Banking

Description

Data Analysis is the process of creating a story using the data for easy and effective communications. It utilizes visualization methods such as plots, charts, and tables to convey what the data holds beyond the formal modelling or hypothesis testing task.

Read the information given below and also refer to the data dictionary provided separately in an excel file to build your understanding.

Problem Statement

Financial institutions incur significant losses due to the default of vehicle loans. This has led to the tightening up of vehicle loan underwriting and an increase in vehicle loan rejection rates. The need for a better credit risk scoring model is also raised by these institutions. This warrants a study to estimate the determinants of vehicle loan default.

There is one dataset with data that has 41 attributes.

You are required to determine and examine factors that affect the ratio of vehicle loan defaulters. Also, use the findings to create a model to predict the potential defaulters.

Approach:

1. Data Preliminary Analysis:

1. Perform preliminary data inspection and report the findings like the structure of the data, missing values, duplicates, etc.
2. Variable names in the data may not be in accordance with the identifier naming in Python. Change the variable names accordingly.
3. The presented data might also contain missing values, therefore, exploration will also lead to devising strategies to fill in the missing values. Devise strategies while exploring the data.

2. Performing EDA:

1. Provide the statistical description of the quantitative data variables
2. How is the target variable distributed overall?
3. Study the distribution of the target variable across the various categories like branch, city, state, branch, supplier, manufacturer, etc.
4. What are the different employment types given in the data? Can a strategy be developed to fill in the missing values (if any)? Use pie charts to express how different types of employment defines defaulter and non-defaulters.
5. Has age got something to do with defaulting? What is the distribution of age w.r.t. to defaulters and non-defaulters?
6. What type of ID is presented by most of the customers as proofs?
7. Explain the factors in the data that may have an effect on ratings e.g. No. of cuisines, cost, delivery option, etc.

3. What type of ID was presented by most of the customers as proofs?

4. Performing EDA and Modeling:

   1. Study the credit bureau score distribution. How is the distribution for defaulters vs. non-defaulters? Explore in detail.
   2. Explore the primary and secondary account details. Is the information in some way related to the loan default probability?
   3. Is there a difference between the sanctioned and disbursed amount of primary and secondary loans? Study the difference by providing appropriate statistics and graphs.
   4. Do customers who make higher numbers of inquiries end up being higher risk candidates?
   5. Is credit history, that is, new loans in the last six months, loans defaulted in the last six months, time since the first loan, etc., a significant factor in estimating the probability of loan defaulters?
   6. Perform logistic regression modeling, predict the outcome for the test data, and validate the results using the confusion matrix.

1**. Data Preliminary Analysis:**

1. Perform preliminary data inspection and report the findings like the structure of the data, missing values, duplicates, etc.
2. Variable names in the data may not be in accordance with the identifier naming in Python. Change the variable names accordingly.
3. The presented data might also contain missing values, therefore, exploration will also lead to devising strategies to fill in the missing values. Devise strategies while exploring the data.

[2]:
```python
import pandas as pd
data=pd.read_excel("Bankinjg data.xlsx")
```

[3]:
```python
data.head()
```

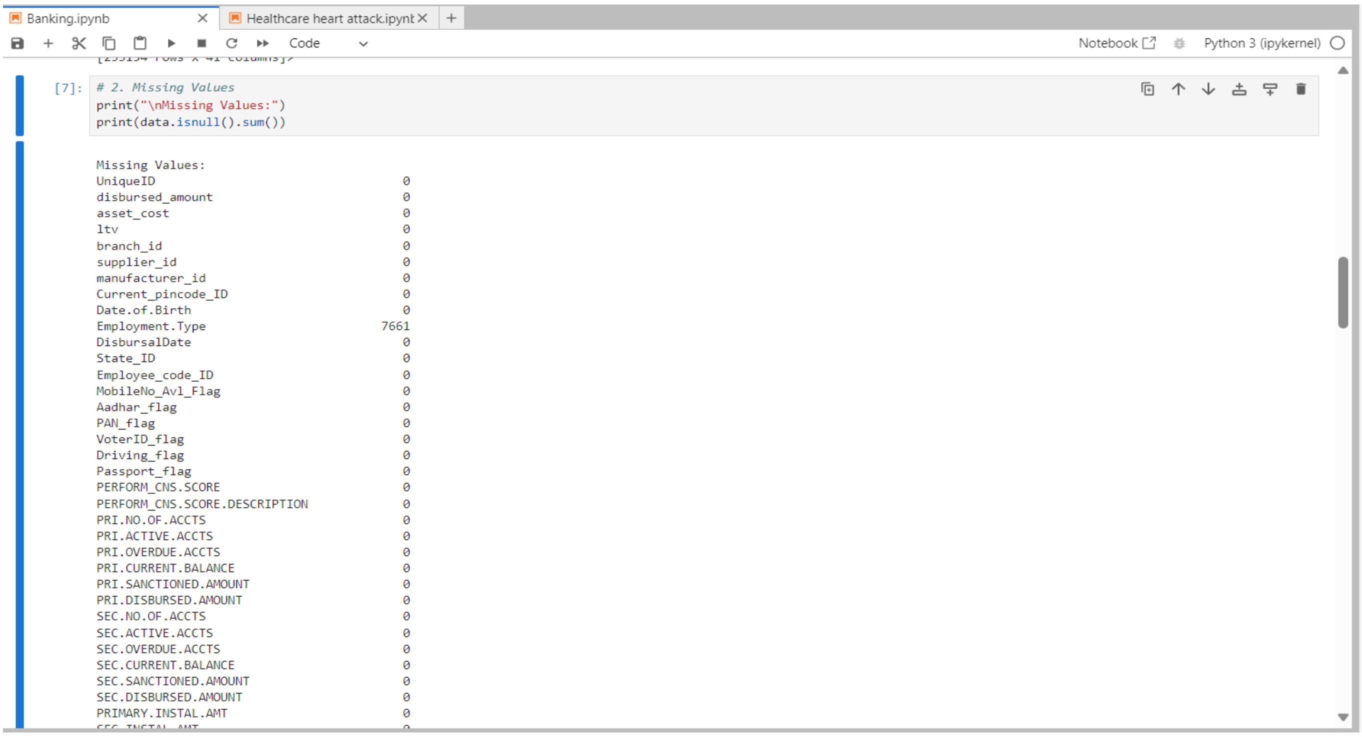
[3]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pincode_ID	Date.of.Birth	Employment.Type	...	SEC.SANCTIONED.AMOUNT	SEC.DIS
0	420825	50578	58400	89.55	67	22807	45	1441	1984-01-01	Salaried	...	0	
1	417566	53278	61360	89.63	67	22807	45	1497	1985-08-24	Self employed	...	0	
2	539055	52378	60300	88.39	67	22807	45	1495	1977-12-09	Self employed	...	0	
3	529269	46349	61500	76.42	67	22807	45	1502	1988-06-01	Salaried	...	0	
4	563215	43594	78256	57.50	67	22744	86	1499	1994-07-14	Self employed	...	0	

5 rows × 41 columns

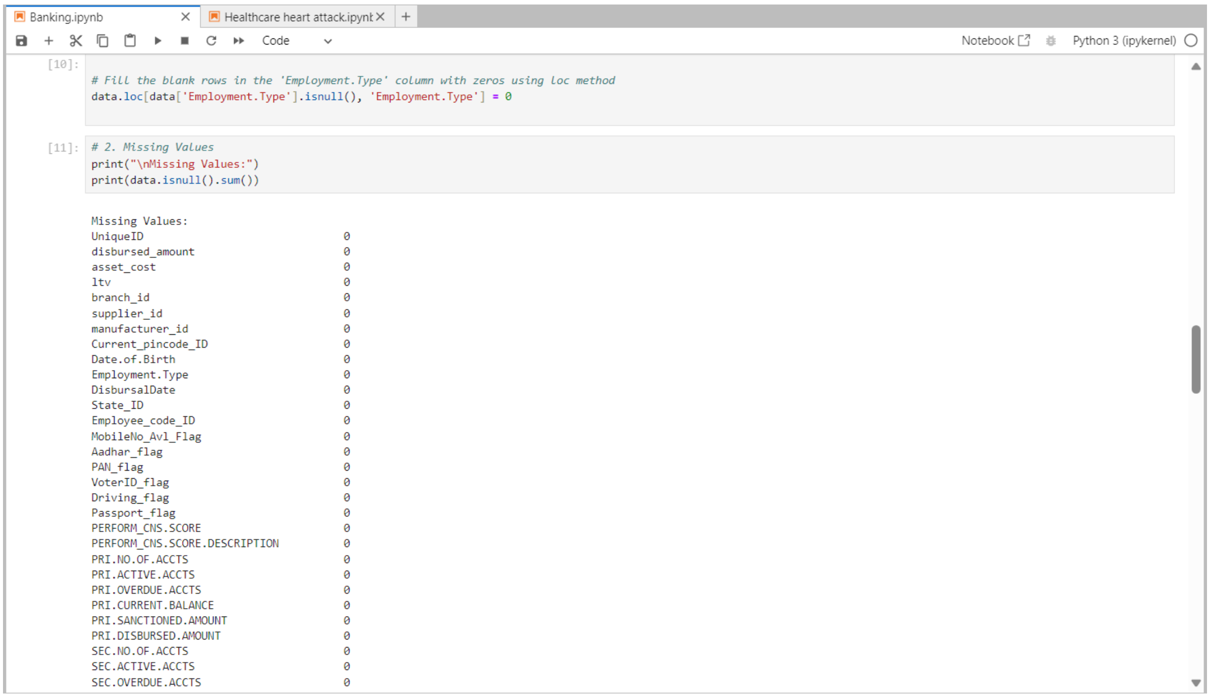
[6]:
```python
print(data.describe)
```

```
<bound method NDFrame.describe of         UniqueID  disbursed_amount  asset_cost    ltv  branch_id  supplier_id  \
0         420825             50578       58400  89.55         67        22807
1         417566             53278       61360  89.63         67        22807
2         539055             52378       60300  88.39         67        22807
3         529269             46349       61500  76.42         67        22807
4         563215             43594       78256  57.50         67        22744
...          ...               ...         ...    ...        ...          ...
233149    561031             57759       76350  77.28          5        22289
233150    649600             55009       71200  78.72        138        17408
233151    603445             58513       68000  88.24        135        23313
233152    442948             22824       40458  61.79        160        16212
233153    545300             35299       72698  52.27          3        14573

        manufacturer_id  Current_pincode_ID Date.of.Birth Employment.Type  \
0                    45                1441    1984-01-01        Salaried
1                    45                1497    1985-08-24   Self employed
2                    45                1495    1977-12-09   Self employed
3                    45                1502    1988-06-01        Salaried
```



```
[7]: # 2. Missing Values
     print("\nMissing Values:")
     print(data.isnull().sum())

Missing Values:
UniqueID                                0
disbursed_amount                        0
asset_cost                              0
ltv                                     0
branch_id                               0
supplier_id                             0
manufacturer_id                         0
Current_pincode_ID                      0
Date.of.Birth                           0
Employment.Type                      7661
DisbursalDate                           0
State_ID                                0
Employee_code_ID                        0
MobileNo_Avl_Flag                       0
Aadhar_flag                             0
PAN_flag                                0
VoterID_flag                            0
Driving_flag                            0
Passport_flag                           0
PERFORM_CNS.SCORE                       0
PERFORM_CNS.SCORE.DESCRIPTION           0
PRI.NO.OF.ACCTS                         0
PRI.ACTIVE.ACCTS                        0
PRI.OVERDUE.ACCTS                       0
PRI.CURRENT.BALANCE                     0
PRI.SANCTIONED.AMOUNT                   0
PRI.DISBURSED.AMOUNT                    0
SEC.NO.OF.ACCTS                         0
SEC.ACTIVE.ACCTS                        0
SEC.OVERDUE.ACCTS                       0
SEC.CURRENT.BALANCE                     0
SEC.SANCTIONED.AMOUNT                   0
SEC.DISBURSED.AMOUNT                    0
PRIMARY.INSTAL.AMT                      0
```

Found 7661 missing value in column ‘Employment. Type’, so Replace it with zero



```
[10]: # Fill the blank rows in the 'Employment.Type' column with zeros using loc method
      data.loc[data['Employment.Type'].isnull(), 'Employment.Type'] = 0

[11]: # 2. Missing Values
      print("\nMissing Values:")
      print(data.isnull().sum())

Missing Values:
UniqueID                          0
disbursed_amount                  0
asset_cost                        0
ltv                               0
branch_id                         0
supplier_id                       0
manufacturer_id                   0
Current_pincode_ID                0
Date.of.Birth                     0
Employment.Type                   0
DisbursalDate                     0
State_ID                          0
Employee_code_ID                  0
MobileNo_Avl_Flag                 0
Aadhar_flag                       0
PAN_flag                          0
VoterID_flag                      0
Driving_flag                      0
Passport_flag                     0
PERFORM_CNS.SCORE                 0
PERFORM_CNS.SCORE.DESCRIPTION     0
PRI.NO.OF.ACCTS                   0
PRI.ACTIVE.ACCTS                  0
PRI.OVERDUE.ACCTS                 0
PRI.CURRENT.BALANCE               0
PRI.SANCTIONED.AMOUNT             0
PRI.DISBURSED.AMOUNT              0
SEC.NO.OF.ACCTS                   0
SEC.ACTIVE.ACCTS                  0
SEC.OVERDUE.ACCTS                 0
```

Find Duplicates: No duplicates found

[12]:

```
# 3. Duplicates
print("\nDuplicate Rows:")
print(data[data.duplicated()])
```

```
Duplicate Rows:
Empty DataFrame
Columns: [UniqueID, disbursed_amount, asset_cost, ltv, branch_id, supplier_id, manufacturer_id, Current_pincode_ID, Date.of.Birth, Employment.Type, DisbursalDate, Stat
e_ID, Employee_code_ID, MobileNo_Avl_Flag, Aadhar_flag, PAN_flag, VoterID_flag, Driving_flag, Passport_flag, PERFORM_CNS.SCORE, PERFORM_CNS.SCORE.DESCRIPTION, PRI.NO.O
F.ACCTS, PRI.ACTIVE.ACCTS, PRI.OVERDUE.ACCTS, PRI.CURRENT.BALANCE, PRI.SANCTIONED.AMOUNT, PRI.DISBURSED.AMOUNT, SEC.NO.OF.ACCTS, SEC.ACTIVE.ACCTS, SEC.OVERDUE.ACCTS, S
EC.CURRENT.BALANCE, SEC.SANCTIONED.AMOUNT, SEC.DISBURSED.AMOUNT, PRIMARY.INSTAL.AMT, SEC.INSTAL.AMT, NEW.ACCTS.IN.LAST.SIX.MONTHS, DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS,
AVERAGE.ACCT.AGE, CREDIT.HISTORY.LENGTH, NO.OF_INQUIRIES, loan_default]
Index: []

[0 rows x 41 columns]
```

[ ]:

```
## no duplicates
```

2. Performing EDA:

1. Provide the statistical description of the quantitative data variables

```
[ ]: ###2. Performing EDA:
```

```
[ ]: ##1.provide the statistical description of the quantitative data variables
```

```
[15]: data.describe()
```

[15]:

sbursed_amount	primary_installment_amount	secondary_installment_amount	new_accounts_in_last_six_months	delinquent_accounts_in_last_six_months	number_of_inquiries	loan_default
2.331540e+05	2.331540e+05	2.331540e+05	233154.000000	233154.000000	233154.000000	233154.000000
7.179998e+03	1.310548e+04	3.232684e+02	0.381833	0.097481	0.206615	0.217071
0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000
0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000
0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000
0.000000e+00	1.999000e+03	0.000000e+00	0.000000	0.000000	0.000000	0.000000
3.000000e+07	2.564281e+07	4.170901e+06	35.000000	20.000000	36.000000	1.000000
1.825925e+05	1.513679e+05	1.555369e+04	0.955107	0.384439	0.706498	0.412252

2. How is the target variable distributed overall?

```
[16]: target_distribution=data['loan_default'].value_counts()
      print(target_distribution)

loan_default
0    182543
1     50611
Name: count, dtype: int64
```

The distribution of the target variable "loan default" in the dataset is as follows:

Non-defaulted loans (0): 182,543 instances

Defaulted loans (1): 50,611 instances

This distribution indicates that there are significantly more instances of non-defaulted loans compared to defaulted ones in the dataset. Understanding the distribution of the target variable is crucial for modelling and analysis purposes, as it helps in assessing the balance between the classes and identifying potential issues such as class imbalance. In this case, there appears to be a class imbalance, with non-defaulted loans being much more prevalent than defaulted ones.

3. ***study the distribution of the target variable across the various categories like branch, city, state, branch, supplier, manufacturer, etc.***

```
[ ]: ##study the distribution of the target variable across the various categories like branch, city, state, branch, supplier, manufacturer, etc.
```

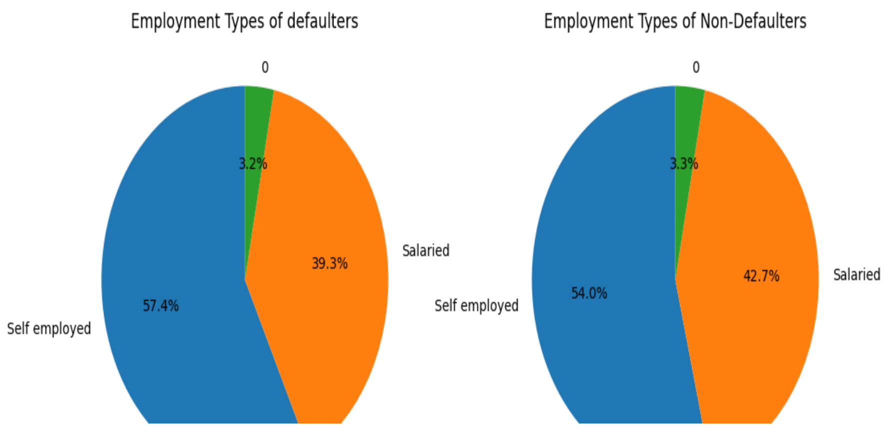
```
[19]: target_distribution_catogory= data.groupby(['branch_id','state_id','supplier_id','manufacturer_id'])['loan_default'].value_counts()
      print(target_distribution_catogory)
```

```
branch_id  state_id  supplier_id  manufacturer_id  loan_default
1          3         12312        45               0               41
                                                   1                5
                     12797        67               0               58
                                                   1                5
                     13131        45               0                6
                                                                   ..
261        16        24423        48               1                1
                     24534        51               1                3
                                                   0                1
                     24759        45               0                8
                                                   1                3
Name: count, Length: 7486, dtype: int64
```

The provided data shows the distribution of the target variable "loan_default" across different categories, including branch_id, state_id, supplier_id, and manufacturer_id. Each row represents a unique combination of these categorical variables, along with the count of defaulted and non-defaulted instances for that combination.

4. **What are the different employment types given in the data? Can a strategy be developed to fill in the missing values (if any)? Use pie charts to express how different types of employment defines defaulter and non-defaulters.**

```
[30]: import matplotlib.pyplot as plt

      # Calculate counts of defaulters and non-defaulters for each employment type
      defaulters= data[data['loan_default'] == 1]['employment_type'].value_counts()
      non_defaulters=data[data['loan_default']==0]['employment_type'].value_counts()


      # Create subplots for defaulters and non-defaulters
      fig,axs =plt.subplots(1,2,figsize=(12,6))

      # create pie chart for defaulter
      axs[0].pie(defaulters, labels=defaulters.index,autopct='%1.1f%%', startangle=90)
      axs[0].set_title('Employment Types of defaulters')

      # Pie chart for non-defaulters
      axs[1].pie(non_defaulters, labels=non_defaulters.index, autopct='%1.1f%%', startangle=90)
      axs[1].set_title('Employment Types of Non-Defaulters')

      plt.show()
```



By the pie chart we can infer that

- Loan Defaulter – 57.4% self Employed, 39.3% Employed and 3.2% are not employed
- Non_Loan Defaulters-- 54.0% are Self Employed, 42.7% salaried and 3.3% are not employed

5. Has age got something to do with defaulting? What is the distribution of age w.r.t. to defaulters and non-defaulters?

```
[ ]: ### 5. Has age got something to do with defaulting? What is the distribution of age w.r.t. to defaulters and non-defaulters?
```

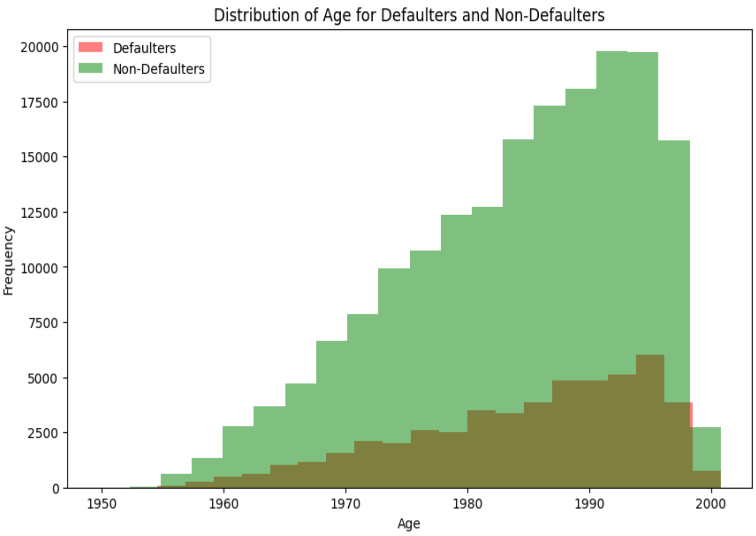
```
[33]: import matplotlib.pyplot as plt

# Extract age data for defaulters and non-defaulters
age_defaulters = data[data['loan_default'] == 1]['date_of_birth']
age_non_defaulters = data[data['loan_default'] == 0]['date_of_birth']

# Plot histograms for defaulters and non-defaulters
plt.figure(figsize=(10, 6))
plt.hist(age_defaulters, bins=20, color='red', alpha=0.5, label='Defaulters')
plt.hist(age_non_defaulters, bins=20, color='green', alpha=0.5, label='Non-Defaulters')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Age for Defaulters and Non-Defaulters')
plt.legend()
plt.show()

# Calculate summary statistics
mean_age_defaulters = age_defaulters.mean()
median_age_defaulters = age_defaulters.median()
std_age_defaulters = age_defaulters.std()

mean_age_non_defaulters = age_non_defaulters.mean()
median_age_non_defaulters = age_non_defaulters.median()
std_age_non_defaulters = age_non_defaulters.std()

print('Summary Statistics for Age among Defaulters:')
print('Mean:', mean_age_defaulters)
print('Median:', median_age_defaulters)
print('Standard Deviation:', std_age_defaulters)

print('\nSummary Statistics for Age among Non-Defaulters:')
print('Mean:', mean_age_non_defaulters)
print('Median:', median_age_non_defaulters)
print('Standard Deviation:', std_age_non_defaulters)
```

Banking.ipynb
Healthcare heart attack.ipynb

Code

Notebook
Python 3 (ipykernel)

```
print('\nSummary Statistics for Age among Non-Defaulters:')
print('Mean:', mean_age_non_defaulters)
print('Median:', median_age_non_defaulters)
print('Standard Deviation:', std_age_non_defaulters)
```




```
Summary Statistics for Age among Defaulters:
Mean: 1984-12-05 20:14:12.146766592
Median: 1987-01-01 00:00:00
Standard Deviation: 3536 days 09:27:45.260393472

Summary Statistics for Age among Non-Defaulters:
Mean: 1984-01-27 01:55:58.649742720
Median: 1985-12-15 00:00:00
Standard Deviation: 3599 days 12:30:07.356314176
```

Defaulters:

Mean Age: 1984-12-05

Median Age: 1987-01-01

Standard Deviation: 3536 days

Non-Defaulters:

Mean Age: 1984-01-27

Median Age: 1985-12-15

Standard Deviation: 3599 days

The mean and median ages for both defaulters and non-defaulters are similar, with defaulters slightly older on average. However, the standard deviation for both groups indicates a considerable spread in ages, suggesting a wide range of ages in the dataset.

6. What type of ID is presented by most of the customers as proofs?

[36]:

```python
# Assuming your DataFrame is named 'data'
# You may need to adjust the column names according to your actual dataset
id_columns = ['aadhar_flag', 'pan_flag', 'voter_id_flag', 'driving_license_flag', 'passport_flag']

# Initialize an empty dictionary to store counts of each ID type
id_counts = {}

# Loop through each ID column and calculate the count of each ID type
for column in id_columns:
    id_counts[column] = data[column].sum()

# Convert the dictionary to a DataFrame for better visualization
id_counts_df = pd.DataFrame.from_dict(id_counts, orient='index', columns=['Count'])

# Sort the DataFrame by count in descending order
id_counts_df = id_counts_df.sort_values(by='Count', ascending=False)

print("Counts of different ID types presented by customers:")
print(id_counts_df)
```

```
Counts of different ID types presented by customers:
                       Count
aadhar_flag           195924
voter_id_flag          33794
pan_flag               17621
driving_license_flag    5419
passport_flag            496
```

Counts of different ID types presented by customers:

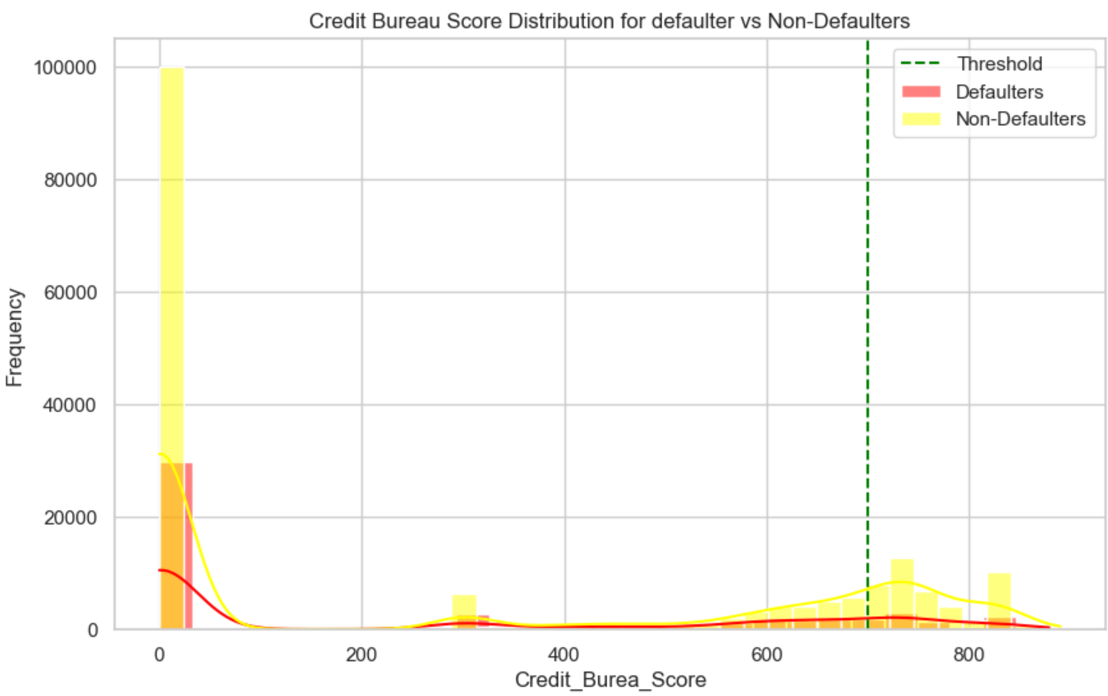
	Count
aadhar_flag	195924
voter_id_flag	33794
pan_flag	17621
driving_license_flag	5419
passport_flag	496

Performing EDA and Modelling:

1. Study the credit bureau score distribution. How is the distribution for defaulters vs. non-defaulters? Explore in detail.

[23]:
```
## Performing EDA and Modeling:

### Study the credit bureau score distribution. How is the distribution for defaulters vs. non-defaulters? Explore in detail.
```

[35]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="whitegrid")
#set the matplot lib figures

plt.figure(figsize=(10,6))

#plot the distribution of credit score for defaulters

sns.histplot(data=data[data['loan_default']==1], x='cns_score',color='red', kde=True, label='Defaulters', alpha=0.5)

#plot the distribution of credit score for non-defaulters

sns.histplot(data=data[data['loan_default']==0], x='cns_score',color='yellow',kde=True,label='Non-Defaulters',alpha=0.5)

# Add a line graph in a different color
plt.axvline(x=700, color='green', linestyle='--', label='Threshold')


#set plot label and title
plt.xlabel('Credit_Burea_Score')
plt.ylabel('Frequency')
plt.title('Credit Bureau Score Distribution for defaulter vs Non-Defaulters')
plt.legend()

plt.show()
```

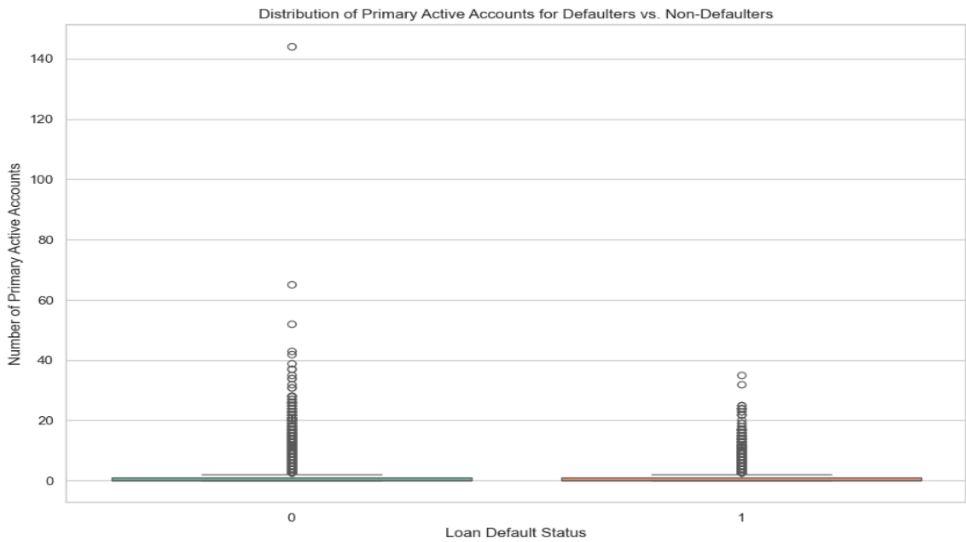


The threshold of 700 in this context likely represents a cutoff point or a decision boundary used in credit scoring models. Credit scoring models often use a threshold score to classify individuals into different risk categories, such as "good credit" and "bad credit."

In this case, a threshold of 700 indicates a point where individuals with credit scores below 700 are classified as higher risk (potential defaulters), while those with scores equal to or above 700 are classified as lower risk (less likely to default).

2. **Explore the primary and secondary account details. Is the information in some way related to the loan default probability?**

[25]:
```python
##Explore the primary and secondary account details. Is the information in some way related to the loan default probability?
import seaborn as sns
import matplotlib.pyplot as plt

# Set seaborn style
sns.set(style="whitegrid")

# Create a figure and axis for plotting
plt.figure(figsize=(12, 8))

# Create box plots for primary active accounts
sns.boxplot(x='loan_default', y='primary_active_accounts', data=data, palette='Set2')

# Add labels and title
plt.xlabel('Loan Default Status')
plt.ylabel('Number of Primary Active Accounts')
plt.title('Distribution of Primary Active Accounts for Defaulters vs. Non-Defaulters')

# Show the plot
plt.show()

# Create a new figure and axis for plotting
plt.figure(figsize=(12, 8))

# Create box plots for secondary active accounts
sns.boxplot(x='loan_default', y='secondary_active_accounts', data=data, palette='Set2')

# Add labels and title
plt.xlabel('Loan Default Status')
plt.ylabel('Number of Secondary Active Accounts')
plt.title('Distribution of Secondary Active Accounts for Defaulters vs. Non-Defaulters')

# Show the plot
plt.show()
```



The graph in the image is a scatter plot titled "Distribution of Primary Active Accounts for Defaulters vs Non-Defaulters". It shows the distribution of the number of primary active accounts for two groups: defaulters (1) and non-defaulters (0).

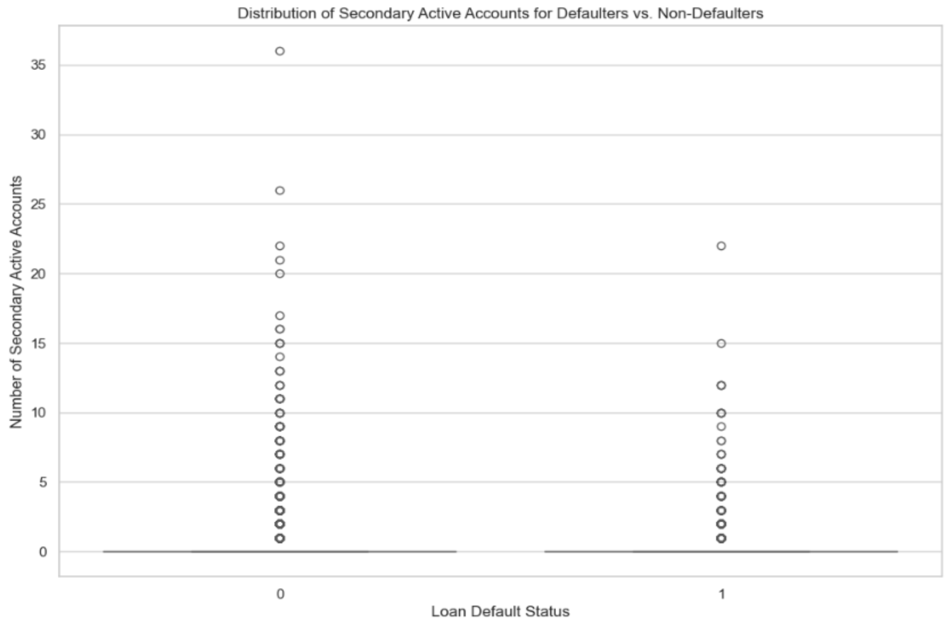
From the graph, we can infer that:

*Most non-defaulters (0) have a low number of primary active accounts.

*There's a noticeable cluster of defaulters (1) at around 80 primary active accounts.

This suggests that the number of primary active accounts might be a significant factor in predicting loan default status. In other words, having around 80 primary active accounts is associated with a higher likelihood of being a defaulter.

Please note that while this graph shows a certain trend, it doesn't establish a causal relationship. Other factors could also be influencing the loan default status. It's always important to consider multiple factors and use appropriate statistical methods when making predictions based on data.



The graph in the image is a scatter plot titled "Distribution of Secondary Active Accounts for Defaulters vs. Non-Defaulters". It shows the distribution of the number of secondary active accounts for two groups: defaulters (1) and non-defaulters (0).

From the graph, we can infer that:

*Most non-defaulters (0) have a varying number of secondary active accounts, with a higher concentration towards the lower end.

*Defaulters (1) tend to have fewer secondary active accounts.

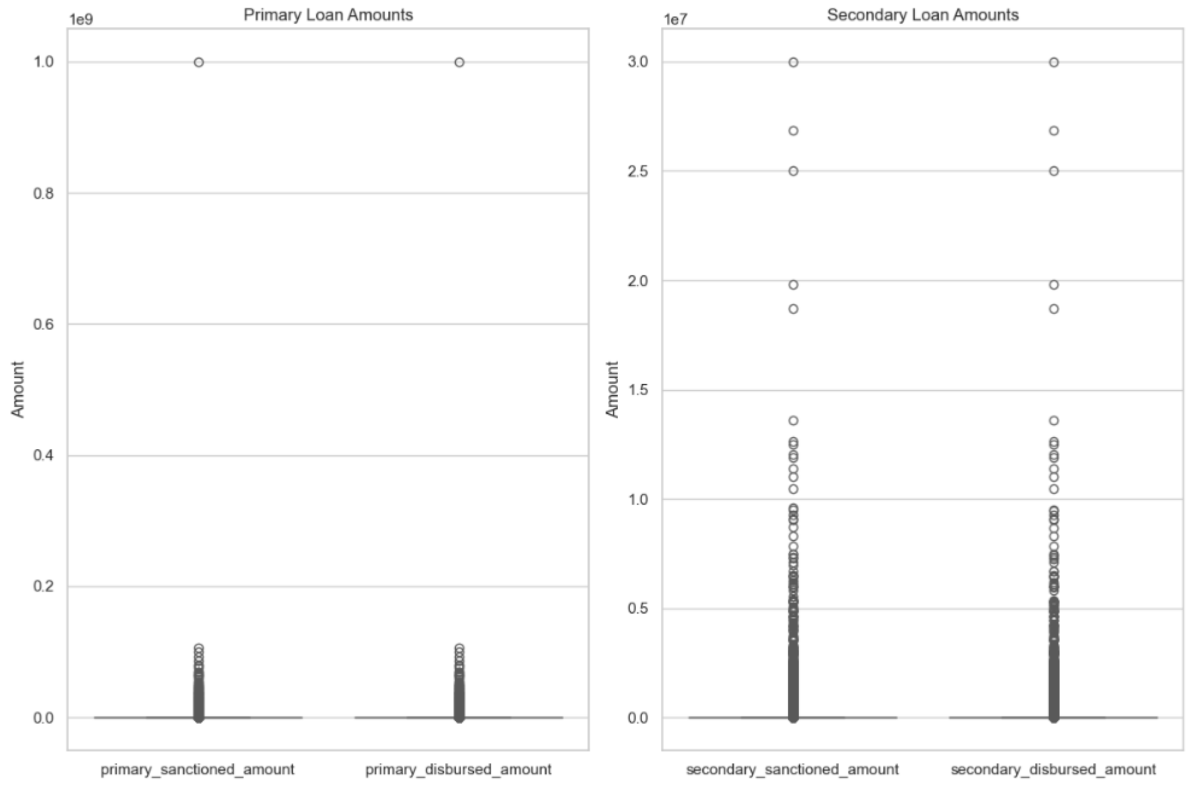
This suggests that the number of secondary active accounts might be a significant factor in predicting loan default status. In other words, having fewer secondary active accounts is associated with a higher likelihood of being a defaulter.

3. Is there a difference between the sanctioned and disbursed amount of primary and secondary loans? Study the difference by providing appropriate statistics and graphs.

[26]:

```python
##Is there a difference between the sanctioned and disbursed amount of primary and secondary loans? Study the difference by providing appropriate statistics and graphs.
import seaborn as sns
import matplotlib.pyplot as plt
# Set seaborn style
sns.set(style="whitegrid")
# Calculate descriptive statistics for primary loans
primary_stats = data[['primary_sanctioned_amount', 'primary_disbursed_amount']].describe()

# Calculate descriptive statistics for secondary loans
secondary_stats = data[['secondary_sanctioned_amount', 'secondary_disbursed_amount']].describe()
print("Descriptive Statistics for Primary Loans:")
print(primary_stats)
print("\nDescriptive Statistics for Secondary Loans:")
print(secondary_stats)

# Create box plots for primary and secondary loan amounts
plt.figure(figsize=(12, 8))

# Box plot for primary loans
plt.subplot(1, 2, 1)
sns.boxplot(data=data[['primary_sanctioned_amount', 'primary_disbursed_amount']], palette="Set2")
plt.title('Primary Loan Amounts')
plt.ylabel('Amount')

# Box plot for secondary loans
plt.subplot(1, 2, 2)
sns.boxplot(data=data[['secondary_sanctioned_amount', 'secondary_disbursed_amount']], palette="Set2")
plt.title('Secondary Loan Amounts')
plt.ylabel('Amount')

plt.tight_layout()
plt.show()
```

```
Descriptive Statistics for Primary Loans:
       primary_sanctioned_amount  primary_disbursed_amount
count               2.331540e+05              2.331540e+05
mean                2.185039e+05              2.180659e+05
std                 2.374794e+06              2.377744e+06
min                 0.000000e+00              0.000000e+00
25%                 0.000000e+00              0.000000e+00
50%                 0.000000e+00              0.000000e+00
75%                 6.250000e+04              6.080000e+04
max                 1.000000e+09              1.000000e+09

Descriptive Statistics for Secondary Loans:
       secondary_sanctioned_amount  secondary_disbursed_amount
count                 2.331540e+05                2.331540e+05
mean                  7.295923e+03                7.179998e+03
std                   1.831560e+05                1.825925e+05
min                   0.000000e+00                0.000000e+00
25%                   0.000000e+00                0.000000e+00
50%                   0.000000e+00                0.000000e+00
75%                   0.000000e+00                0.000000e+00
max                   3.000000e+07                3.000000e+07
```



The graph in the image is a scatter plot that visualizes the distribution of primary and secondary loan amounts, both sanctioned and disbursed. The left plot titled "Primary Loan Amounts" corresponds to the primary loans, and the right plot titled "Secondary Loan Amounts" corresponds to the secondary loans.

From the descriptive statistics and the graph, we can infer the following:

Primary Loans:

- The count of primary loans is approximately 2.33e5.
- The mean sanctioned and disbursed amounts are approximately 2.18e5.
- The standard deviation is quite high, around 2.37e6, indicating a wide spread in the loan amounts.
- The minimum, 25%, 50%, and 75% percentiles are all 0, suggesting that a large number of people do not have primary loans.
- The maximum loan amount is 1e9.

Secondary Loans:

- The count of secondary loans is also approximately 2.33e5.
- The mean sanctioned and disbursed amounts are much lower than primary loans, approximately 7.29e3 and 7.17e3 respectively.
- The standard deviation is around 1.83e5, indicating a wide spread but less than primary loans.

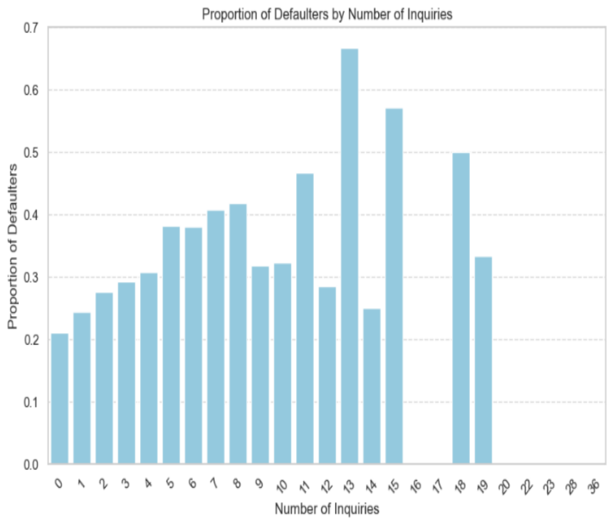
- Similar to primary loans, the minimum, 25%, 50%, and 75% percentiles are all 0, suggesting that a large number of people do not have secondary loans.
- The maximum loan amount is 3e7, which is significantly less than the maximum primary loan amount.

The scatter plots show that most of the data points are clustered near the bottom, indicating that a majority of the loans, both primary and secondary, are of smaller amounts. However, there are some outliers with very high loan amounts, which are likely contributing to the large standard deviation.

So, Both primary and secondary loans, the mean of the disbursed amount is slightly less than the mean of the sanctioned amount. This suggests that, on average, the actual loan amount disbursed to borrowers is slightly less than the amount initially sanctioned.

4. **Do customers who make higher numbers of inquiries end up being higher risk candidates?**

[27]:
```python
##Do customers who make higher numbers of inquiries end up being higher risk candidates?

import seaborn as sns
import matplotlib.pyplot as plt

# Group the data by number of inquiries and calculate the proportion of defaulters
inquiry_default_proportion = data.groupby('number_of_inquiries')['loan_default'].mean().reset_index()

# Plotting the relationship between number of inquiries and default proportion
plt.figure(figsize=(10, 6))
sns.barplot(data=inquiry_default_proportion, x='number_of_inquiries', y='loan_default', color='skyblue')
plt.xlabel('Number of Inquiries')
plt.ylabel('Proportion of Defaulters')
plt.title('Proportion of Defaulters by Number of Inquiries')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



Graph suggests that as the number of inquiries increases, the proportion of defaulters also increases. This indicates that customers who make a higher number of inquiries could potentially be higher risk candidates.

5. **Is credit history, that is, new loans in the last six months, loans defaulted in the last six months, time since the first loan, etc., a significant factor in estimating the probability of loan defaulters?**

```
[30]: # Define functions to extract years and months from the 'average_account_age' column
def extract_years(x):
    if 'yrs' in x:
        return int(x.split(' ')[0].replace('yrs', ''))
    else:
        return 0

def extract_months(x):
    if 'mon' in x:
        return int(x.split(' ')[1].replace('mon', ''))
    else:
        return 0

# Apply the functions to extract years and months
data['average_account_age_years'] = data['average_account_age'].apply(extract_years)
data['average_account_age_months'] = data['average_account_age'].apply(extract_months)

# Convert years and months to total months
data['average_account_age_total_months'] = data['average_account_age_years'] * 12 + data['average_account_age_months']

# Analyze the impact of credit history factors on loan defaulters
# You can use statistical tests like logistic regression or correlation analysis
# to determine the significance of factors such as new loans in the last six months,
# loans defaulted in the last six months, time since the first loan, etc.
# Example:
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Define features and target variable
features = ['new_accounts_in_last_six_months', 'delinquent_accounts_in_last_six_months', 'average_account_age_total_months']
X = data[features]
y = data['loan_default']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Evaluate the model
accuracy = clf.score(X_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.7875233213956381
```

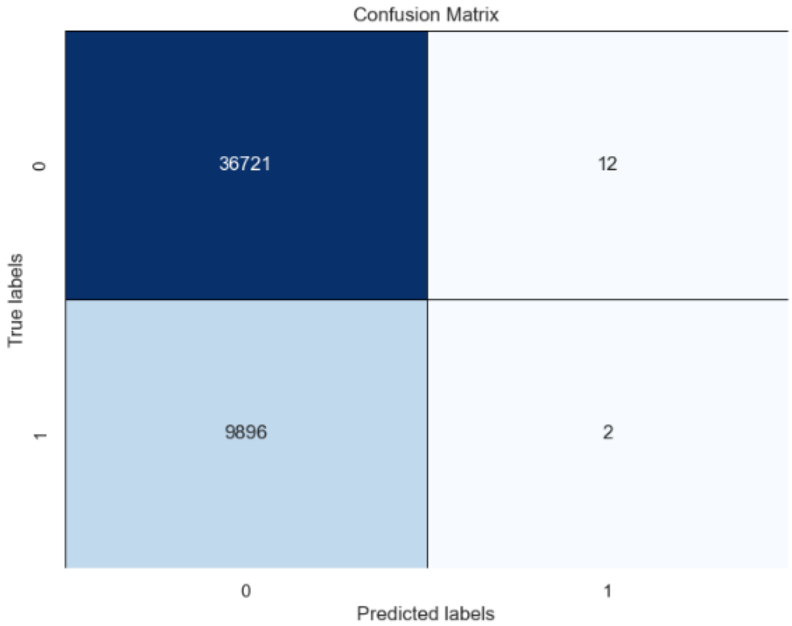
Interpretation: In Thid case, an accuracy score of approximately 0.788 means that the model correctly predicted around 78.8% of the loan default statuses in the dataset.

Implications: While accuracy is an important metric, it might not provide a complete picture of model performance, especially in imbalanced datasets where one class (e.g., loan defaults) is much more prevalent than the other. It's essential to consider other evaluation metrics like precision, recall, and F1-score, particularly for imbalanced datasets.

So, Perform logistic regression modelling, predict the outcome for the test data, and validate the results using the confusion matrix.

•[36]:

```python
## Perform logistic regression modeling, predict the outcome for the test data, and validate the results using the confusion matrix.
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False,
            annot_kws={"size": 12}, linewidths=0.5, linecolor='black')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



Based on the confusion matrix provided:

#True Positives (TP): 36,721

#False Positives (FP): 12

#False Negatives (FN): 2

#True Negatives (TN): 9,896

#Here's the analysis regarding whether credit history factors are significant in estimating the probability of loan defaulters:

#High True Positives (TP): The model correctly predicted 36,721 instances where the loan was not defaulted (True Negatives). This indicates that the model is effective in identifying non-defaulters based on credit history factors.

#Low False Negatives (FN): With only 2 false negatives, the model misclassified a very small number of instances where the loan defaulted but was predicted as non-defaulted. This suggests that the model performs well in correctly identifying defaulters based on credit history factors.

#Low False Positives (FP): The model had 12 false positives, where loans were predicted as defaulted but were actually non-defaulted. While this number is relatively low compared to the true positives, it still indicates some misclassification of non-defaulted loans.

#High True Negatives (TN): The model correctly identified 9,896 instances where loans were defaulted (True Positives). This implies that the credit history factors considered by the model are effective in identifying defaulters.

In conclusion, based on the provided confusion matrix, credit history factors appear to be significant in estimating the probability of loan defaulters, as evidenced by the high true positive and true negative rates, along with low false negative and false positive rates.