e+06	3183.686167	117.404834	
min	14.000000	7.005930e+03	303.645417	-1.000000	
25%	25.000000	1.945750e+04	1625.568229	3.000000	
50%	34.000000	3.757861e+04	3093.745000	6.000000	
75%	42.000000	7.279092e+04	5957.448333	7.000000	
max	8698.000000	2.419806e+07	15204.633333	1798.000000	

Changed_Credit_Limit	Num_Credit_Inquiries	Outstanding_Debt	Credit_Utilization_Ratio
97909.000000	98035.000000	100000.000000	100000.000000
10.389025	27.754251	1426.220376	32.285173
6.789496	193.177339	1155.129026	5.116875
-6.490000	0.000000	0.230000	20.000000
5.320000	3.000000	566.072500	28.052567
9.400000	6.000000	1166.155000	32.305784
14.870000	9.000000	1945.962500	36.496663
36.970000	2597.000000	4998.070000	50.000000

After making conversions, Here is the final result for the transformed data types of each column:

```
data.dtypes
```

```
ID                          object
Customer_ID                 object
Month                       object
Name                        object
Age                          int64
SSN                         object
Occupation                  object
Annual_Income              float64
Monthly_Inhand_Salary      float64
Num_Bank_Accounts            int64
Num_Credit_Card              int64
Interest_Rate                int64
Num_of_Loan                  int64
Type_of_Loan                object
Delay_from_due_date          int64
Num_of_Delayed_Payment     float64
Changed_Credit_Limit       float64
Num_Credit_Inquiries       float64
Credit_Mix                  object
Outstanding_Debt           float64
Credit_Utilization_Ratio   float64
Credit_History_Age          object
Payment_of_Min_Amount       object
Total_EMI_per_month        float64
Amount_invested_monthly    float64
Payment_Behaviour           object
Monthly_Balance            float64
Credit_Score                object
dtype: object
```

HANDLING MISSING VALUES

Addressing missing values: Given that the dataset contains multiple entries for each client, our approach involved retrieving absent client information by cross-referencing available data from other entries pertaining to the same client. For instance, if the client's Social Security Number (SSN) or Name was missing, we attempted to fill these gaps by identifying matching entries for that client within the dataset. Employing a similar methodology with slight variations, we successfully rectified other missing values. In this process, we replaced these gaps with the MODE or MEAN of the corresponding data for that specific CustomerID, depending on the nature of the respective column. As for the remaining missing values, where no associated information could be derived from other columns, we assigned them a constant value. Fortunately, these instances were infrequent, as the majority of missing values were resolved by leveraging data from other entries in the dataset.

As we see below there are 13 columns that have missing values. We applied different approaches for each column to handle their missing values or find missing values from different columns:

	Missing Percentage	Unique Value Count
ID	0.00	100000
Customer_ID	0.00	12500
Month	0.00	8
Name	9.98	10139
Age	0.00	1788
SSN	5.57	12500
Occupation	7.06	15
Annual_Income	0.00	18940
Monthly_Inhand_Salary	15.00	13235
Num_Bank_Accounts	0.00	943
Num_Credit_Card	0.00	1179
Interest_Rate	0.00	1750
Num_of_Loan	0.00	434
Type_of_Loan	11.41	6260
Delay_from_due_date	0.00	73
Num_of_Delayed_Payment	7.00	749
Changed_Credit_Limit	2.09	4375
Num_Credit_Inquiries	1.96	1223
Credit_Mix	20.20	3
Outstanding_Debt	0.00	13178
Credit_Utilization_Ratio	0.00	100000
Credit_History_Age	9.03	404
Payment_of_Min_Amount	0.00	3
Total_EMI_per_month	0.00	14950
Amount_invested_monthly	4.48	91049
Payment_Behaviour	7.60	6
Monthly_Balance	1.20	98792
Credit_Score	0.00	3

NAME, SSN:

For these two columns we observed that we could find their corresponding values from Customer_ID column in which the Name and SSN is not missing. For example in the table below we can see the list of ccustomers whose SSN or Name column have missing values.:

```
data[(data['Name'].isnull()) | (data['SSN'].isnull())]
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	An
7	0x1609	CUS_0xd40	August	NaN	23	NaN	Scientist	
17	0x161b	CUS_0x2dbc	February	NaN	34	486-85-3974	Engineer	
22	0x1620	CUS_0x2dbc	July	NaN	34	486-85-3974	Engineer	
29	0x162b	CUS_0xb891	June	Jasond	55	NaN	NaN	
51	0x164d	CUS_0x284a	April	Nadiaq	34	NaN	Lawyer	
...	...	...	...	...	...	...	...	
99968	0x25fc2	CUS_0xf16	January	Maria Sheahanb	44	NaN	Media_Manager	
99969	0x25fc3	CUS_0xf16	February	NaN	45	868-70-2218	Media_Manager	
99973	0x25fc7	CUS_0xf16	June	NaN	45	868-70-2218	Media_Manager	
99986	0x25fdc	CUS_0x8600	March	NaN	28	031-35-0942	Architect	
99988	0x25fde	CUS_0x8600	May	Sarah McBridec	28	NaN	Architect	

15000 rows × 28 columns

We choose one of those customers randlomly and further look into his data:

```
data[data['Customer_ID']=='CUS_0xd40'][['Customer_ID','Name','SSN']]
```

	Customer_ID	Name	SSN
0	CUS_0xd40	Aaron Maashoh	821-00-0265
1	CUS_0xd40	Aaron Maashoh	821-00-0265
2	CUS_0xd40	Aaron Maashoh	821-00-0265
3	CUS_0xd40	Aaron Maashoh	821-00-0265
4	CUS_0xd40	Aaron Maashoh	821-00-0265
5	CUS_0xd40	Aaron Maashoh	821-00-0265
6	CUS_0xd40	Aaron Maashoh	821-00-0265
7	CUS_0xd40	NaN	NaN

We observe that there are missing values for the customer CUS_0xd40 in his Name and SSN columns. We know that all these columns corresponds to the same person therefore we can replace the missing values for these columns with the available data we have for that customer in other rows. In order to do that we first find the unique customerIDs who has missing values in Name column then replace the missing values on that column with available data for that CustomerID.

```
Customer_IDs = data[data['Name'].isnull()]['Customer_ID'].values
```

```
# fill missing values
for id in Customer_IDs:
    # get real name by customer id
    realName = ''
    realName = data.loc[(data['Customer_ID'] == id) & (data['Name'].notna())]['Name'].values[0]
    # fill missing value
    data.loc[(data['Customer_ID'] == id) & (data['Name'].isna()), ['Name']] = realName
```

```
# check the Name column
data['Name'].isnull().value_counts()
```

We do the same approach for SSN column. An at the end we observe there are zero missing values left in these two columns, meaning that none of the CustomerIDs had all missing values for these columns

and we could successfully extract data for these columns.

OCCUPATION, TYPE_OF_LOAN, CHANGED_CREDIT_LIMIT, NUM_CREDIT_INQUIRIES, CREDIT_MIX, MONTHLY_INHAND_SALARY, AMOUNT_INVESTED_MONTHLY, PAYMENT_BEHAVIOUR, NUM_OF_DELAYED_PAYMENT, CREDIT_HISTORY_AGE

For the list of above columns we did slightly different approach in handling missing values. Although the same concept applied in here, meaning we tried to extract missing values from available data of corresponding CustomerID. However, instead of replacing with the exact values, we replaced missing values of these columns with the MODE of corresponding information of that CustomerID. This approach has been chosen because these columns correspond to discrete numeric values or underlying object values.

```
# Step 1: Find unique 'CustomerID' with missing values in 'Cre
missing_customers = data[data['Occupation'].isnull()]['Custome

# Step 2: Calculate the mode of 'Credit_Limit' for each of tho
modes = data.groupby('Customer_ID')['Occupation'].apply(lambda
                                                        .empty

# Step 3: Replace missing values with the corresponding mode
for customer_id in missing_customers:
    mode = modes.get(customer_id)
    if mode is not None:
        data.loc[(data['Customer_ID'] == customer_id) & (data[

```

For example for the attribute Credit_Mix. We can see Customer_ID =’ CUS_0x942c’ has 4 missing values in ‘Credirt_Mix’ column. We replaced those missing values with the Mode of other available Credit_Mix values for that Customer. The resulting values for that Customer is presented in the right column below:

Before Replacing Missing Values	After Replacing Missing Values
`data[data['Customer_ID']`	`data[data['Customer_ID']=`
99992 NaN 99993 Good 99994 NaN 99995 NaN 99996 NaN 99997 Good 99998 Good 99999 Good Name: Credit_Mix, dtype: obj	99992 Good 99993 Good 99994 Good 99995 Good 99996 Good 99997 Good 99998 Good 99999 Good Name: Credit_Mix, dtype: obje

We did the same approach for all the specified attributes above. And as the result we could see that all missing values were replaced with a value, meaning that all attributes above had at least one nonmissing value for each of those customers. However this was not the case fir Type_of_Loan attribute that we will discuss below:

TYPE_OF_LOAN

As you can see below there are many customers who have missing values for Type_of_loan andthose customers that have missing values for this column, do not have any available information that we can use to extract the type of loan.

```
data[data['Customer_ID']=='CUS_0xad4f']['Type_of_Loan']
```

```
99936    NaN
99937    NaN
99938    NaN
99939    NaN
99940    NaN
99941    NaN
99942    NaN
99943    NaN
Name: Type_of_Loan, dtype: object
```

Therefore we replaced missing values of this column with a constant separate category as ‘Not Specified’.

```
# Replace Missing with 'Not Specified'
replacement_value = 'Not Specified'
data['Type_of_Loan'].fillna(replacement_value, inplace=True)
```

```
data['Type_of_Loan'].isna().sum()
```

```
0
```

2.4 MONTHLY_BALANCE

We used the same apptoach for Monthly_Balance with slightly different method. As this attribute corresponds to float values, we replaced missing values with MEAN of available data:

Code:

```
# Replace with Mean
# Step 1: Find unique 'Customer_ID' with missing values in 'Monthly_
missing_history_customers = data[data['Monthly_Balance'].isnull()][

# Step 2: Calculate the mean of 'Monthly_Balance' for each of those
means = data.groupby('Customer_ID')['Monthly_Balance'].transform('m

# Step 3: Replace missing values with the corresponding mean
for customer_id in missing_history_customers:
    data.loc[(data['Customer_ID'] == customer_id) & (data['Monthly_

print(data[data['Customer_ID']=='CUS_0xa5f9'][['Customer_ID','Month
```

Result of Transformation:

Before Tansformation	After Transformation

```
1 data[data['Customer_
```

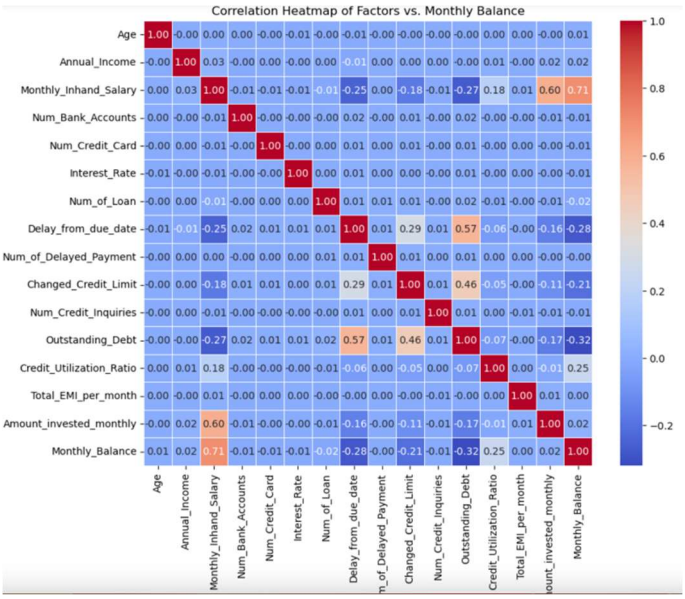
	Customer_ID	Monthly_Bal
192	CUS_0xa5f9	295.95
193	CUS_0xa5f9	327.78
194	CUS_0xa5f9	27.30
195	CUS_0xa5f9	324.01
196	CUS_0xa5f9	356.40
197	CUS_0xa5f9	
198	CUS_0xa5f9	316.92
199	CUS_0xa5f9	362.52

```
12
13 print(data[data['Custo
```

	Customer_ID	Monthly_B
192	CUS_0xa5f9	295.
193	CUS_0xa5f9	327.
194	CUS_0xa5f9	27.
195	CUS_0xa5f9	324.
196	CUS_0xa5f9	356.
197	CUS_0xa5f9	287.
198	CUS_0xa5f9	316.
199	CUS_0xa5f9	362.

However after this transformation we observe that there are still few customers who have missing values for Monthly_Balance as there were no available data for them to use to replace the missing values.

In order to address this we try to find if we can use any other column other than CustomerID to find those values from. After drawing the correlation matrix we find that Montly_Balance has very high correlation with Monthly_Inhand_Salary column with correlation of 0.70. Therefore we used this column to find the average monthly_Balance for the rest of missing values in this column.



In this regard we grouped data based on Monthly_Inhand_Salary and found the Mode for their corresponding Monthly_Balance and then replaced missing values with that Mode.

Code:

```
customer_mode_payment = data.groupby('Monthly_Inhand_Salary')['M

# Fill missing values with the corresponding customer's mode pay
data['Monthly_Balance'] = data['Monthly_Balance'].fillna(custome
```

Result:

Customer_ID= 'CUS_0x942c'

Before transformation		After Transformation	
	Monthly_Inhand_Sala		Monthly_Inhand_Sala
99992	3359.4158	99992	3359.4158
99993	3359.4158	99993	3359.4158
99994	3359.4158	99994	3359.4158
99995	3359.4158	99995	3359.4158
99996	3359.4158	99996	3359.4158
99997	3359.4158	99997	3359.4158
99998	3359.4158	99998	3359.4158
99999	3359.4158	99999	3359.4158

At the end we end up with dataframe with zero missing values:

```
1 data.isna().sum()
```

```
ID                          0
Customer_ID                 0
Month                       0
Name                        0
Age                         0
SSN                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Type_of_Loan                0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_Score                0
dtype: int64
```

EXPLORE FEATURES

Summary:
To explore the features further, we analyzed the data distributions, and used visualization like histograms and box plots. It helped us to understand the data distribution and assess the potential impact of outliers on our analysis.

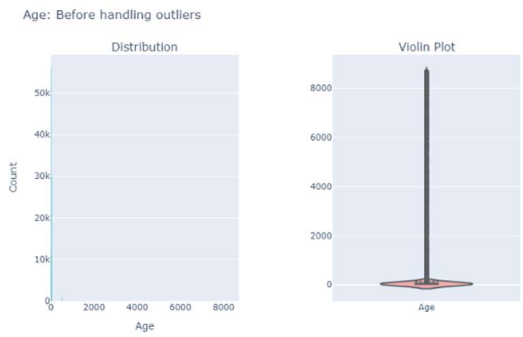
Visualizing distribution:
We used a histogram to show the distribution of each feature and a violin plot to visualize outliers.

Handling Outliers:
The data is grouped by 'Customer_ID,' and a list of non-null values for the specified feature is created within each group. The mode, representing the most frequent value, is then computed for each group, and both the mode_min and mode_max are extracted. Values in the feature that are less than 0, less than mode_min, or greater than mode_max are assigned NaN. Subsequently, NaN values are filled with the local mode computed for each group. If any NaN values persist, they are filled with the mean of the feature, ensuring a comprehensive treatment of missing or outlier values within the grouped data structure. At last the negative values are replaced with 0.

We have the following **numerical features** to analyze: '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', 'Total_EMI_per_month', 'Amount_invested_monthly', 'Monthly_Balance']

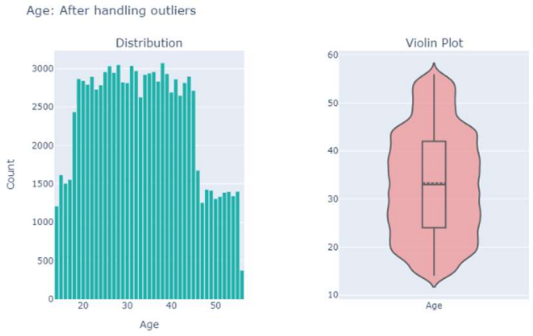
1. Age:

Distribution and box plot:



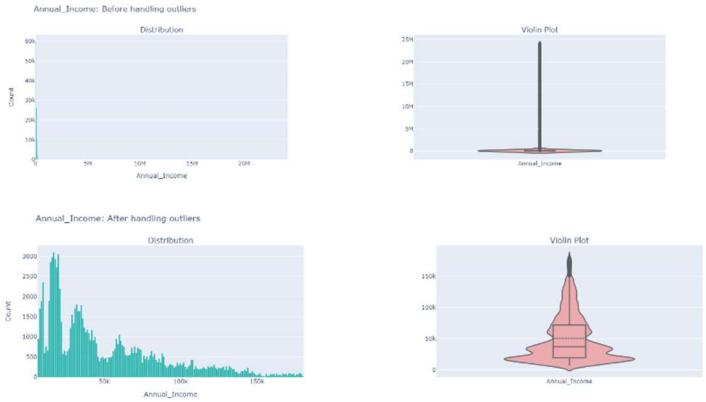
It shows that there are age values higher than 100 which are not realistic for human ages. The outliers seem to be data entry errors.

To address the issue of outliers in the 'Age' feature, we will first identify and handle values that are above a specified threshold (80 in our case) as outliers. Afterward, we identify unique customers who have these age outliers and fill in the missing age values for those customers based on non-missing age values of the same customer.



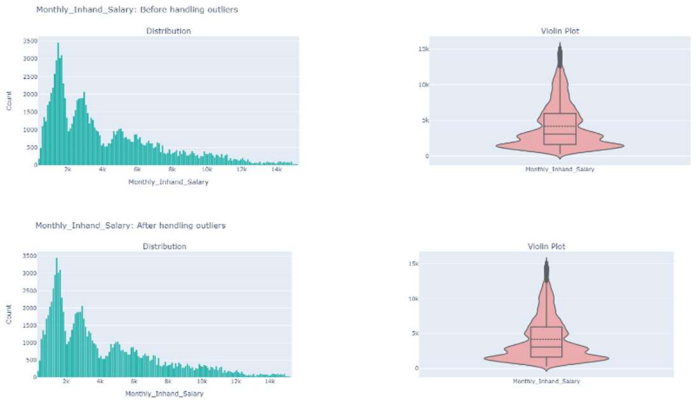
- The distribution of age appears to be relatively normally distributed.
- No significant outliers are visible from the box plot.
- Age is an essential feature for credit scoring, as it can correlate with an individual's financial stability and repayment ability.

2. Annual Income:



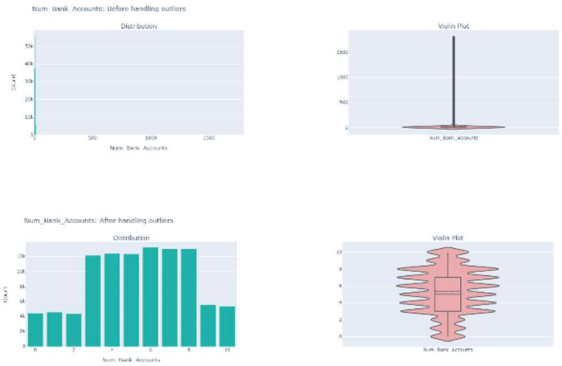
- The distribution of annual income is right-skewed.
- There are some outliers with exceptionally high incomes.
- High-income outliers might indicate high earning individuals, potentially low credit risk.

3. Monthly Inhand Salary:



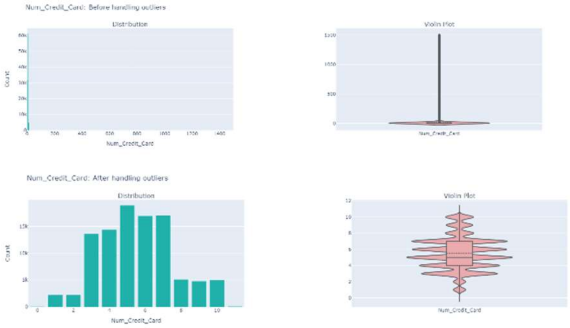
- The distribution appears to be similar to the annual income distribution but on a monthly basis.
- It also exhibits right-skewness with some high-income outliers.
- High monthly inhand salary may suggest good repayment capacity.

4. Number of Bank Accounts:



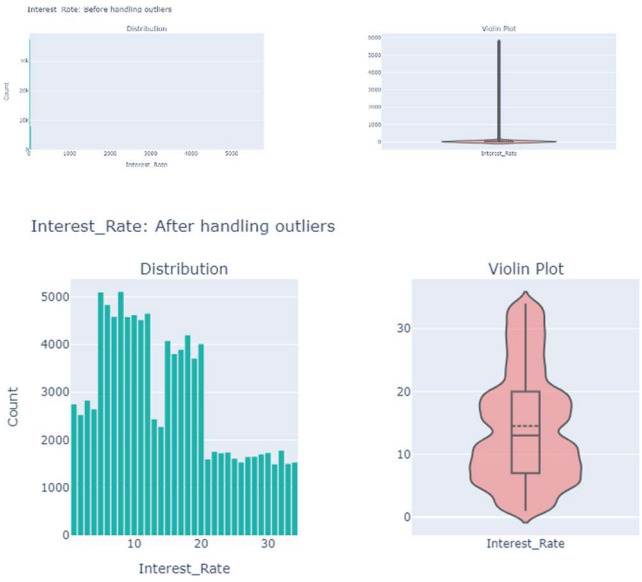
- The number of bank accounts is a discrete numerical variable.
- It is typically a count and has integer values.

Number of Credit Cards:



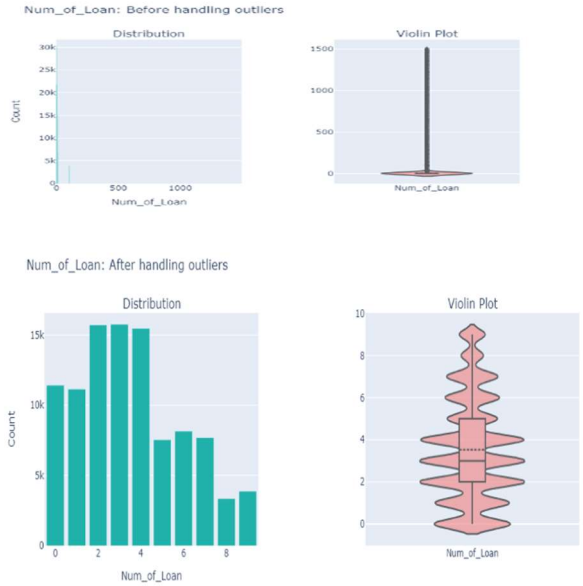
- Similar to the number of bank accounts, this is a discrete numerical variable.
- It represents the count of credit cards held by individuals

6. **Interest Rate:**



- Interest rates vary widely, and the distribution depends on the type of loans and the financial products under consideration.
- It appears normally distributed with no prominent outliers.
- Higher interest rates may indicate riskier loans.

7. **Number of Loans:**

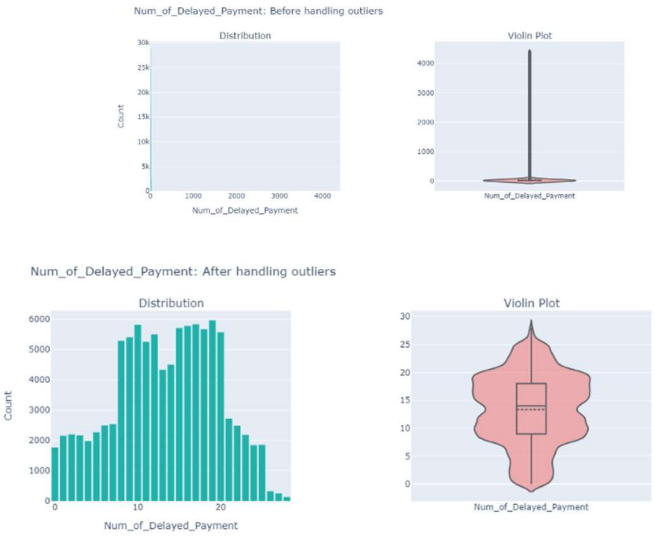


- The number of loans is another discrete numerical variable.
- It represents the count of loans held by individuals.

8. **Delay from Due Date:**

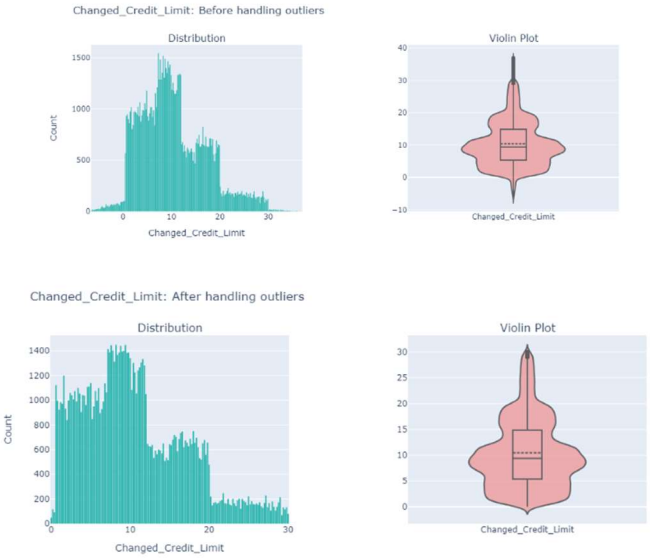
- This feature represents the time delay in loan repayments.
- Outliers indicate individuals with exceptionally long delays in repayments, which can be a red flag for creditworthiness.

Number of Delayed Payment:



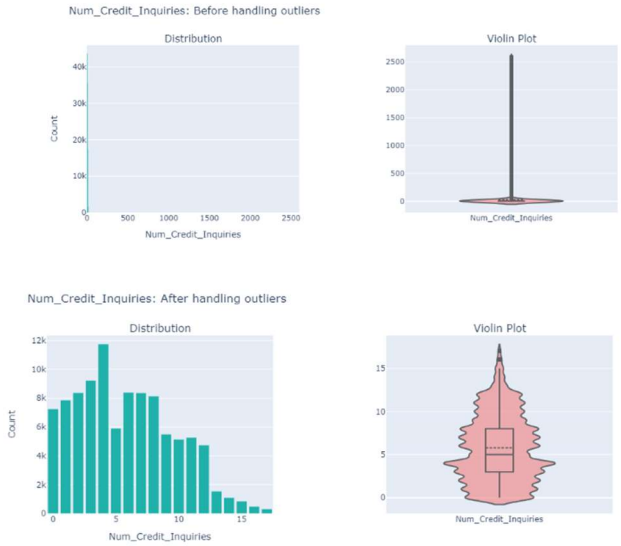
- Similar to delay from the due date, this represents the number of delayed payments.
- Outliers here might indicate individuals with a high frequency of delayed payments. But after adjustments we don't have outliers here.

10. Changed Credit Limit:



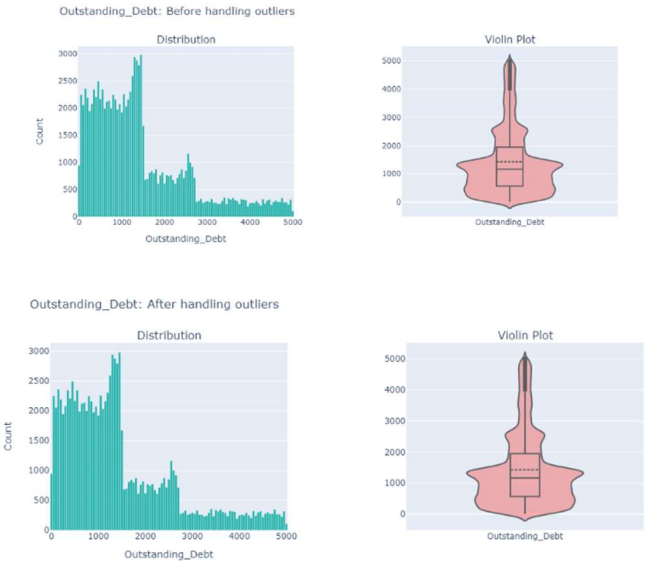
- This feature indicates changes in credit limits.
- Changes in credit limits are driven by changing financial conditions or credit management practices.

11. Number of Credit Inquiries:



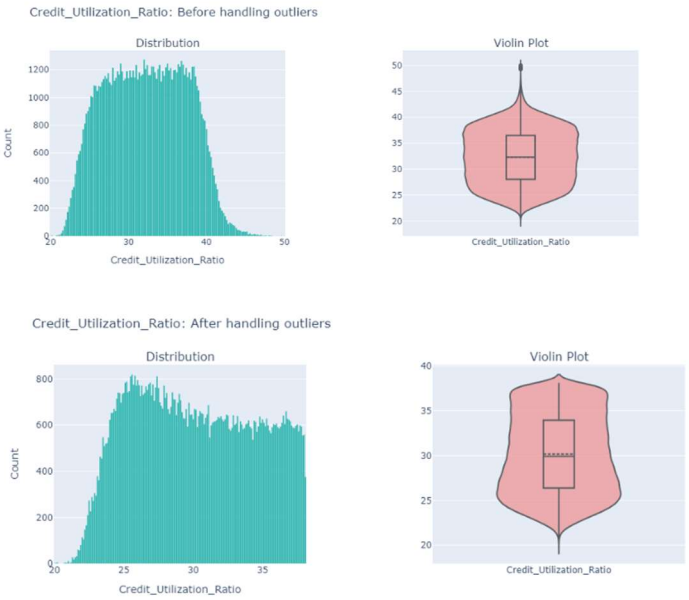
- Credit inquiries can affect credit scores and indicate how often individuals seek new credit.
- A high number of credit inquiries in a short period might indicate financial distress or frequent credit applications.
- There are some outliers with higher number of credit inquiries.

Outstanding Debt:



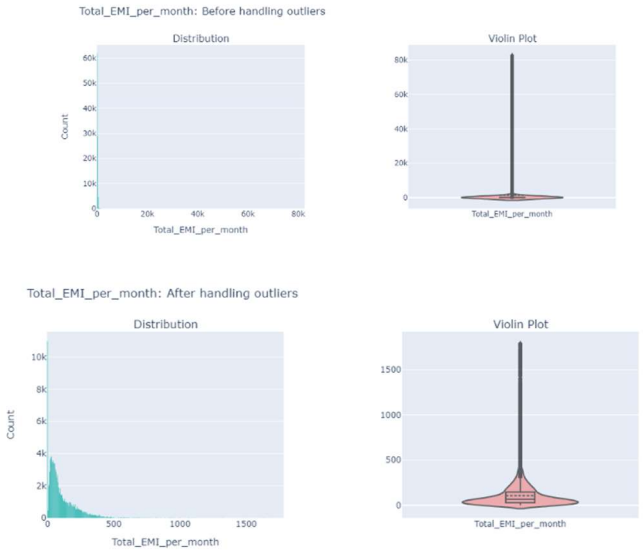
- The distribution of outstanding debt appears right-skewed, suggesting that a few individuals have significantly higher debt balances.
- High outstanding debt can negatively impact credit scores and repayment capacity.
- There are some outliers with higher outstanding debt that impacts credit scores.

13. **Credit Utilization Ratio:**



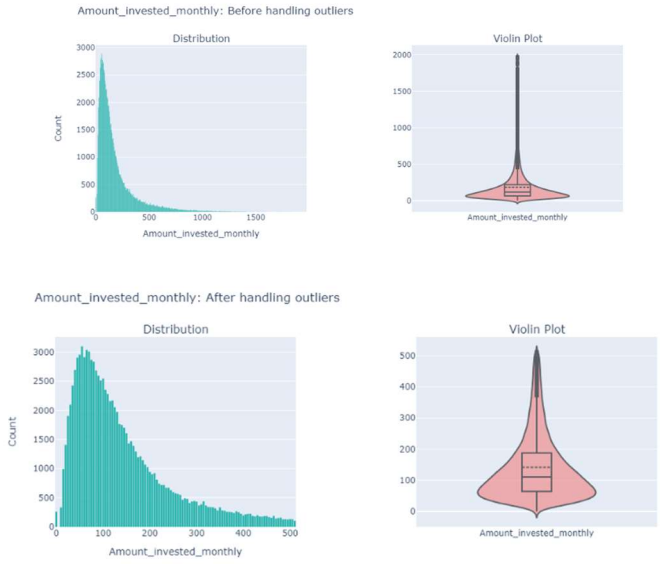
- Credit utilization is an important factor in credit scoring. High utilization (close to the credit limit) can negatively affect credit scores.
- Outliers may indicate individuals with high credit utilization.
- The distribution appears normal without outliers.

14. Total EMI per Month:



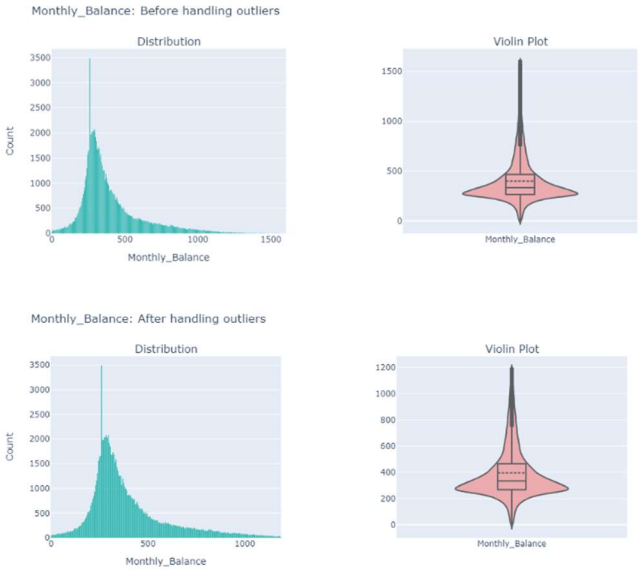
- The distribution appears right-skewed with some high outliers.
- High EMIs can impact an individual's capacity to take on new loans.

15. Amount Invested Monthly:



- This feature represents investments made by individuals.
- Outliers may suggest high investment activity or large investment amounts.

16. Monthly Balance:



- Monthly balance represents the available funds individuals have at the end of the month.
- Low monthly balances might indicate financial stress, while high balances might suggest financial stability.
- The outliers are crucial for credit score analysis

DATA PREPROCESSISING

```
[16] from sklearn.preprocessing import LabelEncoder

     # Identify categorical columns
     categorical_columns = df.select_dtypes(include=['object']).columns

     # Apply label encoding to categorical columns
     le = LabelEncoder()
     for column in categorical_columns:
         df[column] = le.fit_transform(df[column])
```

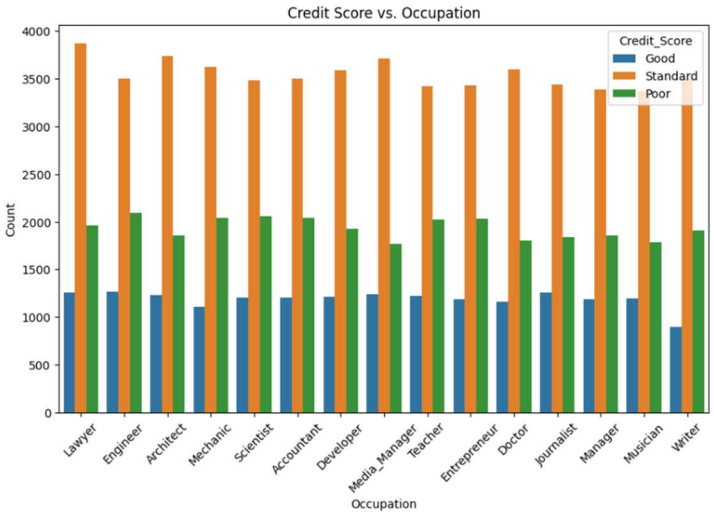
For data preprocessing we used label encoding, For each column, we applied the fit_transform method of the LabelEncoder to transform the categorical values into numerical values. The transformed numerical values were stored back in the same columns, effectively replacing the original categorical values with their corresponding numerical representations.

```
Unique encoded values for ID: [10277 10286 10295 ... 39167 39168 39169]
Unique encoded values for Customer_ID: [10351   894  1592 ...  8949  6593  7422]
Unique encoded values for Month: [3 2 6 0 7 5 4 1]
Unique encoded values for Name: [  76 6499 4267 ... 1365 6887 5677]
Unique encoded values for SSN: [8556   48 5124 ... 1415  342  849]
Unique encoded values for Occupation: [12 13  4  5  2  7 10  3  6  8  0 11  9 14  1]
Unique encoded values for Type_of_Loan: [ 105  547   53 ... 1701 3901 3972]
Unique encoded values for Credit_Mix: [1 2 0]
Unique encoded values for Credit_History_Age: [180 184 185 186 187 188 236 237 229 2
 116 117 118 297 298 290 299 300 303 311 318  73  81  82  74  84  87 170
 235 135 136 137 138 139 140 141 222 224 225 226 227 245 383 392 393 396
 397 398 400 123 124 125 126 127 129 130  97  98 107 108 111 274 363 364
 365 366 367 358 359 368 244 142  24  27  28  29  30  32  33  54  56  57
  49  52  55  58 109 120  69  62  71  72  75 292 293 294 295 296 289 128
 121 131 390 391 382  60  63  64  66  67  68 174 175 176 177 178 169 179
 217 228 231  70 251 252 255 256 258 259 371 380 384 385 387 212 213 214
 208  15  16  17  18  19  20  21  26  35  41  42 156  25  34  77 165 166
 171 309 310  43  44  45  46  37 205 206 215 216 219 220 119  31 304 305
 306 307 308 350 351 352 353 354 355 209 210 211 101 102 103 104 105 106
 386 388 389 322 313 323 326  51 302 312 315 316 317 319  50  65 266 291
 324 325 327  39  40 257 260 261 253 267 268 270 271 272 273 132 159 160
 161 162 163 164 262 254 263  22  14 328 329 330 331 264 343 335 344 345
 348 349 301 346 347 357 360 361 133 181 182 190 192 195 196 183  89  90
  91  92  93  94  86  59 148 149 150 151 152 153 154 145 234  85  99 100
 399 401 402 403 246 248 249 122 381 238  48  38  53 232 242 168 191 197
  95  78  79  80  83  96 189 198 199 221 223 218  88 379 370 241 314 369
 372 373 374 375 233 200 201 173 172 158 287 378 321 265 356 193 203 207
 394 250  47  36 377 194 134 155 269 362   0   3   5   6   7   8   9 247
 395 288   4 376  23  13  76 204 339 340 341 342 278 333 336 338 157 167
 332 337 320  61 202 279 280 281 283 284 286 334 144  10 146 275 285 277
  11  12   2   1 147 143 276 282]
Unique encoded values for Payment_of_Min_Amount: [1 0 2]
Unique encoded values for Payment_Behaviour: [2 3 4 5 1 0]
Unique encoded values for Credit_Score: [0 2 1]
```

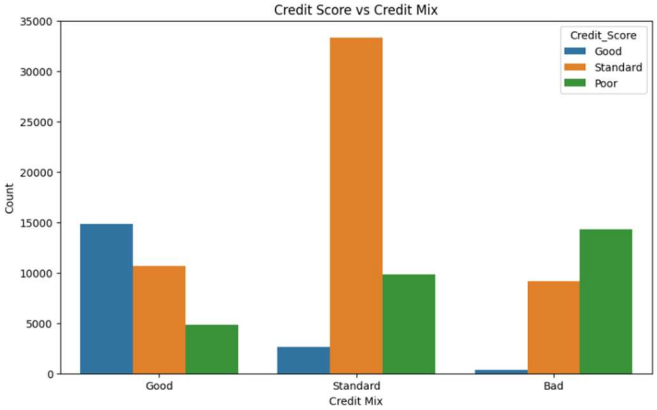
ANALYZING DISTRIBUTION OF YOUR TARGET VARIABLE.

The following is the analysing distribution of the target variable, our target variable in the data is Credit score When analysing the distribution of our target variable, which is "Credit Score," we observe that the distribution is imbalanced. To mitigate the impact of the imbalanced distribution, we plan to use stratified sampling in our modelling process. Stratification ensures that each class is proportionately represented in the training and testing datasets.

Credit Score was chosen as the target variable due to its pivotal role in financial decision-making and its direct impact on creditworthiness. Our analysis of 'Credit Score' in relation to 'Annual Income' and 'Occupation' aims to provide actionable insights for financial decision-makers and individuals seeking to enhance their financial standing.

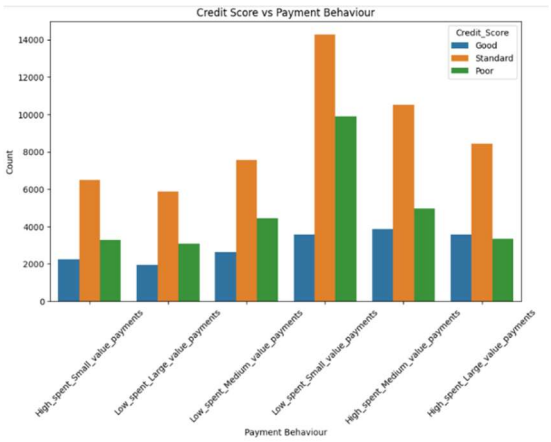


Further Visualizations credit mix Vs Credit Score and Credit score Vs Payment behaviour was added, This visualization illustrates the relationship between credit mix categories (e.g., 'Good,' 'Standard,' 'Poor') and credit scores, showing that customers with a 'Good' credit mix tend to have higher credit scores compared to those with a 'Poor' credit mix , This chart reveals how different payment behaviors, such as 'High_spent_Small_value_payments' and 'Low_spent_Small_value_payments,' are associated with credit scores, emphasizing that customers exhibiting 'High_spent_Small_value_payments' typically have better credit scores, while those with 'Low_spent_Small_value_payments' may have lower scores.



This visualization depicts the relationship between customers' occupations and their credit scores, highlighting variations in credit scores across different occupations. It suggests that certain occupations may be associated with higher or lower credit scores, providing insights into how employment may impact creditworthiness.

The stacked bar plot shows that 'Standard' credit scores are common across different age groups, while 'poor' scores are also prevalent. Good credit scores are less frequent in the dataset, suggesting credit challenges in a smaller portion of individuals. This analysis reveals the credit score distribution by age, which can be useful for assessing the creditworthiness of various age groups.



For our credit scoring model, we applied supervised machine learning techniques because of the nature of our dataset - labeled instances of credit classes. Within supervised learning, we chose classification over regression since our business case requires categorizing applicants into discrete credit classes ('Poor', 'Standard', 'Good')—making it a multi-class classification problem.

We experimented with a variety of algorithms to establish benchmarks and understand the performance of the algorithms:

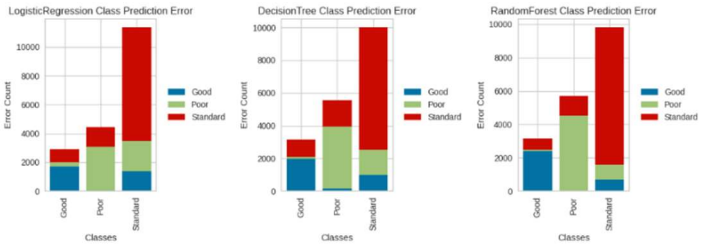
Logistic Regression: As a baseline because it's a fundamental classification technique that works well for binary problems, which we extended to the multi-class scenario using the one-vs-rest approach.

Decision Trees: Provide a clear indication of feature importance but can be prone to overfitting, which can be controlled by tuning the depth and minimum samples per leaf.

Random Forest: An ensemble method that builds multiple decision trees and merges them to get a more accurate and stable prediction.

Gradient Boosting Algorithms (XGBoost, LightGBM): Ability to build and train sequential trees over the errors of the previous one, leads to superior performance and reduction in residual errors.

We **split the data** into training and testing sets, typically using a 70:30 ratio to ensure that our models are validated on unseen data.



These visualizations represents the misclassification errors for three classes: Good, Poor, and Standard.

- The Logistic Regression misclassifies a significant number of observations across all classes, with the most errors occurring in the Standard class, followed by the Good class.
- The Decision Tree is also likely to misclassify the Standard class, although with a lower magnitude in comparison to Logistic Regression.
- The Random Forest, XGBoost, and LightGBM models have a lower number of errors overall. However, they still show some misclassifications for the Standard class, which is common issue among all the models however ensemble models have performed relatively better.

Hyperparameter Tuning:

- **Logistic Regression:**
  Grid Search Parameters:
  C: The inverse of regularization strength where smaller values specify stronger regularization, tested with values 0.1, 1, and 10.
- **Decision Tree:**
  Grid Search Parameters:
  max_depth: The maximum depth of the tree, with tested values of 3, 5, and 10.
  min_samples_split: The minimum number of samples required to split an internal node, with tested values of 2, 5, and 10.
- **Random Forest:**
  Grid Search Parameters:
  n_estimators: The number of trees in the forest, with tested values of 50, 100, and 200.
  max_depth: The maximum depth of the trees, with tested values including unrestricted growth (None), 10, and 20.
  min_samples_split: The minimum number of samples required to split a node, with tested values of 2, 5, and 10.
- **XGBoost:**
  use_label_encoder=False: Disables the use of label encoding to handle categorical variables.
  eval_metric='mlogloss': Uses the multinomial log-likelihood loss function for evaluation.

Grid Search Parameters:
learning_rate: The step size shrinkage used to prevent overfitting, with tested values of 0.01, 0.05, and 0.1.
n_estimators: The number of gradient-boosted trees, with tested values of 100, 150, and 200.
max_depth: The maximum depth of each tree, with tested values of 3, 5, and 7.

Grid Search: A systematic approach to tuning hyperparameters. We use this method to find the optimal combination of parameters for each model. Grid Search trains a model for every combination of parameters specified in model_params and evaluates each model's performance to select the best parameters. This method ensures that we explore a wide range of parameter settings and select the most effective ones to improve our models' predictive power.

Cross-Validation: After determining the best hyperparameters, it conducts a separate 5-fold cross-

validation on the training data using the best model configuration to estimate its performance. This is outputted as both individual fold scores and the average across folds.

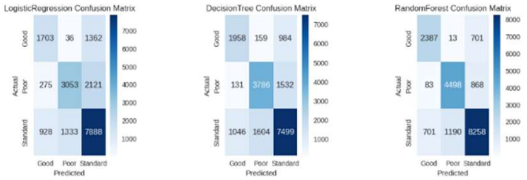
EVALUATION

We utilized several evaluation **metrics to evaluate our models' performance**:

Accuracy: Provides a general sense of overall correctness but can be misleading in imbalanced datasets.

Precision and Recall: Especially important in financial contexts where the cost of false negatives (predicting a bad credit customer as good) can be high.

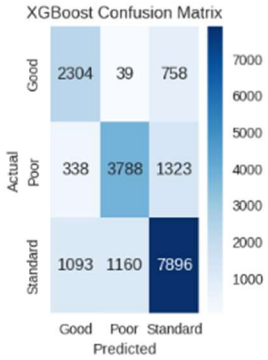
F1 Score: Harmonic means of precision and recall, which is crucial when we need a balance between precision and recall.

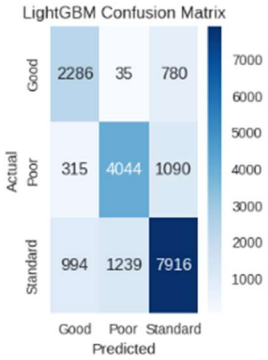


Based on the results, the Random Forest classifier has the highest accuracy, recall and F1 Score, making it the best-performing model. Ensemble methods like Random Forest generally perform better as they combine decisions from multiple trees, reducing variance and overfitting.

The model would perform reasonably well on unseen data, especially the Random Forest model. However, some models like Logistic Regression might not perform as well given their lower performance on multi-class classification.

Confusion Matrix: To visualize the correct and incorrect predictions across different classes





The Logistic Regression model shows high confusion between the Good and Standard classes, with a considerable number of Good instances predicted as Standard.

- confusion matrix suggests that decision tree has a higher tendancy of misclassifying observation belonging to all classes when compared to ensemble models.
- The Random Forest, XGBoost, and LightGBM models' confusion matrices illustrates a more balanced classification with lower misclassified observations, especially for the Standard class, highlighting better performance.
- Differentiating between the Poor and Standard classes is a common limitation for all models, as depicted in the visuals.

The **Random Forest model performed best**, likely due to its ability to handle the variance in the data and its robustness to overfitting compared to a single decision tree.

WHAT SHORTCOMINGS YOUR MODEL HAS AND POSSIBLE IMPLICATIONS?

The Random Forest model may suffer from complexity issues, making it less interpretable and potentially less suitable for situations that demand transparency in decision-making. It is resource-intensive, both in terms of computation and the need for expert knowledge for tuning, which could be problematic for large datasets or in environments with limited computational resources. Additionally, the model could exhibit bias towards majority classes, affecting its performance on imbalanced datasets and leading to fairness concerns.

What would you do to improve the results?

Model	Accuracy	F1_Score	Precision	Recall
RandomForest	0.809829	0.810087	0.810849	0.809829
LightGBM	0.761859	0.763180	0.766090	0.761859
XGBoost	0.748061	0.749211	0.753193	0.748061
DecisionTree	0.708220	0.708473	0.708801	0.708220
LogisticRegression	0.676186	0.672175	0.674883	0.676186

- **Feature Engineering:** More predictive features could be engineered using techniques like binning or aggregating features.

- **Hyperparameter Tuning:** Optimize hyperparameters using **grid search or randomized search** to improves model performance.
- **Cross-Validation:** Use k-fold cross-validation to ensure that the model's performance is consistent across different subsets of the data.
- **Advanced Sampling Techniques:** oversampling or under sampling techniques to address class imbalance, such as **SMOTE.**

TOP-DOWN FEATURE SELECTION USING RANDOM FOREST

The goal of this exercise was to identify the most significant features impacting the target variable 'Credit_Score', and to improve the overall efficiency and performance of the Random Forest model.

This approach started with the full set of features, iteratively removing the least important feature (as indicated by the Random Forest model) in each round. After each removal, the model was retrained, and its performance was evaluated.

- FEATURE IMPORTANCE

  The initial model highlighted the relative importance of each feature. Features like 'Outstanding_Debt', 'Interest_Rate', and 'Credit_Mix' appeared as significant predictors, whereas others like 'Payment_of_Min_Amount' had lesser importance.

- ITERATIVE REMOVAL AND MODEL PERFORMANCE:

  With each iteration of feature removal, the model's performance was closely monitored. A gradual decline in accuracy was observed as less important features were removed. However, the model maintained a relatively high-performance level until a critical point was reached, beyond which the performance notably decreased.

- OPTIMAL FEATURE SET

  The optimal feature set was determined based on the balance between model simplicity and accuracy. The final model, with a reduced set of features, achieved comparable accuracy to the initial model with all features, indicating a successful feature reduction with minimal loss in predictive power.

Data Preprocessing Techniques:

Data preprocessing involved several stages, each chosen with the aim of improving model performance:

- Missing values:

  Given that the dataset contains multiple entries for each client, our approach involved retrieving absent client information by cross-referencing available data from other entries pertaining to the same client. For instance, if the client's Social Security Number (SSN) or Name was missing, we attempted to fill these gaps by identifying matching entries for that client within the dataset. Employing a similar methodology with slight variations, we successfully rectified other missing values. In this process, we replaced these gaps with the MODE or MEAN of the corresponding data for that specific CustomerID, depending on the nature of the respective column. As for the remaining missing values, where no associated information could be derived from other columns, we assigned them a constant value. Fortunately, these instances were infrequent, as the majority of missing values were resolved by leveraging data from other entries in the dataset.

- Outliers:

for the outliers we implemented the Interquartile Range (IQR) method for identifying and removing outliers in a DataFrame. The IQR method is a statistical technique used to assess the spread and distribution of data. We chose this method as it is a good approach for handling outliers because it is robust and resistant to extreme values, making it a reliable choice for identifying and addressing outliers in a dataset. It focuses on the range between the first quartile (Q1) and the third quartile (Q3) of a dataset. Outliers are detected by calculating the IQR, which is the difference between Q3 and Q1. A data point is considered an outlier if it falls below the lower boundary (Q1 - iqr_multiplier * IQR) or above the upper boundary (Q3 + iqr_multiplier * IQR). This approach is particularly robust to the presence of extreme values and allows for customized outlier detection by adjusting the iqr_multiplier parameter. By iteratively applying the IQR method to numeric columns, the code identifies and removes outliers from the DataFrame, resulting in a more reliable and

robust dataset for subsequent analyses and modeling.

- Categorical encoding techniques such label encoding was applied to convert categorical variables into a machine-readable format.
- Feature scaling made sure that all the details (like income, age, etc.) were given equal importance in the model's decisions. This also helped some of the models work faster and better.

EVALUATION METRIC DECISION PROCESS:

- the cost of **misclassifying a bad credit risk as good (false negative)** could be significantly higher than the reverse. Therefore, **recall was selected**, as it evaluates the ability to correctly identify all positive observations. We also evaluated F1 score, which balances precision and recall, **to minimize false positives where a potentially good customer is denied credit.**
- confusion matrix for each model to examine the balance between false positives and false negatives and determined the F1, precision, and recall scores for a more in-depth evaluation.

MACHINE LEARNING ALGORITHM SELECTION:

- **Logistic Regression** was selected for its simplicity and interpretability, which allows for easy explanation to stakeholders. **Decision Trees** were included for their ability to navigate decision paths.
- Ensemble methods were utilized for their improved predictive power, which often comes at the expense of explain ability. These models work well with **unbalanced datasets.**
- The comparison of class prediction errors to assess the performance in classifying different classes and to further support the rationale behind the algorithm selection process.