Project Scope

Objectives

To reduce financial losses from vehicle loan defaults, this project will analyze key factors that lead to defaults and develop a model to predict potential defaulters.

Key Deliverables

- Exploratory Data Analysis (EDA): Understanding the characteristics of the data, treating missing values, and handling outliers.
- Feature Selection & Importance Analysis: Identifying the primary factors contributing to loan defaults.
- Model Development: Using logistic regression to build a predictive model for identifying potential defaulters.
- Evaluation: Assessing the model's accuracy.

Dataset Overview:

The dataset comprises 41 attributes, capturing various customer details, loan specifics, and historical financial behaviors, serving as a foundation for our analysis.

This project aims to provide financial institutions with a predictive tool and insights to enhance their loan decision-making process.

Preliminary Data Analysis

1.Import Libraries

In [40]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from pyproj import Proj, transform
from datetime import datetime
from IPython.display import display

import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

from IPython.core.display import display, HTML
```

2.Read datasets

In [19]:
```
loan =pd.read_excel('../../02 Data/Original Data/internal/data.xlsx')
```

3.Quickly explore the data

In [20]:
```
display(loan.head())
number_of_rows = loan.shape[0]
print ( 'This dataset includes {} observations and {} columns.'.format(loan.shape[0] ,
```

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pi
0	420825	50578	58400	89.55	67	22807	45	
1	417566	53278	61360	89.63	67	22807	45	
2	539055	52378	60300	88.39	67	22807	45	
3	529269	46349	61500	76.42	67	22807	45	
4	563215	43594	78256	57.50	67	22744	86	

5 rows × 41 columns

```
This dataset includes 233154 observations and 41 columns.
```

In [21]:
```
loan.info();
print("\n It seems that we have missing value for employee type.")
print(" Some variables need to be renamed; their current names are not compatible with
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 233154 entries, 0 to 233153
Data columns (total 41 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   UniqueID                              233154 non-null  int64
 1   disbursed_amount                      233154 non-null  int64
 2   asset_cost                            233154 non-null  int64
 3   ltv                                   233154 non-null  float64
 4   branch_id                             233154 non-null  int64
 5   supplier_id                           233154 non-null  int64
 6   manufacturer_id                       233154 non-null  int64
 7   Current_pincode_ID                    233154 non-null  int64
 8   Date.of.Birth                         233154 non-null  datetime64[ns]
 9   Employment.Type                       225493 non-null  object
 10  DisbursalDate                         233154 non-null  datetime64[ns]
 11  State_ID                              233154 non-null  int64
 12  Employee_code_ID                      233154 non-null  int64
 13  MobileNo_Avl_Flag                     233154 non-null  int64
 14  Aadhar_flag                           233154 non-null  int64
 15  PAN_flag                              233154 non-null  int64
 16  VoterID_flag                          233154 non-null  int64
 17  Driving_flag                          233154 non-null  int64
 18  Passport_flag                         233154 non-null  int64
 19  PERFORM_CNS.SCORE                     233154 non-null  int64
 20  PERFORM_CNS.SCORE.DESCRIPTION         233154 non-null  object
 21  PRI.NO.OF.ACCTS                       233154 non-null  int64
 22  PRI.ACTIVE.ACCTS                      233154 non-null  int64
 23  PRI.OVERDUE.ACCTS                     233154 non-null  int64
 24  PRI.CURRENT.BALANCE                   233154 non-null  int64
 25  PRI.SANCTIONED.AMOUNT                 233154 non-null  int64
 26  PRI.DISBURSED.AMOUNT                  233154 non-null  int64
 27  SEC.NO.OF.ACCTS                       233154 non-null  int64
 28  SEC.ACTIVE.ACCTS                      233154 non-null  int64
 29  SEC.OVERDUE.ACCTS                     233154 non-null  int64
 30  SEC.CURRENT.BALANCE                   233154 non-null  int64
 31  SEC.SANCTIONED.AMOUNT                 233154 non-null  int64
 32  SEC.DISBURSED.AMOUNT                  233154 non-null  int64
 33  PRIMARY.INSTAL.AMT                    233154 non-null  int64
 34  SEC.INSTAL.AMT                        233154 non-null  int64
 35  NEW.ACCTS.IN.LAST.SIX.MONTHS          233154 non-null  int64
 36  DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS   233154 non-null  int64
 37  AVERAGE.ACCT.AGE                      233154 non-null  object
 38  CREDIT.HISTORY.LENGTH                 233154 non-null  object
 39  NO.OF_INQUIRIES                       233154 non-null  int64
 40  loan_default                          233154 non-null  int64
dtypes: datetime64[ns](2), float64(1), int64(34), object(4)
memory usage: 72.9+ MB

 It seems that we have missing value for employee type.
 Some variables need to be renamed; their current names are not compatible with Pytho
n conventions.
```

4.Data cleaning

-Renaming variables, replaceing "." by "_"

In [22]:
```
loan.columns  = loan.columns.str.replace(".","_") ;
loan.columns
```

```
C:\Users\M\AppData\Local\Temp\ipykernel_29720\458693661.py:1: FutureWarning: The defa
ult value of regex will change from True to False in a future version. In addition, s
ingle character regular expressions will *not* be treated as literal strings when reg
ex=True.
  loan.columns  = loan.columns.str.replace(".","_") ;
```

Out[22]:
```
Index(['UniqueID', 'disbursed_amount', 'asset_cost', 'ltv', 'branch_id',
       'supplier_id', 'manufacturer_id', 'Current_pincode_ID', 'Date_of_Birth',
       'Employment_Type', 'DisbursalDate', 'State_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_OF_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', 'SEC_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'],
      dtype='object')
```

In [23]:
```
loan.Employee_code_ID.unique().shape
```

Out[23]:
```
(3270,)
```

-Finding and Dealing with missing vlaues

In [24]:
```
loan.isna().sum()
```

Out[24]:
```
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
SEC_INSTAL_AMT                        0
NEW_ACCTS_IN_LAST_SIX_MONTHS          0
DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS   0
AVERAGE_ACCT_AGE                      0
CREDIT_HISTORY_LENGTH                 0
NO_OF_INQUIRIES                       0
loan_default                          0
dtype: int64
```

Missing value treatment for Employement_Type

Let's check if we could find Employement_Type by other columns like "asset_cost.

In [25]:
```
loan.Employment_Type.unique()
```

Out[25]:
```
array(['Salaried', 'Self employed', nan], dtype=object)
```

In [26]:
```
salaried_mean = loan[loan.Employment_Type == 'Salaried'].asset_cost.mean()
self_employed_mean = loan[loan.Employment_Type == 'Self employed'].asset_cost.mean()
print("Asset_cost's average for Salaried EmploymentType : {} ". format(salaried_mean))
print("Aset_cost's average for Self Employed EmploymentType : {} ". format(self_employ
print("There is no significant difference in the averages between the two groups. Ther
```

```
Asset_cost's average for Salaried EmploymentType : 74395.27703406978
Aset_cost's average for Self Employed EmploymentType : 76578.66417518706
There is no significant difference in the averages between the two groups. Therefore,
this variable may not be helpful for imputing Employment_type.
```

In [96]:
```
#fig, axes = plt.subplots(1, 2, figsize=(12, 6))
#sns.boxplot(data = loan[loan.Employment_Type =='Self employed'] , y = 'asset_cost' ,
#sns.boxplot(data = loan[loan.Employment_Type =='Salaried'] , y = 'asset_cost' , ax=ax

#print("There is no significant difference, so we can not use asset_cost for Employmen
```

I used mode for missing value imputation

In [27]:
```
loan.Employment_Type.fillna(loan.Employment_Type.mode()[0],inplace=True)
```

In [28]:
```
missing_values = loan.isna().sum()
columns_with_missing_values = missing_values[missing_values > 0]
print(columns_with_missing_values)
```

```
Series([], dtype: int64)
```

-Finding and Removing Duplicates

No duplicates found

In [29]:
```
loan[loan.duplicated()]
```

Out[29]:

UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pinco

0 rows × 41 columns

-Finding Inconsistant texts and typos -Fixing Inconsistant texts and typos

-converting AVERAGE_ACCT_AGE and CREDIT_HISTORY_LENGTH to months

In [30]:
```
# This function converts "xyrs ymon" to the number of months
def convert_to_month(duration) :
    # extract year and month
    years = int(duration.split()[0].replace('yrs','').replace('years',''))
    months = int(duration.split()[1].replace('mon' ,'').replace('months',''))
    # convert to months and return
    return ( years * 12  + months )
# Apply the function to those columns
loan['AVERAGE_ACCT_AGE'] = loan['AVERAGE_ACCT_AGE'].apply(convert_to_month)
loan['CREDIT_HISTORY_LENGTH'] = loan['CREDIT_HISTORY_LENGTH'].apply(convert_to_month)
#loan[(loan.AVERAGE_ACCT_AGE != 0 ) | (loan.CREDIT_HISTORY_LENGTH != 0)]
```

-converting Date_of_Birth to Age and DisbursalDate to Disbursal_time

In [31]:
```
# Convert the "Date_of_Birth" column to datetime
loan['Date_of_Birth'] = pd.to_datetime(loan['Date_of_Birth'])

# Calculate the age based on the current date
current_date = datetime.now()
loan['Age'] = (current_date - loan['Date_of_Birth']).astype('<m8[Y]')
loan['DisbursalTime'] = (current_date - loan['DisbursalDate']).astype('<m8[Y]')

# Drop Dat_of_Birth column
#loan = loan.drop('Date_of_Birth' , axis = 1)
#loan = loan.drop('DisbursalDate' , axis = 1)
```

In [32]:
```
loan.dtypes
```

Out[32]:
```
UniqueID                                int64
disbursed_amount                        int64
asset_cost                              int64
ltv                                   float64
branch_id                               int64
supplier_id                             int64
manufacturer_id                         int64
Current_pincode_ID                      int64
Date_of_Birth                  datetime64[ns]
Employment_Type                        object
DisbursalDate                  datetime64[ns]
State_ID                                int64
Employee_code_ID                        int64
MobileNo_Avl_Flag                       int64
Aadhar_flag                             int64
PAN_flag                                int64
VoterID_flag                            int64
Driving_flag                            int64
Passport_flag                           int64
PERFORM_CNS_SCORE                       int64
PERFORM_CNS_SCORE_DESCRIPTION          object
PRI_NO_OF_ACCTS                         int64
PRI_ACTIVE_ACCTS                        int64
PRI_OVERDUE_ACCTS                       int64
PRI_CURRENT_BALANCE                     int64
PRI_SANCTIONED_AMOUNT                   int64
PRI_DISBURSED_AMOUNT                    int64
SEC_NO_OF_ACCTS                         int64
SEC_ACTIVE_ACCTS                        int64
SEC_OVERDUE_ACCTS                       int64
SEC_CURRENT_BALANCE                     int64
SEC_SANCTIONED_AMOUNT                   int64
SEC_DISBURSED_AMOUNT                    int64
PRIMARY_INSTAL_AMT                      int64
SEC_INSTAL_AMT                          int64
NEW_ACCTS_IN_LAST_SIX_MONTHS            int64
DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS     int64
AVERAGE_ACCT_AGE                        int64
CREDIT_HISTORY_LENGTH                   int64
NO_OF_INQUIRIES                         int64
loan_default                            int64
Age                                   float64
DisbursalTime                         float64
dtype: object
```

-assign predefined risk categories to values in the 'PERFORM_CNS_SCORE_DESCRIPTION' column, helping to categorize and analyze the data based on specific risk level

In [33]:
```
loan.PERFORM_CNS_SCORE_DESCRIPTION.value_counts()
```

Out[33]:
```
No Bureau History Available                                   116950
C-Very Low Risk                                                16045
A-Very Low Risk                                                14124
D-Very Low Risk                                                11358
B-Very Low Risk                                                 9201
M-Very High Risk                                                8776
F-Low Risk                                                      8485
K-High Risk                                                     8277
H-Medium Risk                                                   6855
E-Low Risk                                                      5821
I-Medium Risk                                                   5557
G-Low Risk                                                      3988
Not Scored: Sufficient History Not Available                    3765
J-High Risk                                                     3748
Not Scored: Not Enough Info available on the customer           3672
Not Scored: No Activity seen on the customer (Inactive)         2885
Not Scored: No Updates available in last 36 months              1534
L-Very High Risk                                                1134
Not Scored: Only a Guarantor                                     976
Not Scored: More than 50 active Accounts found                     3
Name: PERFORM_CNS_SCORE_DESCRIPTION, dtype: int64
```

In [34]:
```
PERFORM_CNS_SCORE_DESCRIPTION_dict = {
                                        'C-Very Low Risk'  :  'low'     ,
                                        'A-Very Low Risk'  :  'low'     ,
                                        'D-Very Low Risk'  :  'low'     ,
                                        'B-Very Low Risk'  :  'low'     ,
                                        'M-Very High Risk' :  'high'    ,
                                        'F-Low Risk'       :  'low'     ,
                                        'K-High Risk'      :  'high'    ,
                                        'H-Medium Risk'    :  'medium'  ,
                                        'E-Low Risk'       :  'low'     ,
                                        'I-Medium Risk'    :  'medium'  ,
                                        'G-Low Risk'       :  'low'     ,
                                        'J-High Risk'      :  'high'    ,
                                        'L-Very High Risk' :  'high'    ,
                                        'No Bureau History Available'
                                        'Not Scored: Sufficient History Not Available'
                                        'Not Scored: Not Enough Info available on the
                                        'Not Scored: No Activity seen on the customer
                                        'Not Scored: No Updates available in last 36 m
                                        'Not Scored: Only a Guarantor'
                                        'Not Scored: More than 50 active Accounts foun

}

loan.PERFORM_CNS_SCORE_DESCRIPTION = loan.PERFORM_CNS_SCORE_DESCRIPTION.map(PERFORM_CN
loan.PERFORM_CNS_SCORE_DESCRIPTION.value_counts()
```

Out[34]:
```
not_available    116950
low               69022
high              21935
not_scored        12835
medium            12412
Name: PERFORM_CNS_SCORE_DESCRIPTION, dtype: int64
```

-Create year, month, and day column for DisbursalDate

In [35]:
```
#loan['DisbursalDate_Year'] = pd.to_datetime(loan.DisbursalDate).dt.year
#loan['DisbursalDate_Month'] = pd.to_datetime(loan.DisbursalDate).dt.month
#loan['DisbursalDate_Day'] = pd.to_datetime(loan.DisbursalDate).dt.day
```

-Create Age group column

In [36]:
```
# Define the bin edges and labels
bin_edges = [0, 18, 30, 45, 60, 100]  # Adjust the age ranges as needed
bin_labels = ['0-18', '19-30', '31-45', '46-60', '61+']

# Create a new column 'Age_Group' with the bin labels
loan['Age_Group'] = pd.cut(loan['Age'], bins=bin_edges, labels=bin_labels)

# Now, 'Age_Group' will contain the age bins for each row
```

-Finding and Dealing with outliers

In [37]:
```
loan.describe()
loan_pri_sec_acc = ['PRI_NO_OF_ACCTS' , 'PRI_ACTIVE_ACCTS' , 'PRI_OVERDUE_ACCTS' , 'PR
                    'PRI_SANCTIONED_AMOUNT','PRI_DISBURSED_AMOUNT' ,'SEC_NO_OF_ACCTS',
                    'SEC_OVERDUE_ACCTS','SEC_CURRENT_BALANCE','SEC_SANCTIONED_AMOUNT',
                    'SEC_DISBURSED_AMOUNT','PRIMARY_INSTAL_AMT','SEC_INSTAL_AMT']
other_columns= ['disbursed_amount','asset_cost','ltv','PERFORM_CNS_SCORE','NEW_ACCTS_I
                    'DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS','AVERAGE_ACCT_AGE','CREDIT_H
                    'NO_OF_INQUIRIES','Age']

loan.iloc[:,21:34].describe()
```

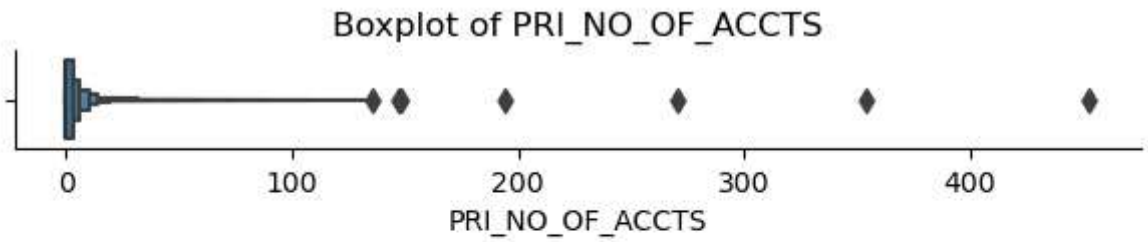
Out[37]:

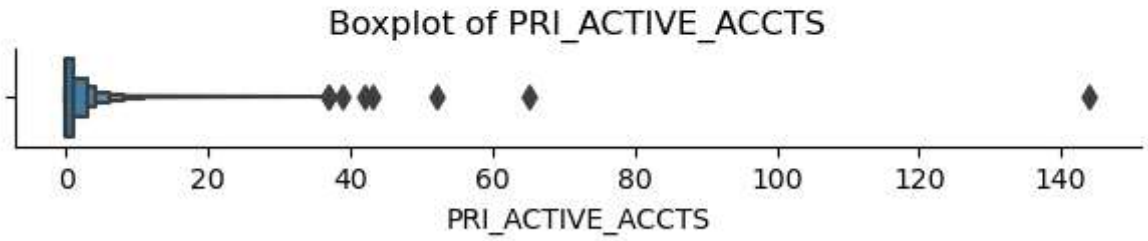
	PRI_NO_OF_ACCTS	PRI_ACTIVE_ACCTS	PRI_OVERDUE_ACCTS	PRI_CURRENT_BALANCE	PRI_SAN
count	233154.000000	233154.000000	233154.000000	2.331540e+05	
mean	2.440636	1.039896	0.156549	1.659001e+05	
std	5.217233	1.941496	0.548787	9.422736e+05	
min	0.000000	0.000000	0.000000	-6.678296e+06	
25%	0.000000	0.000000	0.000000	0.000000e+00	
50%	0.000000	0.000000	0.000000	0.000000e+00	
75%	3.000000	1.000000	0.000000	3.500650e+04	
max	453.000000	144.000000	25.000000	9.652492e+07	

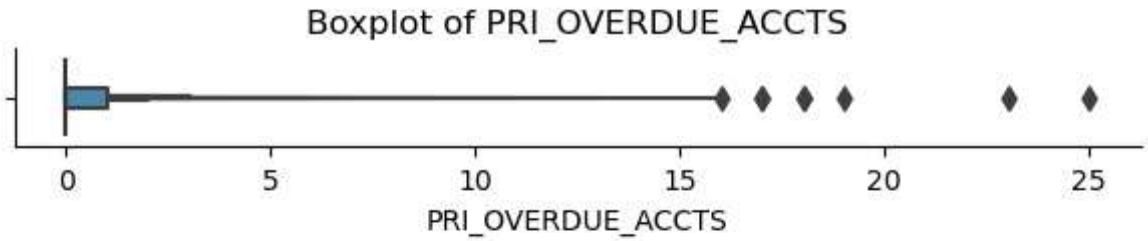
First I chech box plot for these numerical columns: disbursed_amount,asset_cost,ltv,NEW_ACCTS_IN_LAST_SIX_MONTHS,DELINQUEN AVERAGE_ACCT_AGE,CREDIT_HISTORY_LENGTH,NO_OF_INQUIRIES,Age

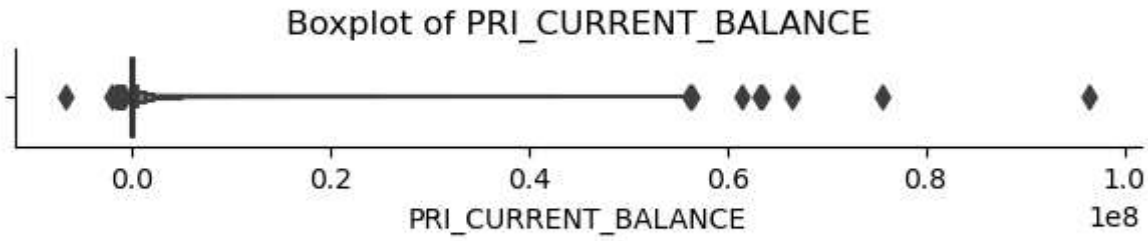
In [41]:
```
columns =['disbursed_amount','asset_cost','ltv','NEW_ACCTS_IN_LAST_SIX_MONTHS','DELINQ
          'AVERAGE_ACCT_AGE','CREDIT_HISTORY_LENGTH','NO_OF_INQUIRIES','Age',]
#for col in loan.columns:
#       if pd.api.types.is_numeric_dtype(loan[col]):
```

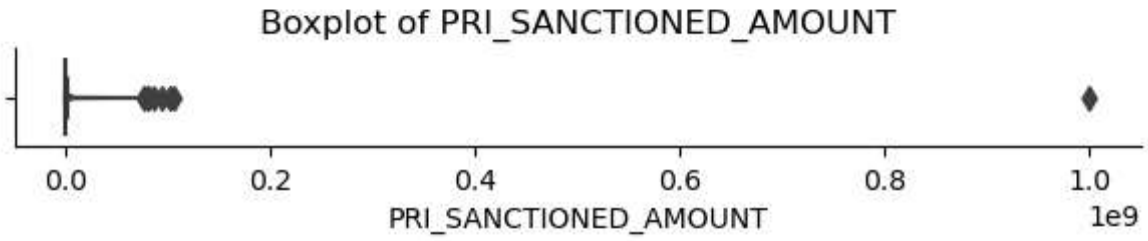
```
#             columns.append(col)
fig,ax = plt.subplots(len(loan_pri_sec_acc),layout = "tight" , figsize=(6, 18))
plt.subplots_adjust(hspace=4)  # Adjust the vertical spacing
#print(columns)
for col ,ax in zip(loan_pri_sec_acc,ax.flat):
    sns.boxenplot(data=loan,x=col,ax=ax) ;
    sns.despine();
    ax.set_title("Boxplot of " + col)
    # Use this command to enable the scrolling for outputs in the notebook
display(HTML("<style>div.output_scroll { height: auto; }</style>"))
```

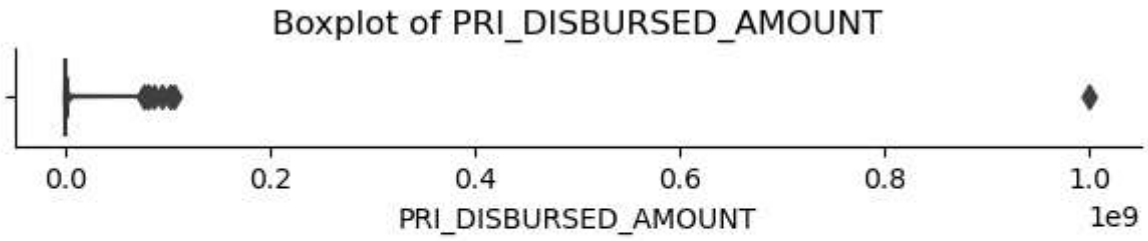

Boxplot of PRI_NO_OF_ACCTS
0
100
200
300
400
PRI_NO_OF_ACCTS

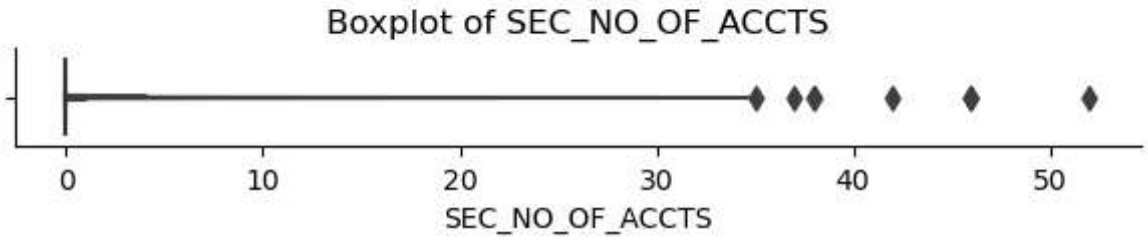

Boxplot of PRI_ACTIVE_ACCTS
0
20
40
60
80
100
120
140
PRI_ACTIVE_ACCTS


Boxplot of PRI_OVERDUE_ACCTS
0
5
10
15
20
25
PRI_OVERDUE_ACCTS

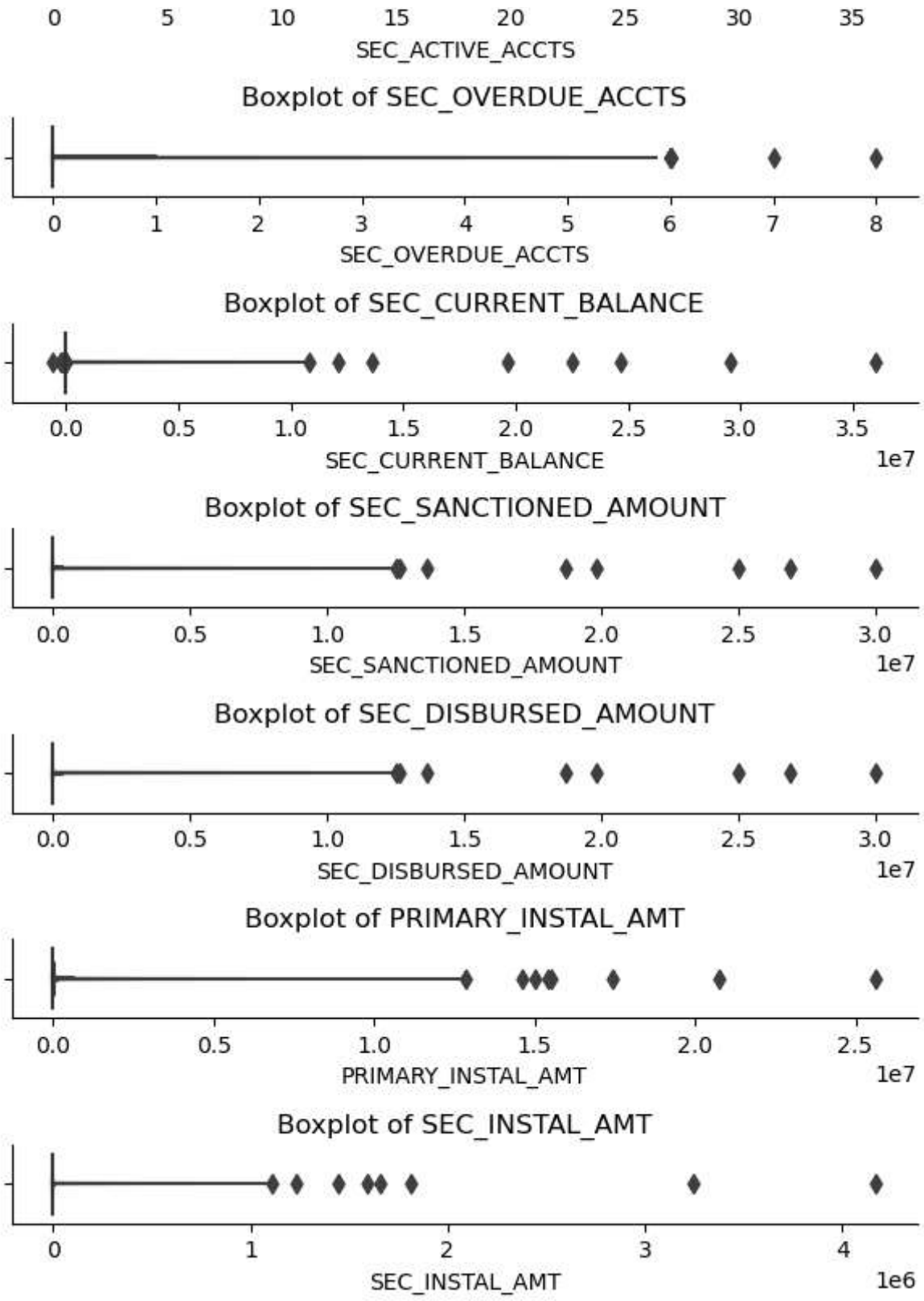

Boxplot of PRI_CURRENT_BALANCE
0.0
0.2
0.4
0.6
0.8
1.0
PRI_CURRENT_BALANCE
1e8


Boxplot of PRI_SANCTIONED_AMOUNT
0.0
0.2
0.4
0.6
0.8
1.0
PRI_SANCTIONED_AMOUNT
1e9


Boxplot of PRI_DISBURSED_AMOUNT
0.0
0.2
0.4
0.6
0.8
1.0
PRI_DISBURSED_AMOUNT
1e9


Boxplot of SEC_NO_OF_ACCTS
0
10
20
30
40
50
SEC_NO_OF_ACCTS

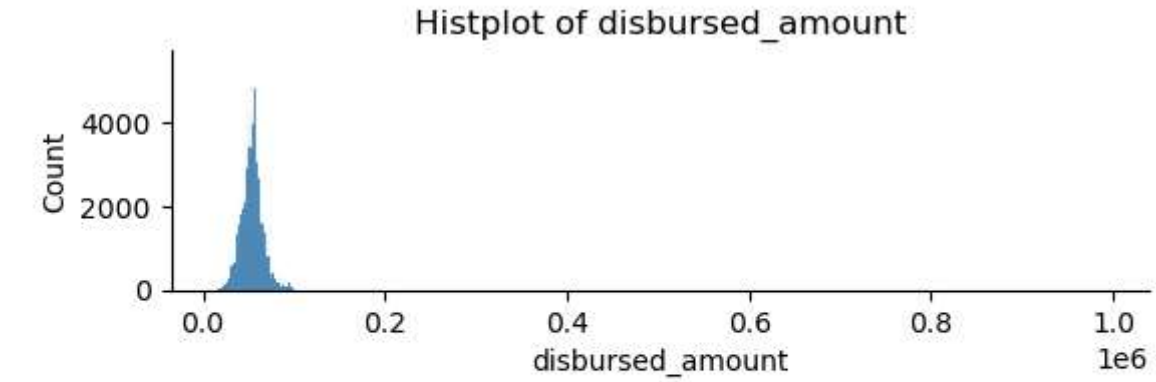

Boxplot of SEC_ACTIVE_ACCTS

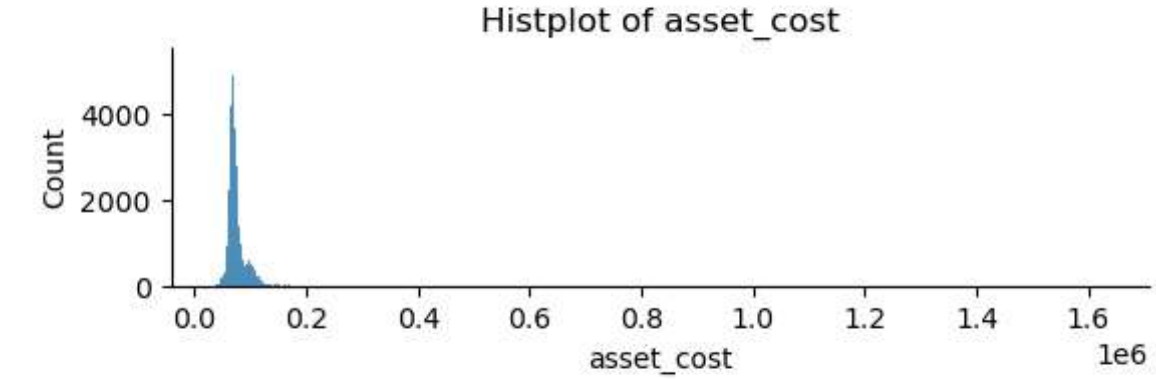


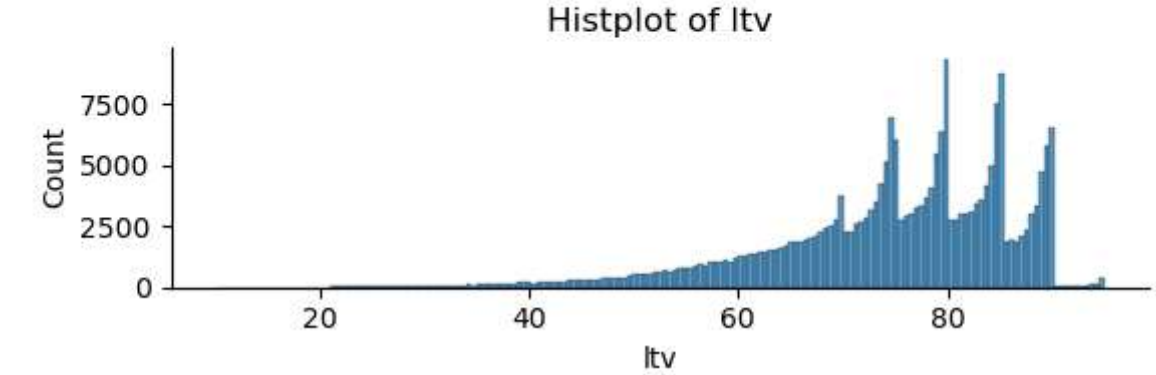
Let's check the histogram

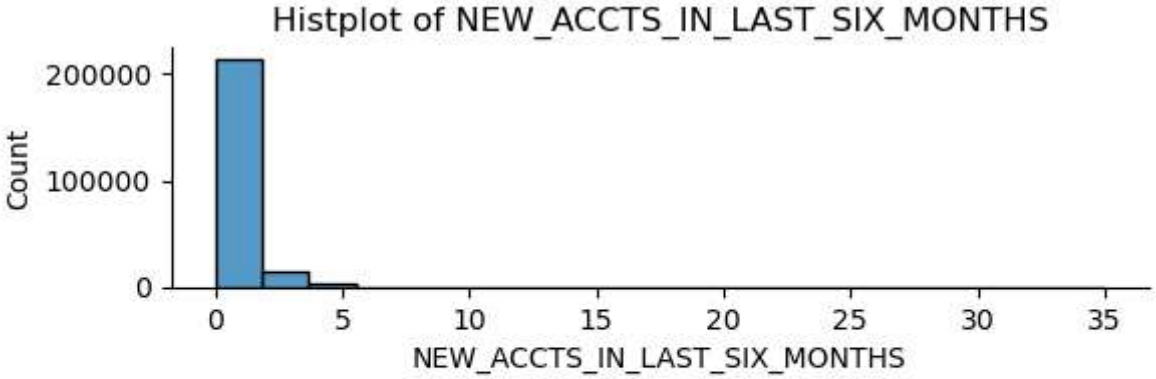
In [42]:
```
#columns =['disbursed_amount','asset_cost','ltv','NEW_ACCTS_IN_LAST_SIX_MONTHS','DELIN
#          'AVERAGE_ACCT_AGE','CREDIT_HISTORY_LENGTH','NO_OF_INQUIRIES','Age',]
#for col in loan.columns:
#       if pd.api.types.is_numeric_dtype(loan[col]):
#            columns.append(col)
fig,ax = plt.subplots(len(columns),layout = "tight" , figsize=(6, 18))
```

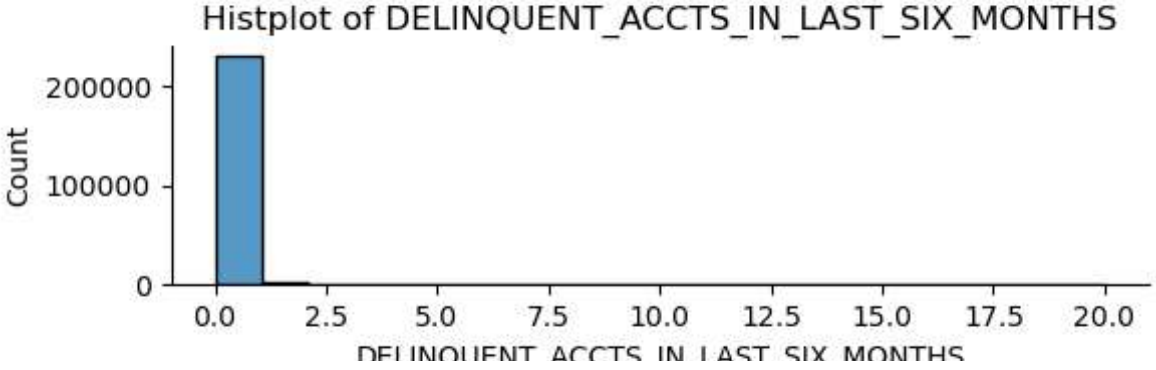
```
plt.subplots_adjust(hspace=1)  # Adjust the vertical spacing
#print(columns)
for col ,ax in zip(columns,ax.flat):
    sns.histplot(data=loan,x=col,ax=ax) ;
    sns.despine();
    ax.set_title("Histplot of " + col)
```

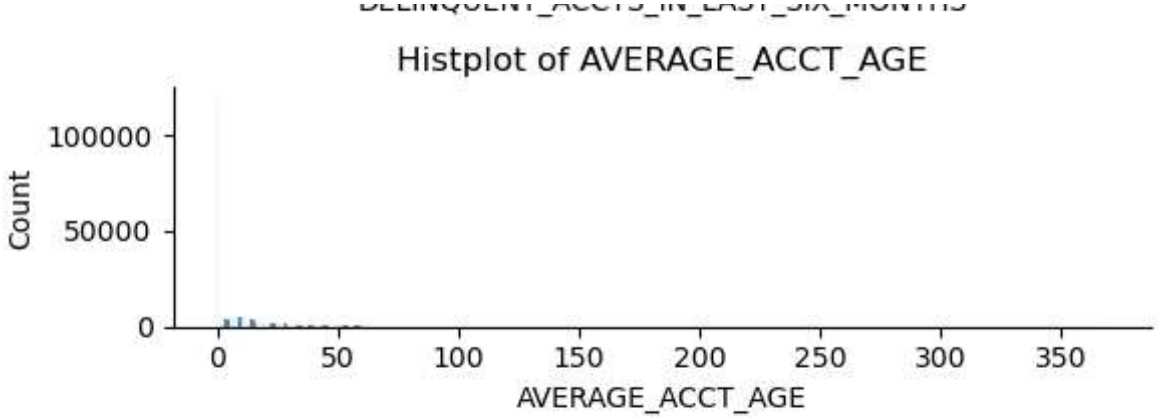

Histplot of disbursed_amount
Count
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
disbursed_amount
1e6

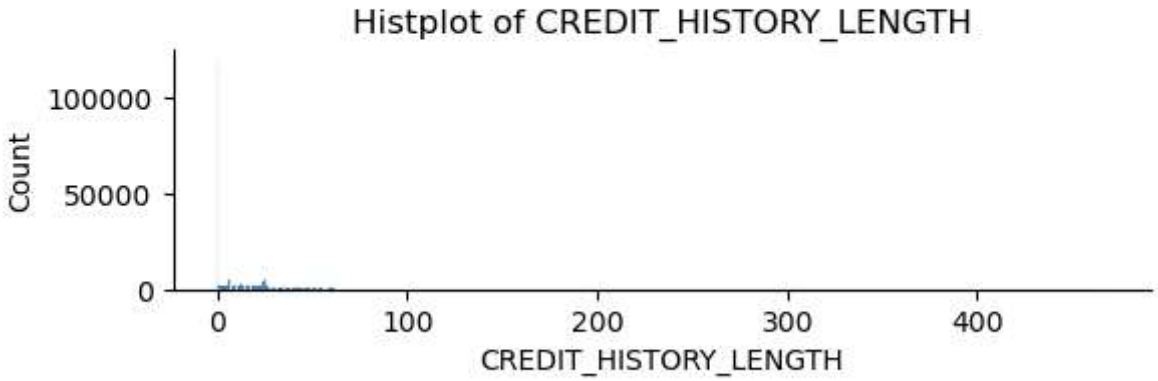

Histplot of asset_cost
Count
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
asset_cost
1e6

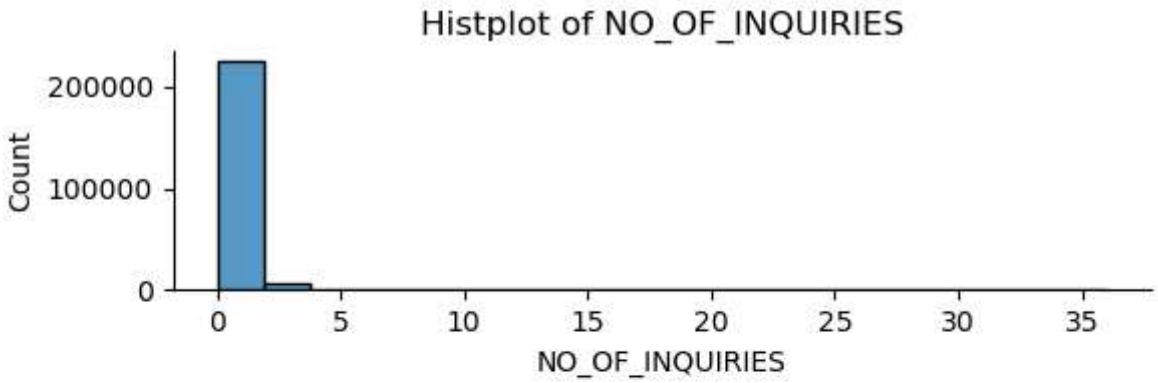

Histplot of ltv
Count
7500
5000
2500
0
20
40
60
80
ltv

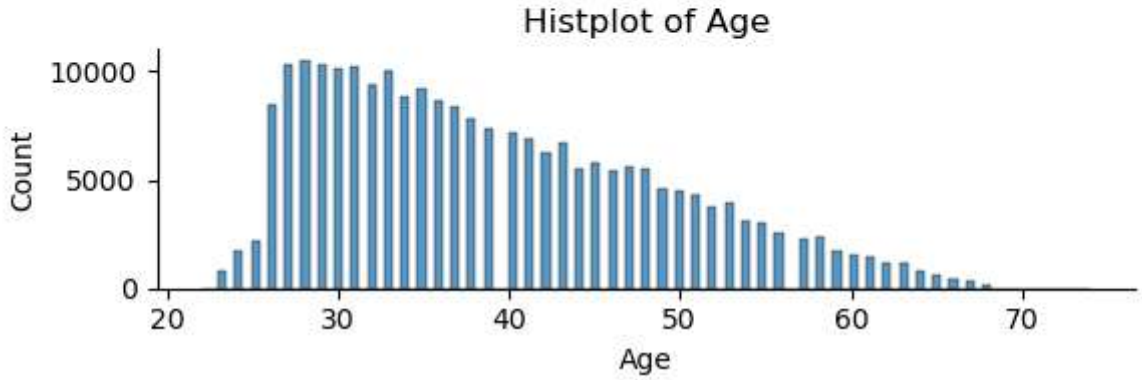

Histplot of NEW_ACCTS_IN_LAST_SIX_MONTHS
Count
200000
100000
0
0
5
10
15
20
25
30
35
NEW_ACCTS_IN_LAST_SIX_MONTHS


Histplot of DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS
Count
200000
100000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0









In [45]:
```
loan_pri_sec_acc = ['PRI_NO_OF_ACCTS' , 'PRI_ACTIVE_ACCTS' , 'PRI_OVERDUE_ACCTS' , 'PR
                    'PRI_SANCTIONED_AMOUNT','PRI_DISBURSED_AMOUNT' ,'SEC_NO_OF_ACCTS',
                    'SEC_OVERDUE_ACCTS','SEC_CURRENT_BALANCE','SEC_SANCTIONED_AMOUNT',
                    'SEC_DISBURSED_AMOUNT','PRIMARY_INSTAL_AMT','SEC_INSTAL_AMT']
other_columns= ['disbursed_amount','asset_cost','ltv','PERFORM_CNS_SCORE','NEW_ACCTS_I
                    'DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS','AVERAGE_ACCT_AGE','CREDIT_H
                    'NO_OF_INQUIRIES','Age']
```

```
loan.iloc[:,21:34].describe()
```

Out[45]:

	PRI_NO_OF_ACCTS	PRI_ACTIVE_ACCTS	PRI_OVERDUE_ACCTS	PRI_CURRENT_BALANCE	PRI_SAN
count	233154.000000	233154.000000	233154.000000	2.331540e+05	
mean	2.440636	1.039896	0.156549	1.659001e+05	
std	5.217233	1.941496	0.548787	9.422736e+05	
min	0.000000	0.000000	0.000000	-6.678296e+06	
25%	0.000000	0.000000	0.000000	0.000000e+00	
50%	0.000000	0.000000	0.000000	0.000000e+00	
75%	3.000000	1.000000	0.000000	3.500650e+04	
max	453.000000	144.000000	25.000000	9.652492e+07	

First I start with Primary, Secondary account columns

In [47]:

```
outlier_boundaries = {}
number_of_rows = loan.shape[0]

for col in loan_pri_sec_acc:
    Q25, Q50, Q75 = np.percentile(loan[col] , (25,50,75))
    IQR = Q75 - Q25
    lower_bound = Q25 - 1.5 * IQR
    upper_bound = Q75 + 1.5 * IQR
    outlier_boundaries[col] = (lower_bound, upper_bound)

    number_of_outliers =  len(  loan[ (loan[col]> upper_bound) | (loan[col]<lower_bour
    percentage_of_outliers =  round( ((number_of_outliers/number_of_rows) * 100 ),2)
    print ("Columnn {}: The number of outliers is: {}, and the percentage is {}%" .for
```

```
Columnn PRI_NO_OF_ACCTS: The number of outliers is: 21914, and the percentage is 9.4%
Columnn PRI_ACTIVE_ACCTS: The number of outliers is: 32534, and the percentage is 13.
95%
Columnn PRI_OVERDUE_ACCTS: The number of outliers is: 26275, and the percentage is 1
1.27%
Columnn PRI_CURRENT_BALANCE: The number of outliers is: 41044, and the percentage is
17.6%
Columnn PRI_SANCTIONED_AMOUNT: The number of outliers is: 39587, and the percentage i
s 16.98%
Columnn PRI_DISBURSED_AMOUNT: The number of outliers is: 39712, and the percentage is
17.03%
Columnn SEC_NO_OF_ACCTS: The number of outliers is: 5865, and the percentage is 2.52%
Columnn SEC_ACTIVE_ACCTS: The number of outliers is: 3817, and the percentage is 1.6
4%
Columnn SEC_OVERDUE_ACCTS: The number of outliers is: 1337, and the percentage is 0.5
7%
Columnn SEC_CURRENT_BALANCE: The number of outliers is: 3364, and the percentage is
1.44%
Columnn SEC_SANCTIONED_AMOUNT: The number of outliers is: 3736, and the percentage is
1.6%
Columnn SEC_DISBURSED_AMOUNT: The number of outliers is: 3704, and the percentage is
1.59%
Columnn PRIMARY_INSTAL_AMT: The number of outliers is: 38868, and the percentage is 1
6.67%
Columnn SEC_INSTAL_AMT: The number of outliers is: 2217, and the percentage is 0.95%
```

>> For variables with an outlier percentage of less than 15%, I utilize the capping method.

>> For variables with an outlier percentage greater than 15%, I utilize log transformation. Due to the high number of zeros, I employ the log-plus-one transformation method.

In [48]:
```
less_outlier_columns = ['PRI_NO_OF_ACCTS' , 'PRI_ACTIVE_ACCTS','PRI_OVERDUE_ACCTS','SE
                        ,'SEC_OVERDUE_ACCTS' , 'SEC_CURRENT_BALANCE','SEC_SANCTIONED_AM
high_outlier_columns = ['PRI_CURRENT_BALANCE','PRI_SANCTIONED_AMOUNT' ,'PRI_DISBURSED_
```

For Less than 15% - First I check some specific percentiles

In [49]:
```
# less_outlier_columns

# Describe the selected columns and store it in a DataFrame
#quantiles = loan[less_outlier_columns].describe()

# Create a custom DataFrame with q1 and q5 added as rows
custom_description = pd.DataFrame()

# Calculate q1 and q5 for each column and add them to the custom_description DataFrame
for col in less_outlier_columns:
    quantiles = loan[col].describe()
    q1,q5,q10,q25,q50,q70,q75,q90,q95,q99 = np.percentile(loan[col], (1,5,10,25,50,70,
    iqr = q75 - q25
    custom_description[col] = quantiles
    custom_description.at['q1', col] = q1
    custom_description.at['q5', col] = q5
    custom_description.at['q10', col] = q10
```

```
    custom_description.at['q25', col] = q25
    custom_description.at['q50', col] = q50
    custom_description.at['q70', col] = q70
    custom_description.at['q90', col] = q90
    custom_description.at['q95', col] = q95
    custom_description.at['q99', col] = q99
    custom_description.at['iqr', col] = iqr

# Print the custom description
custom_description
```

Out[49]:

	PRI_NO_OF_ACCTS	PRI_ACTIVE_ACCTS	PRI_OVERDUE_ACCTS	SEC_NO_OF_ACCTS	SEC_ACTIVE_AC
count	233154.000000	233154.000000	233154.000000	233154.000000	233154.00
mean	2.440636	1.039896	0.156549	0.059081	0.02
std	5.217233	1.941496	0.548787	0.626795	0.31
min	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	0.000000	0.00
50%	0.000000	0.000000	0.000000	0.000000	0.00
75%	3.000000	1.000000	0.000000	0.000000	0.00
max	453.000000	144.000000	25.000000	52.000000	36.00
q1	0.000000	0.000000	0.000000	0.000000	0.00
q5	0.000000	0.000000	0.000000	0.000000	0.00
q10	0.000000	0.000000	0.000000	0.000000	0.00
q25	0.000000	0.000000	0.000000	0.000000	0.00
q50	0.000000	0.000000	0.000000	0.000000	0.00
q70	2.000000	1.000000	0.000000	0.000000	0.00
q90	7.000000	3.000000	1.000000	0.000000	0.00
q95	11.000000	5.000000	1.000000	0.000000	0.00
q99	23.000000	9.000000	2.000000	2.000000	1.00
iqr	3.000000	1.000000	0.000000	0.000000	0.00

In [50]:

```
print("Based on the above data summary: ")
print("PRI_NO_OF_ACCTS   : Sharp increase from the 95th percentile (11) to the 99th pe
print("PRI_ACTIVE_ACCTS  : Sharp increase from the 95th percentile (5) to the 99th per
print("PRI_OVERDUE_ACCTS : The data increases from 1 at the 90th percentile to 2 at th
print("\nI will remove these columns-before running the model- due to hight number of
of constant values (like zeros in this case) are less likely to contribute meaningful
print("SEC_NO_OF_ACCTS: The 95th percentile is 0, which means 95% of the values are ze
print("SEC_ACTIVE_ACCTS: 95% of the values are zero.")
print("SEC_OVERDUE_ACCTS: More than 95% of the values are zero.")
print("SEC_SANCTIONED_AMOUNT: Even though the 99th percentile has a value, 90% of the
print("SEC_DISBURSED_AMOUNT: Similar to the sanctioned amount, 90% of the data is zerc
print("SEC_INSTAL_AMT: 99% of the values are zero.")
```

Based on the above data summary:
PRI_NO_OF_ACCTS : Sharp increase from the 95th percentile (11) to the 99th percentile (23). Capping value : 11
PRI_ACTIVE_ACCTS : Sharp increase from the 95th percentile (5) to the 99th percentile (9). Capping value : 5
PRI_OVERDUE_ACCTS : The data increases from 1 at the 90th percentile to 2 at the 99th percentile. Capping value : 1

I will remove these columns-before running the model- due to hight number of zero. Given that I am using logistic regression,columns with a high proportion of constant values (like zeros in this case) are less likely to contribute meaningful information to the model.
SEC_NO_OF_ACCTS: The 95th percentile is 0, which means 95% of the values are zero.
SEC_ACTIVE_ACCTS: 95% of the values are zero.
SEC_OVERDUE_ACCTS: More than 95% of the values are zero.
SEC_SANCTIONED_AMOUNT: Even though the 99th percentile has a value, 90% of the data is zero.
SEC_DISBURSED_AMOUNT: Similar to the sanctioned amount, 90% of the data is zero.
SEC_INSTAL_AMT: 99% of the values are zero.

In [51]:
```
loan.PRI_NO_OF_ACCTS = np.where(loan.PRI_NO_OF_ACCTS > 11 , 11 , loan.PRI_NO_OF_ACCTS
loan.PRI_ACTIVE_ACCTS   = np.where(loan.PRI_ACTIVE_ACCTS   > 5 , 5 , loan.PRI_ACTIVE_A
loan.PRI_OVERDUE_ACCTS  = np.where(loan.PRI_OVERDUE_ACCTS  > 1 , 1 , loan.PRI_OVERDUE_
skewness_data = loan[less_outlier_columns].skew()
print(skewness_data)
print("The skewness for those three columns has decreased; the other columns will be r
```

```
PRI_NO_OF_ACCTS            1.735229
PRI_ACTIVE_ACCTS           1.620500
PRI_OVERDUE_ACCTS          2.449631
SEC_NO_OF_ACCTS           27.986090
SEC_ACTIVE_ACCTS          30.599510
SEC_OVERDUE_ACCTS         24.129271
SEC_CURRENT_BALANCE      108.506295
SEC_SANCTIONED_AMOUNT     75.254932
SEC_DISBURSED_AMOUNT      75.764252
SEC_INSTAL_AMT           153.806369
dtype: float64
The skewness for those three columns has decreased; the other columns will be removed later.
```

For more than 15% - First I check the skewness

In [52]:
```
# Plot histograms for columns with more than 15% outliers  - high_outlier_columns
#plt.figure(figsize=(15, 20))

#for i, col in enumerate(high_outlier_percentage, 1):
#    plt.subplot(5, 3, i)
#    sns.histplot(loan.sample(30000, random_state=42)[col], kde=True)
#    plt.title(f'Histogram for {col}')
#    plt.xlabel(col)
#    plt.ylabel('Frequency')

#plt.tight_layout()
#plt.show()

print("It takes a lot to plot the distribution, so to see the skewness I used the skew
skewness_data = loan[high_outlier_columns].skew()
print(skewness_data)
print("\n Since the data is highly skewed, I will use log transformation to handle pos
```

```
 So, I use log-plus-one transformation.")

print("For the PRI_CURRENT_BALANCE_shifted we have some negative values, so we need to
shift_value = abs(loan['PRI_CURRENT_BALANCE'].min()) + 1
loan['PRI_CURRENT_BALANCE'] = loan['PRI_CURRENT_BALANCE'] + shift_value

# Apply the transformation
for col in high_outlier_columns:
    loan[col] = np.log1p(loan[col])

skewness_data = loan[high_outlier_columns].skew()
print("\nThe skewness after the transformation:")
print(skewness_data)

print("\nThe transformed PRI_CURRENT_BALANCE data is now concentrated around a specifi
print(loan.PRI_CURRENT_BALANCE.describe()  )
```

```
It takes a lot to plot the distribution, so to see the skewness I used the skew funct
ion.
PRI_CURRENT_BALANCE       29.425813
PRI_SANCTIONED_AMOUNT    323.697212
PRI_DISBURSED_AMOUNT     322.541495
PRIMARY_INSTAL_AMT        69.916156
dtype: float64

 Since the data is highly skewed, I will use log transformation to handle positive sk
ewness, we have a lot of 0. So, I use log-plus-one transformation.
For the PRI_CURRENT_BALANCE_shifted we have some negative values, so we need to shift
the data first.

The skewness after the transformation:
PRI_CURRENT_BALANCE     -16.728728
PRI_SANCTIONED_AMOUNT     0.481521
PRI_DISBURSED_AMOUNT      0.487272
PRIMARY_INSTAL_AMT        0.940679
dtype: float64

The transformed PRI_CURRENT_BALANCE data is now concentrated around a specific value-
the original data had a lot of zeros-, with the 25th, 50th (median), and 75th percent
iles all being very close to each other (around 15.714 to 15.720). This suggests that
a significant portion of the data is clustered around this range.
count    233154.000000
mean         15.734689
std           0.084498
min           0.693147
25%          15.714374
50%          15.714374
75%          15.719602
max          18.452211
Name: PRI_CURRENT_BALANCE, dtype: float64
```

In [53]:
```
print(loan['PRI_CURRENT_BALANCE'].unique())
```

```
[15.71437372 15.84647733 15.89367078 ... 18.45221059 17.215544
 17.72498372]
```

In [55]:
```
custom_description = pd.DataFrame()

# Calculate q1 and q5 for each column and add them to the custom_description DataFrame
for col in other_columns:
```

```
    quantiles = loan[col].describe()
    q1,q5,q10,q25,q50,q70,q75,q90,q95,q99 = np.percentile(loan[col], (1,5,10,25,50,70,
    iqr = q75 - q25
    custom_description[col] = quantiles
    custom_description.at['q1', col] = q1
    custom_description.at['q5', col] = q5
    custom_description.at['q10', col] = q10
    custom_description.at['q25', col] = q25
    custom_description.at['q50', col] = q50
    custom_description.at['q70', col] = q70
    custom_description.at['q90', col] = q90
    custom_description.at['q95', col] = q95
    custom_description.at['q99', col] = q99
    custom_description.at['iqr', col] = iqr

# Print the custom description
display(custom_description)
#display(loan[other_columns].describe())
display(loan[other_columns].skew())
outlier_boundaries = {}
number_of_rows = loan.shape[0]

for col in other_columns:
    Q25, Q50, Q75 = np.percentile(loan[col] , (25,50,75))
    IQR = Q75 - Q25
    lower_bound = Q25 - 1.5 * IQR
    upper_bound = Q75 + 1.5 * IQR
    outlier_boundaries[col] = (lower_bound, upper_bound)

    number_of_outliers =  len(  loan[ (loan[col]> upper_bound) | (loan[col]<lower_bour
    percentage_of_outliers =  round( ((number_of_outliers/number_of_rows) * 100 ),2)
    print ("Columnn {}: The number of outliers is: {}, and the percentage is {}%" .for
```

	disbursed_amount	asset_cost	ltv	PERFORM_CNS_SCORE	NEW_ACCTS_IN_LAST_S
count	233154.000000	2.331540e+05	233154.000000	233154.000000	23
mean	54356.993528	7.586507e+04	74.746530	289.462994	
std	12971.314171	1.894478e+04	11.456636	338.374779	
min	13320.000000	3.700000e+04	10.030000	0.000000	
25%	47145.000000	6.571700e+04	68.880000	0.000000	
50%	53803.000000	7.094600e+04	76.800000	0.000000	
75%	60413.000000	7.920175e+04	83.670000	678.000000	
max	990572.000000	1.628992e+06	95.000000	890.000000	
q1	26529.000000	4.709953e+04	39.250000	0.000000	
q5	34939.000000	5.826400e+04	52.310000	0.000000	
q10	39794.000000	6.133000e+04	58.860000	0.000000	
q25	47145.000000	6.571700e+04	68.880000	0.000000	
q50	53803.000000	7.094600e+04	76.800000	0.000000	
q70	58759.000000	7.680700e+04	82.010000	627.000000	
q90	68882.000000	9.914000e+04	87.950000	761.000000	
q95	74122.350000	1.096800e+05	89.380000	825.000000	
q99	95391.000000	1.560160e+05	89.950000	836.000000	
iqr	13268.000000	1.348475e+04	14.790000	678.000000	

```
disbursed_amount                      4.492240
asset_cost                            6.133485
ltv                                  -1.075766
PERFORM_CNS_SCORE                     0.445150
NEW_ACCTS_IN_LAST_SIX_MONTHS          4.839326
DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS   6.641996
AVERAGE_ACCT_AGE                      3.285142
CREDIT_HISTORY_LENGTH                 2.969155
DisbursalTime                        -1.016395
NO_OF_INQUIRIES                       7.870683
Age                                   0.609162
dtype: float64
```

```
Columnn disbursed_amount: The number of outliers is: 9868, and the percentage is 4.2
3%
Columnn asset_cost: The number of outliers is: 24388, and the percentage is 10.46%
Columnn ltv: The number of outliers is: 6170, and the percentage is 2.65%
Columnn PERFORM_CNS_SCORE: The number of outliers is: 0, and the percentage is 0.0%
Columnn NEW_ACCTS_IN_LAST_SIX_MONTHS: The number of outliers is: 51660, and the perce
ntage is 22.16%
Columnn DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS: The number of outliers is: 18195, and th
e percentage is 7.8%
Columnn AVERAGE_ACCT_AGE: The number of outliers is: 15924, and the percentage is 6.8
3%
Columnn CREDIT_HISTORY_LENGTH: The number of outliers is: 16056, and the percentage i
s 6.89%
Columnn DisbursalTime: The number of outliers is: 0, and the percentage is 0.0%
Columnn NO_OF_INQUIRIES: The number of outliers is: 31193, and the percentage is 13.3
8%
```

<<<<<<<<<< Here is a summary and my approach for outlier treatment on the remaining columns: >>>>>>>>>>

- **disbursed_amount:**

Skewness: 4.49

Outliers: 4.23%

Approach: Moderate skewness and low outlier percentage. I will utilize capping based on percentiles.

- **asset_cost:**

Skewness: 6.13

Outliers: 10.46%

Approach: High skewness and moderate outlier percentage. Log transformation can be considered, or capping can be applied.

- **ltv:**

Skewness: -1.08

Outliers: 2.65%

Approach: Negative skewness indicates left skewness. I will keep the values as is.

- **PERFORM_CNS_SCORE:**

Skewness: 0.44

Outliers: 0%

Approach: Relatively low skewness and no outliers. I will keep this column as it is.

- **NEW_ACCTS_IN_LAST_SIX_MONTHS:**

Skewness: 4.84

Outliers: 22.16%

Approach: High skewness and a significant outlier percentage. Log transformation can be considered. We have a lot of 0. So, I use log-plus-one transformation.

- **DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS:**

Skewness: 6.64

Outliers: 7.8%
Approach: High skewness. Log transformation can be considered, or capping can be applied.

- **AVERAGE_ACCT_AGE:**

Skewness: 3.29
Outliers: 6.83%
Approach: Moderate skewness. I will use capping or log transformation.

- **CREDIT_HISTORY_LENGTH:**

Skewness: 2.97
Outliers: 6.89%
Approach: Moderate skewness. Consider capping or log transformation.

- **DisbursalTime:**

Skewness: -0.83
Outliers: 0%
Approach: Negative skewness, but no outliers. I will keep this column as it is.

- **NO_OF_INQUIRIES:**

Skewness: 7.87
Outliers: 13.38%
Approach: High skewness and moderate outlier percentage. Log transformation can be considered.

- **Age:**

Skewness: 0.61
Outliers: 0.01%
Approach: Relatively low skewness and very few outliers. I will keep this column as it is.

To summarize:

For columns with high skewness (>2 or <-2) and a significant percentage of outliers, I consider log transformation.

For columns with moderate skewness and outliers, I consider capping based on percentiles.

For columns with low skewness and outliers, I keep them as they are.

In [56]:
```
columns_for_capping_treatment = { 'disbursed_amount' :74122 ,'AVERAGE_ACCT_AGE' :68 ,'
# Uppdating values upper than Cap Value
for col,cap_value in columns_for_capping_treatment.items():
    loan[col] = np.where(loan[col] > cap_value , cap_value , loan[col]  )

# I use  Log_plus_one, due to high number of zeros.
columns_for_log_treatment = ['asset_cost','NEW_ACCTS_IN_LAST_SIX_MONTHS','DELINQUENT_A
```

```
for col in columns_for_log_treatment:
    loan[col] = np.log1p(loan[col])
```

In [58]:

```
custom_description = pd.DataFrame()

# Calculate q1 and q5 for each column and add them to the custom_description DataFrame
for col in other_columns:
    quantiles = loan[col].describe()
    q1,q5,q10,q25,q50,q70,q75,q90,q95,q99 = np.percentile(loan[col], (1,5,10,25,50,70,
    iqr = q75 - q25
    custom_description[col] = quantiles
    custom_description.at['q1', col] = q1
    custom_description.at['q5', col] = q5
    custom_description.at['q10', col] = q10
    custom_description.at['q25', col] = q25
    custom_description.at['q50', col] = q50
    custom_description.at['q70', col] = q70
    custom_description.at['q90', col] = q90
    custom_description.at['q95', col] = q95
    custom_description.at['q99', col] = q99
    custom_description.at['iqr', col] = iqr

# Print the custom description
display(custom_description)
#display(loan[other_columns].describe())
display(loan[other_columns].skew())
outlier_boundaries = {}
number_of_rows = loan.shape[0]

for col in other_columns:
    Q25, Q50, Q75 = np.percentile(loan[col] , (25,50,75))
    IQR = Q75 - Q25
    lower_bound = Q25 - 1.5 * IQR
    upper_bound = Q75 + 1.5 * IQR
    outlier_boundaries[col] = (lower_bound, upper_bound)

    number_of_outliers =  len(  loan[ (loan[col]> upper_bound) | (loan[col]<lower_bour
    percentage_of_outliers =  round( ((number_of_outliers/number_of_rows) * 100 ),2)
    print ("Columnn {}: The number of outliers is: {}, and the percentage is {}%" .for
```

	disbursed_amount	asset_cost	ltv	PERFORM_CNS_SCORE	NEW_ACCTS_IN_LAST_S
count	233154.000000	233154.000000	233154.000000	233154.000000	2
mean	53717.306180	11.213166	74.746530	289.462994	
std	10844.110749	0.206688	11.456636	338.374779	
min	13320.000000	10.518700	10.030000	0.000000	
25%	47145.000000	11.093128	68.880000	0.000000	
50%	53803.000000	11.169688	76.800000	0.000000	
75%	60413.000000	11.279766	83.670000	678.000000	
max	74122.000000	14.303473	95.000000	890.000000	
q1	26529.000000	10.760040	39.250000	0.000000	
q5	34939.000000	10.972757	52.310000	0.000000	
q10	39794.000000	11.024041	58.860000	0.000000	
q25	47145.000000	11.093128	68.880000	0.000000	
q50	53803.000000	11.169688	76.800000	0.000000	
q70	58759.000000	11.249064	82.010000	627.000000	
q90	68882.000000	11.504298	87.950000	761.000000	
q95	74122.000000	11.605331	89.380000	825.000000	
q99	74122.000000	11.957720	89.950000	836.000000	
iqr	13268.000000	0.186638	14.790000	678.000000	

```
disbursed_amount                    -0.171861
asset_cost                           1.279172
ltv                                 -1.075766
PERFORM_CNS_SCORE                    0.445150
NEW_ACCTS_IN_LAST_SIX_MONTHS         2.052650
DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS  3.723389
AVERAGE_ACCT_AGE                     2.155461
CREDIT_HISTORY_LENGTH                2.475342
DisbursalTime                       -1.016395
NO_OF_INQUIRIES                      3.007293
Age                                  0.609162
dtype: float64
```

```
Columnn disbursed_amount: The number of outliers is: 2620, and the percentage is 1.1
2%
Columnn asset_cost: The number of outliers is: 19687, and the percentage is 8.44%
Columnn ltv: The number of outliers is: 6170, and the percentage is 2.65%
Columnn PERFORM_CNS_SCORE: The number of outliers is: 0, and the percentage is 0.0%
Columnn NEW_ACCTS_IN_LAST_SIX_MONTHS: The number of outliers is: 51660, and the perce
ntage is 22.16%
Columnn DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS: The number of outliers is: 18195, and th
e percentage is 7.8%
Columnn AVERAGE_ACCT_AGE: The number of outliers is: 15924, and the percentage is 6.8
3%
Columnn CREDIT_HISTORY_LENGTH: The number of outliers is: 16056, and the percentage i
s 6.89%
Columnn DisbursalTime: The number of outliers is: 0, and the percentage is 0.0%
Columnn NO_OF_INQUIRIES: The number of outliers is: 31193, and the percentage is 13.3
8%
```

In [59]:
```
loan['NEW_ACCTS_IN_LAST_SIX_MONTHS'] = np.where(loan['NEW_ACCTS_IN_LAST_SIX_MONTHS'] >
loan['NO_OF_INQUIRIES'] = np.where(loan['NO_OF_INQUIRIES'] > 1.386294 , 1.386294 , loa
```

Performing EDA

In [60]:
```
loan.describe()
```

Out[60]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id
count	233154.000000	233154.000000	233154.000000	233154.000000	233154.000000	233154.000000
mean	535917.573376	53717.306180	11.213166	74.746530	72.936094	19638.635035
std	68315.693711	10844.110749	0.206688	11.456636	69.834995	3491.949566
min	417428.000000	13320.000000	10.518700	10.030000	1.000000	10524.000000
25%	476786.250000	47145.000000	11.093128	68.880000	14.000000	16535.000000
50%	535978.500000	53803.000000	11.169688	76.800000	61.000000	20333.000000
75%	595039.750000	60413.000000	11.279766	83.670000	130.000000	23000.000000
max	671084.000000	74122.000000	14.303473	95.000000	261.000000	24803.000000

8 rows × 39 columns

In [63]:
```
loan.manufacturer_id.unique
```

Out[63]:
```
<bound method Series.unique of 0         45
1         45
2         45
3         45
4         86
          ..
233149    51
233150    51
233151    45
233152    48
233153    45
Name: manufacturer_id, Length: 233154, dtype: int64>
```

-Provide the statistical description of the quantitative data variables

Data Summary: The dataset contains a total of 233,154 records.

Disbursed Amount: The disbursed amount varies with a mean of approximately 53,458.59 and a standard deviation of approximately 9,304.92. The minimum disbursed amount is 29,484, while the maximum disbursed amount is 77,122.

Asset Cost: The asset cost has a mean of approximately 75,865.07 and a standard deviation of approximately 18,944.78. The minimum asset cost is 37,000, and the maximum asset cost is 1,628,992.

Loan-to-Value Ratio (LTV): The LTV ratio has a mean of approximately 74.75, with a standard deviation of approximately 11.46. The LTV ratio ranges from a minimum of 10.03 to a maximum of 95.00.

New Accounts in Last Six Months: On average, there are 0.38 new accounts in the last six months, with a maximum of 35.

Delinquent Accounts in Last Six Months: On average, there are 0.097 delinquent accounts in the last six months, with a maximum of 20.

Average Account Age: The average account age ranges from 0 to 369 months, with a mean of approximately 8.92 months.

Credit History Length: Credit history length varies widely, with a mean of approximately 16.25 months and a maximum of 468 months.

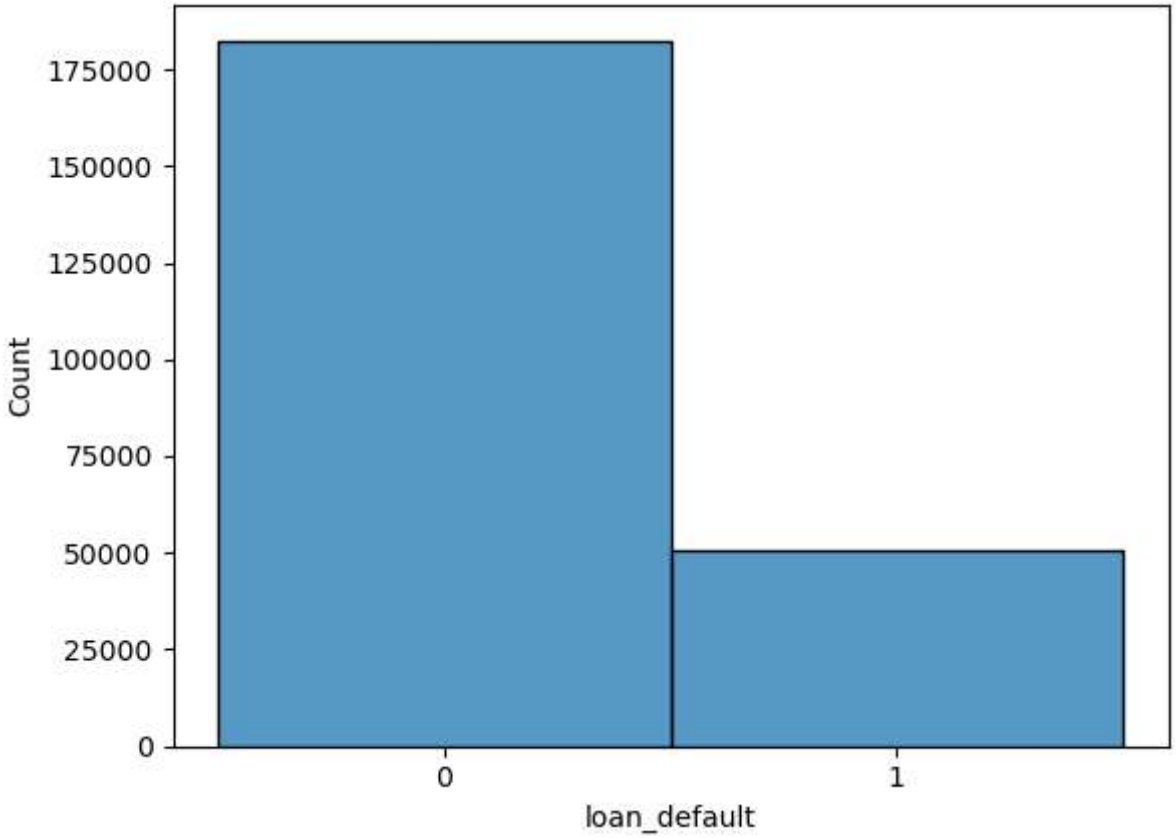
Number of Inquiries: On average, there are 0.21 inquiries, with a maximum of 36.

Loan Default: The dataset indicates loan default, with 0 indicating non-default and 1 indicating default.

Age: The ages of the individuals in the dataset vary, with a mean of approximately 38.96 years.

-Explain how is the target variable distributed overall

In [64]:
```
colors = ["red" , "yellow"]
sns.histplot(loan , x=loan.loan_default , discrete = True , multiple='stack')
plt.xticks([0, 1])
plt.show()
print(loan.loan_default.value_counts())
print("Approximately 78.27% of the entries are 0 (no default).\nApproximately 21.73% o
print("So, the dataset is imbalanced with respect to the target variable loan_default"
```




```
0    182543
1     50611
Name: loan_default, dtype: int64
Approximately 78.27% of the entries are 0 (no default).
Approximately 21.73% of the entries are 1 (defaulted).
So, the dataset is imbalanced with respect to the target variable loan_default
```

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

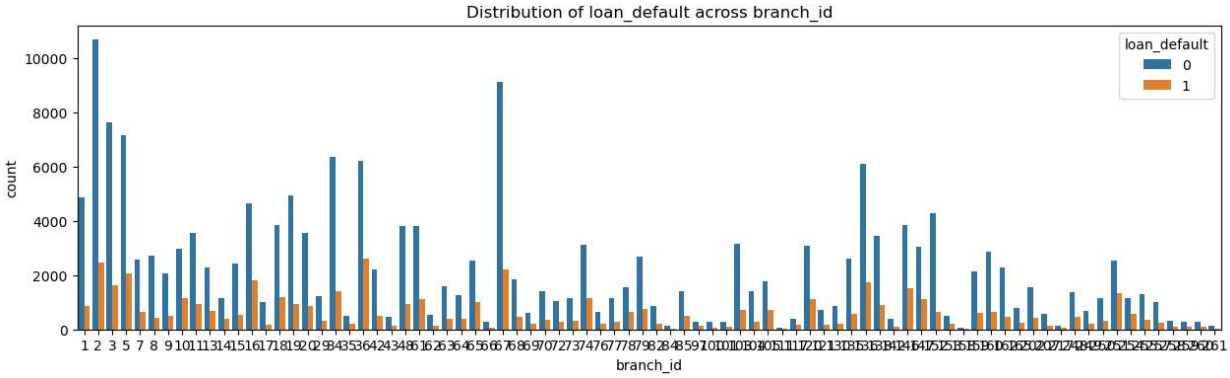
First, I will plot the distribution. Then, I will calculate the rate for each category to see which items have a higher default rate

In [95]:

```
categories = ['branch_id','supplier_id','manufacturer_id','State_ID']
for col in categories:
    display(
        loan
            .groupby([col,'loan_default'])['UniqueID']
            .count()
            .reset_index()
            .rename(columns = {'UniqueID' : 'count'})
            .sort_values(by = 'count' , ascending = False))

    plt.figure(figsize=(15,4))
    ax = sns.countplot(data=loan, x=col,hue='loan_default')
    ax.set_title('Distribution of loan_default across '+ col)
    plt.show()

print(" I will calcaulate the rate for each category to see what items have more dafau
for col in categories:
    # Calculate the percentage of defaulters for each branch
    default_rate = loan.groupby(col).apply(lambda x: (x['loan_default'].sum() / x['loa
    default_rate.columns = [col, 'default_rate']

    # Sort the dataframe by default_rate for better visualization
    default_rate = default_rate.sort_values(by='default_rate', ascending=False)

    # Plotting
    plt.figure(figsize=(15,4))
    ax = sns.barplot(data=default_rate, x=col, y='default_rate', order=default_rate[co
    ax.set_title('Percentage of Defaulters across '+col)
    plt.ylabel('Default Rate (%)')
    plt.xlabel(col)
   # plt.xticks(rotation=45)  # Rotate branch_id labels for better readability
    plt.show()


# Use this command to enable the scrolling for outputs in the notebook
display(HTML("<style>div.output_scroll { height: auto; }</style>"))
```

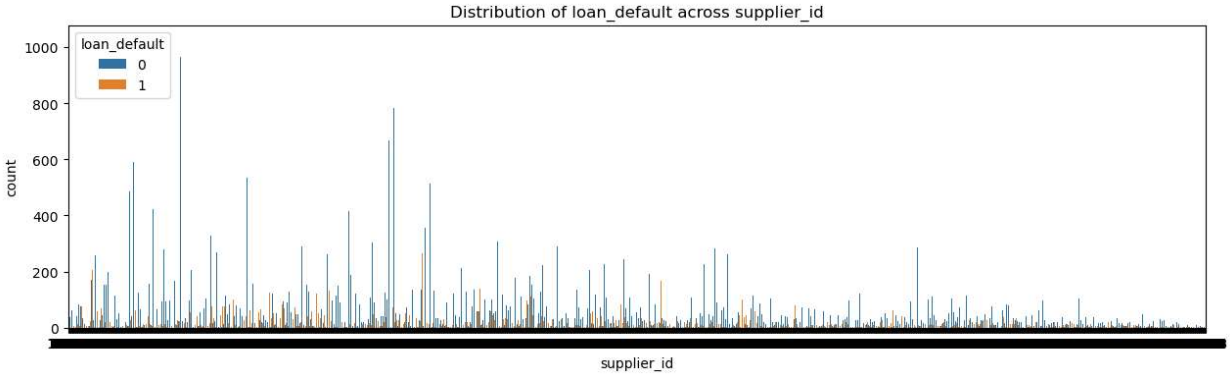
	branch_id	loan_default	count
2	2	0	10683
60	67	0	9130
4	3	0	7616
6	5	0	7171
36	34	0	6375
...	...	...	...
141	217	1	45
163	261	1	39
85	84	1	31
101	111	1	20
127	158	1	19

164 rows × 3 columns

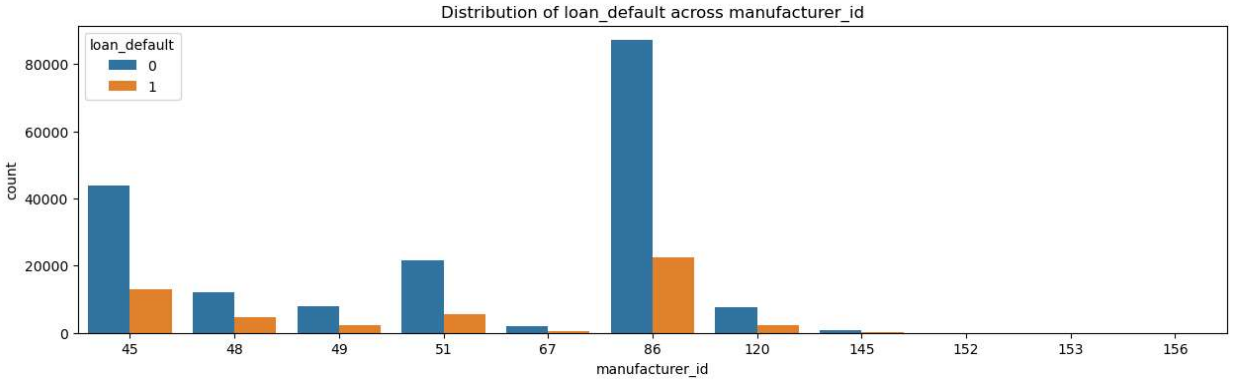


	supplier_id	loan_default	count
1412	18166	0	1028
219	14375	0	988
1310	17980	0	983
565	15663	0	967
1496	18317	0	950
...	...	...	...
2211	21891	1	1
2199	21872	1	1
2198	21872	0	1
2187	21847	1	1
5564	24803	1	1

5565 rows × 3 columns

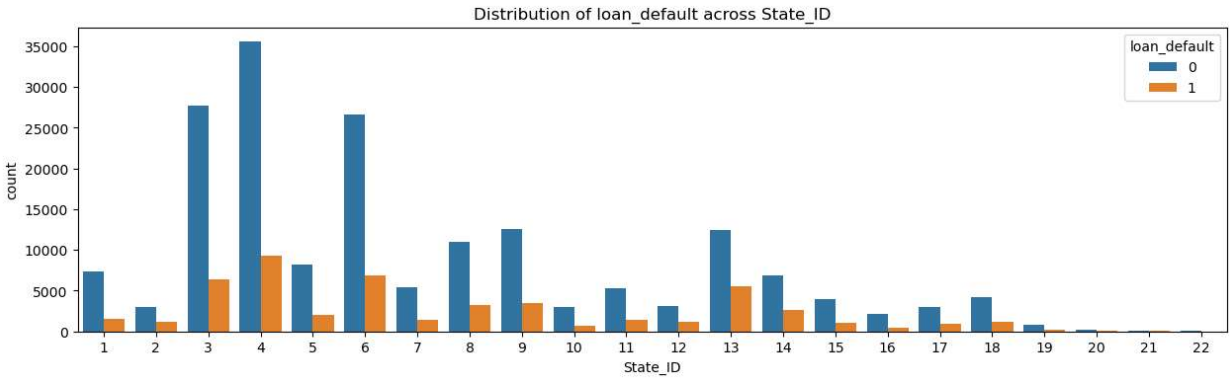


	manufacturer_id	loan_default	count
10	86	0	87124
0	45	0	43687
11	86	1	22410
6	51	0	21547
1	45	1	12939
2	48	0	12156
4	49	0	7984
12	120	0	7526
7	51	1	5657
3	48	1	4554
5	49	1	2236
13	120	1	2132
8	67	0	1882
14	145	0	622
9	67	1	523
15	145	1	156
17	153	0	8
16	152	0	6
18	153	1	4
19	156	0	1

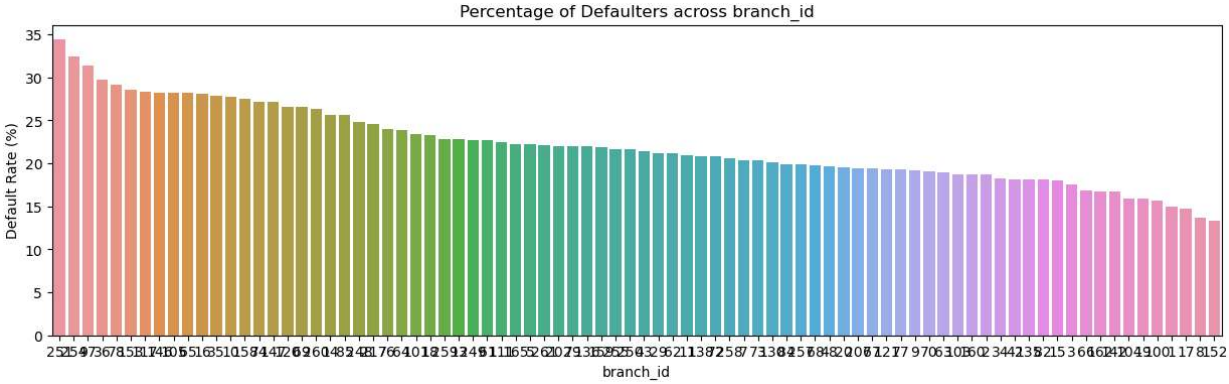

Distribution of loan_default across manufacturer_id
loan_default
0
1
count
80000
60000
40000
20000
0
45
48
49
51
67
86
120
145
152
153
156
manufacturer_id

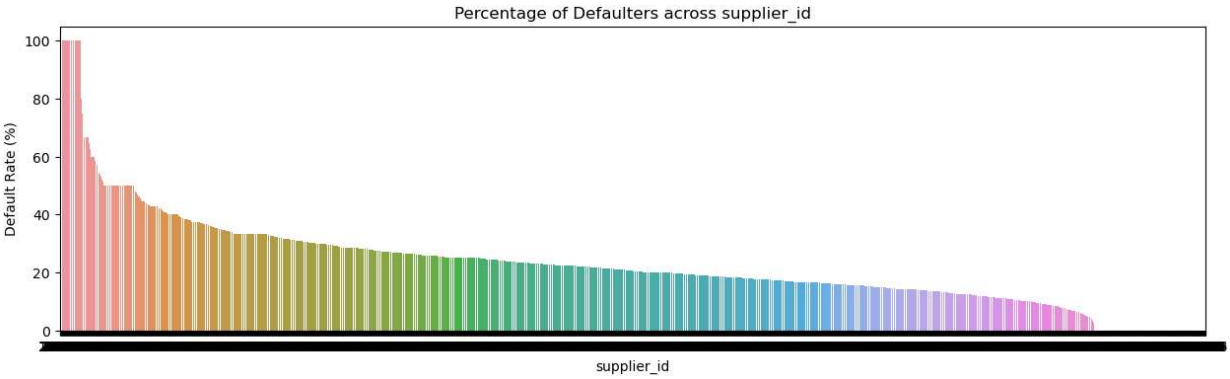
	State_ID	loan_default	count
6	4	0	35544
4	3	0	27733
10	6	0	26615
16	9	0	12530
24	13	0	12401
14	8	0	10939
7	4	1	9326
8	5	0	8154
0	1	0	7353
11	6	1	6890
26	14	0	6817
5	3	1	6345
25	13	1	5483
12	7	0	5417
20	11	0	5348
34	18	0	4221
28	15	0	3981
17	9	1	3492
15	8	1	3258
22	12	0	3092
2	2	0	3031
32	17	0	3010
18	10	0	2972
27	14	1	2597
30	16	0	2192
9	5	1	2023
1	1	1	1583
21	11	1	1373
13	7	1	1369
35	18	1	1191
3	2	1	1129
23	12	1	1118
29	15	1	1068
33	17	1	981

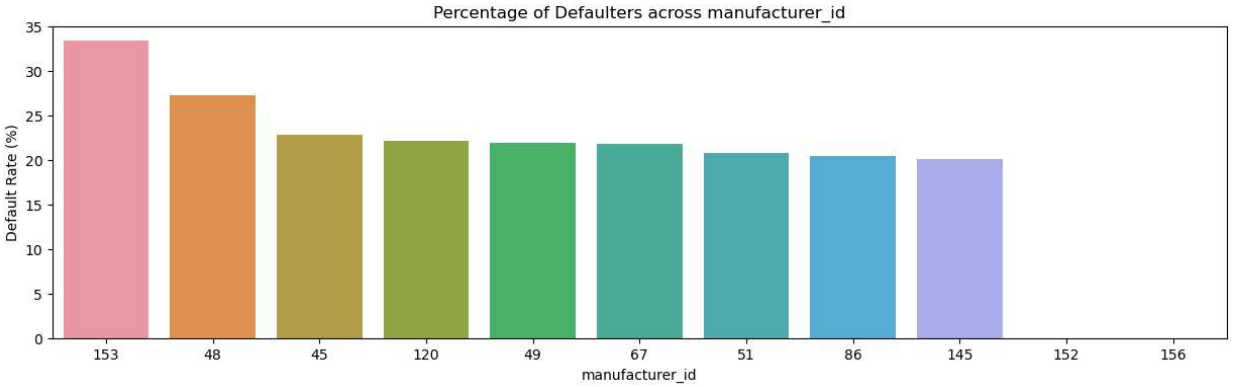
	State_ID	loan_default	count
36	19	0	848
19	10	1	633
31	16	1	493
37	19	1	187
38	20	0	153
40	21	0	125
42	22	0	67
39	20	1	32
41	21	1	31
43	22	1	9

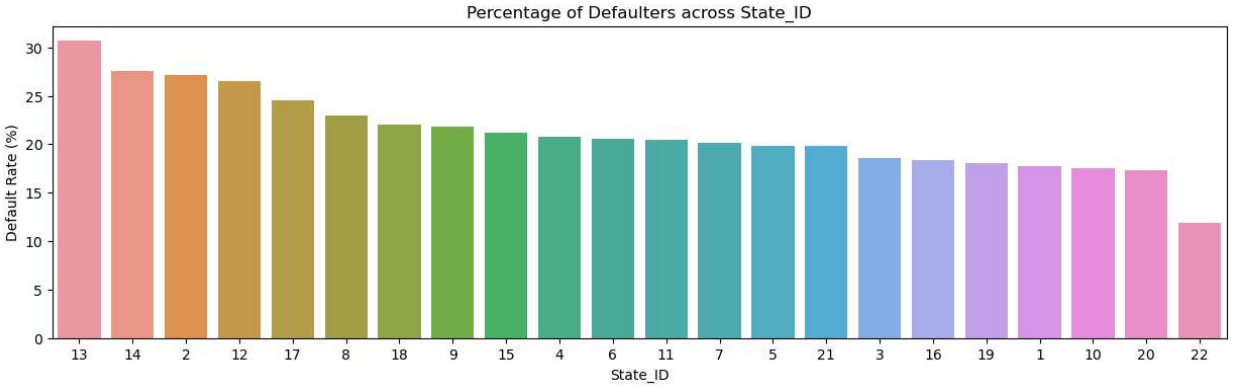


I will calcaulate the rate for each category to see what items have more dafaulters rate





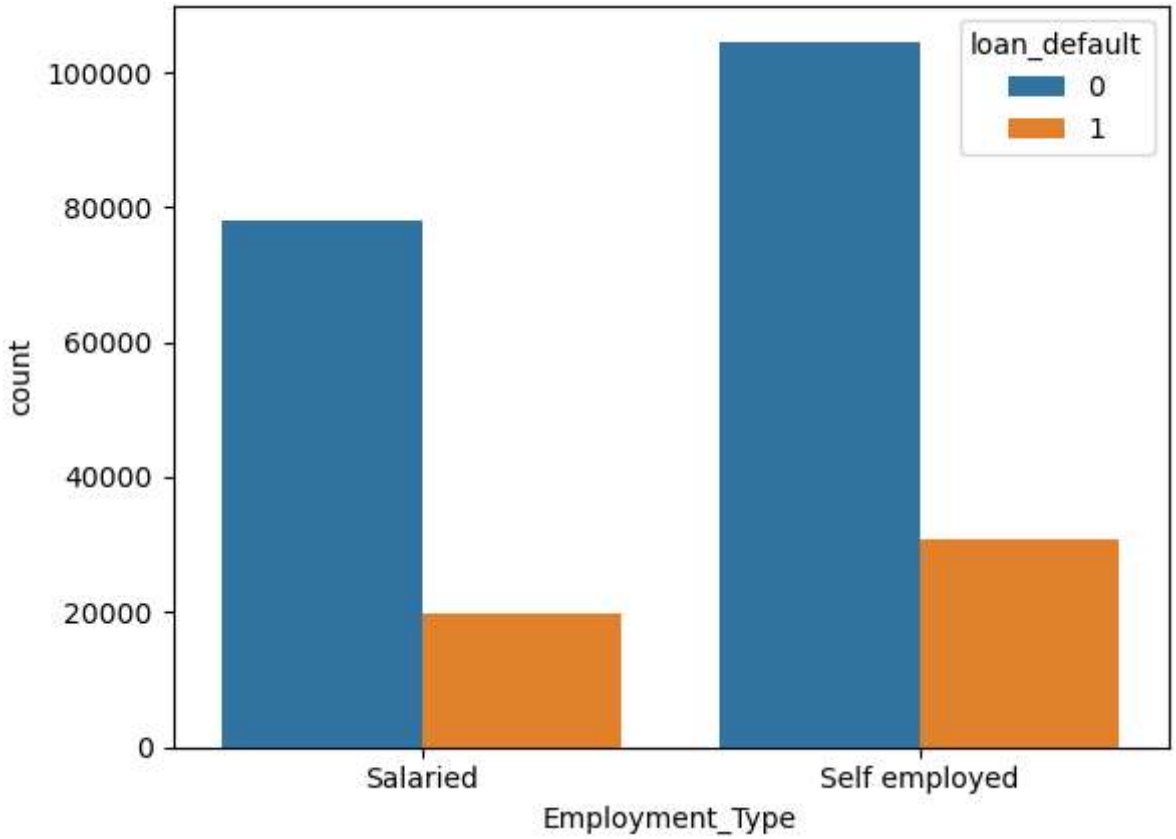




-What are the different employment types given in the data? Can a strategy be developed to fill in the missing values (if any)? Use pie charts to express the different types of employment that define the defaulters and non-defaulters.

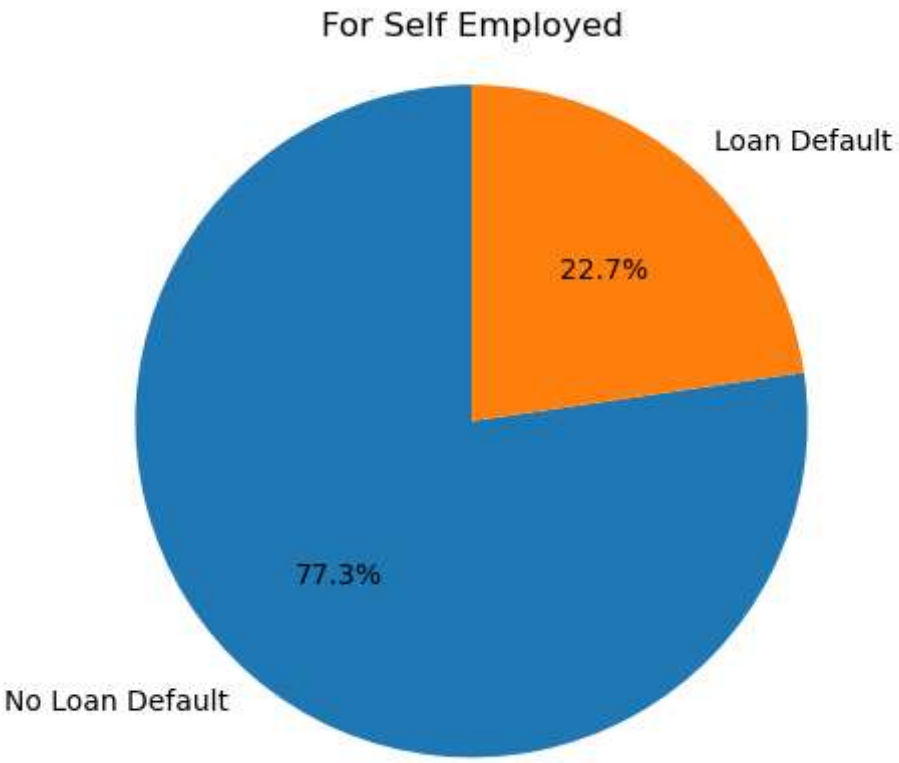
In [65]:
```
d= loan.groupby(['Employment_Type','loan_default'])['UniqueID'].count().reset_index().
sns.barplot(data = d  ,x='Employment_Type' , y ='count',hue ='loan_default' )
```

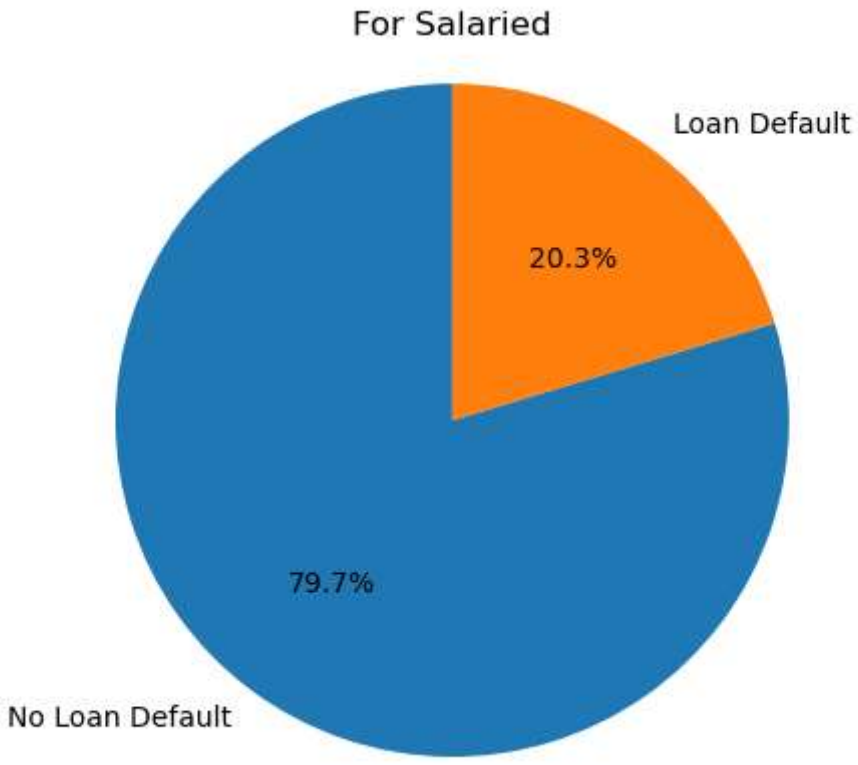
Out[65]:
```
<AxesSubplot:xlabel='Employment_Type', ylabel='count'>
```



In [67]:
```
print("Similar behavior across Employment Types: despite the difference in employment
both groups exhibit somewhat comparable behavior when it comes to loan repayment. \
This suggests that the nature of employment (Salaried vs. Self Employed) might not be

l_self_employed = loan[loan.Employment_Type =='Self employed']
l_salaried = loan[loan.Employment_Type=='Salaried']
counts = l_self_employed['loan_default'].value_counts()
labels = ['No Loan Default', 'Loan Default']
sizes = counts.values

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('For Self Employed')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()

counts = l_salaried['loan_default'].value_counts()
labels = ['No Loan Default', 'Loan Default']
sizes = counts.values

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('For Salaried')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
print("Regarding the missing values, I tried to use Asset_cost for finding the corresp
and the average for Self Employed EmploymentType is : 76578.66\
There is no significant difference in the averages between the two groups. Therefore,
So I used mode for imputation in the missing value section.")
```

```
Similar behavior across Employment Types: despite the difference in employment types,
both groups exhibit somewhat comparable behavior when it comes to loan repayment. Thi
s suggests that the nature of employment (Salaried vs. Self Employed) might not be a
dominant factor in determining loan default
```



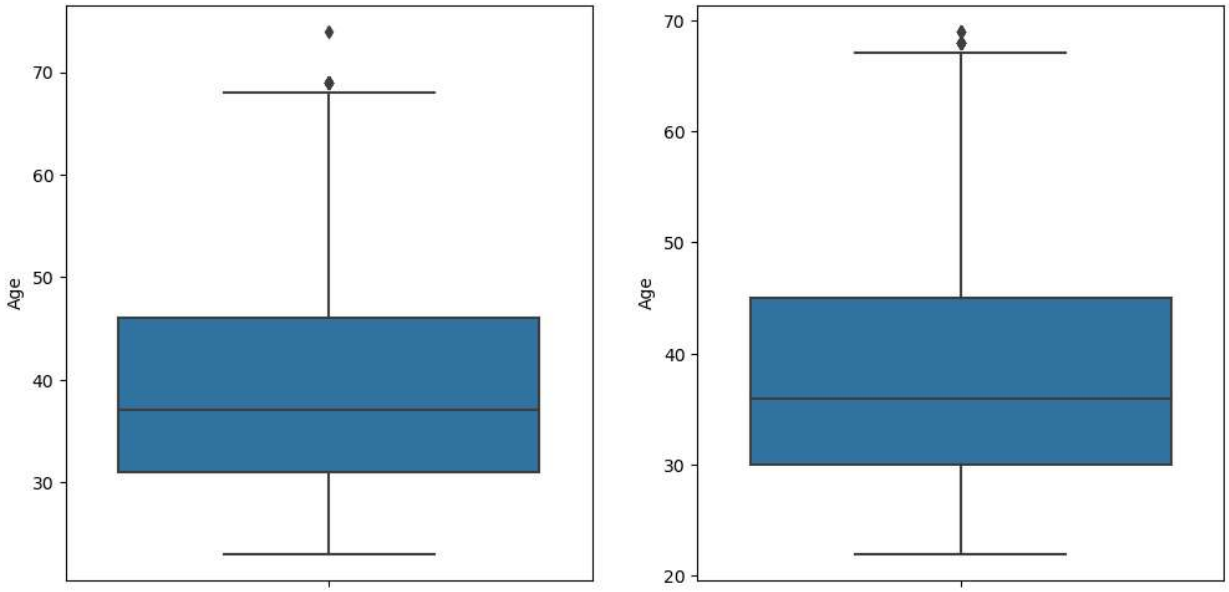


Regarding the missing values, I tried to use Asset_cost for finding the correspondant EmploymentType, but the average for Salaried EmploymentType : 74395.27and the average for Self Employed EmploymentType is : 76578.66There is no significant difference in the averages between the two groups. Therefore, this variable may not be helpful for imputing Employment_type.So I used mode for imputation in the missing value section.

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

In [68]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
sns.boxplot(data = loan[loan.loan_default ==0] , y = 'Age' , ax=axes[0])
sns.boxplot(data = loan[loan.loan_default ==1] , y = 'Age' , ax=axes[1])
```

Out[68]:
```
<AxesSubplot:ylabel='Age'>
```

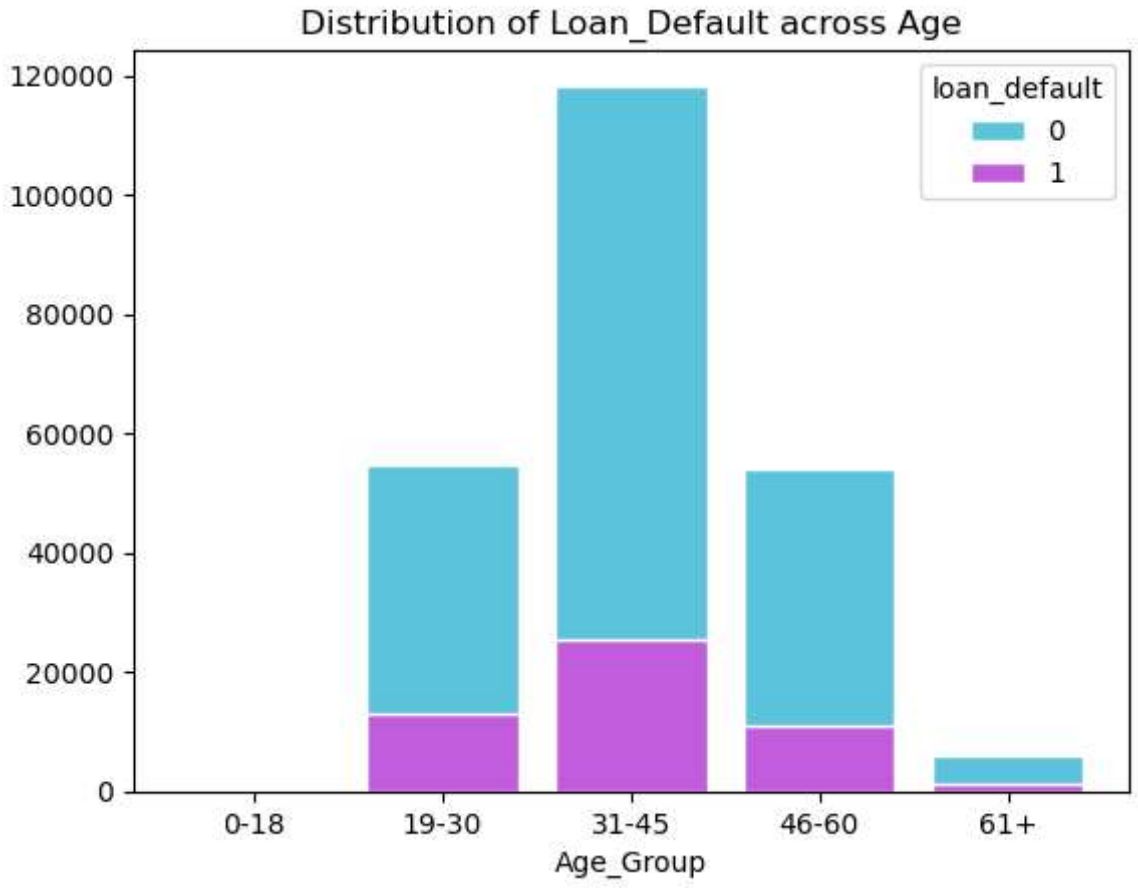


In [69]:
```
print("Age does seem to have a relationship with defaulting. Younger individuals are m
print("Approximately 23% of the young adults(19-30 years) have defaulted on their loan


d = loan.groupby(['Age_Group','loan_default']).count()['UniqueID'].reset_index().renam
display(d)
ax = sns.histplot(data=d , x = 'Age_Group' , hue = 'loan_default', multiple ='stack' ,
            palette=['#24b1d1', '#ae24d1'],
            edgecolor='white',
            shrink=0.8)
ax.set_title('Distribution of Loan_Default across Age')
# Remove 'Count' ylabel.
ax.set_ylabel(None)
```

```
Age does seem to have a relationship with defaulting. Younger individuals are more li
kely to default compare to other age groups.
Approximately 23% of the young adults(19-30 years) have defaulted on their loans. As
age increases, the default rate seems to slightly decrease.
```

	Age_Group	loan_default	count
0	0-18	0	0
1	0-18	1	0
2	19-30	0	41514
3	19-30	1	13048
4	31-45	0	92745
5	31-45	1	25577
6	46-60	0	43281
7	46-60	1	10885
8	61+	0	5003
9	61+	1	1101

Out[69]: Text(0, 0.5, '')



-What type of ID was presented by most of the customers for proof?

In [73]:
```
columns = ['MobileNo_Avl_Flag','Aadhar_flag','PAN_flag','VoterID_flag','Driving_flag',
column_sums = loan[columns].sum()
print(column_sums.sort_values(ascending = False))
print("The Aadhar card is the most popular choice for ID proof among customers, \
```

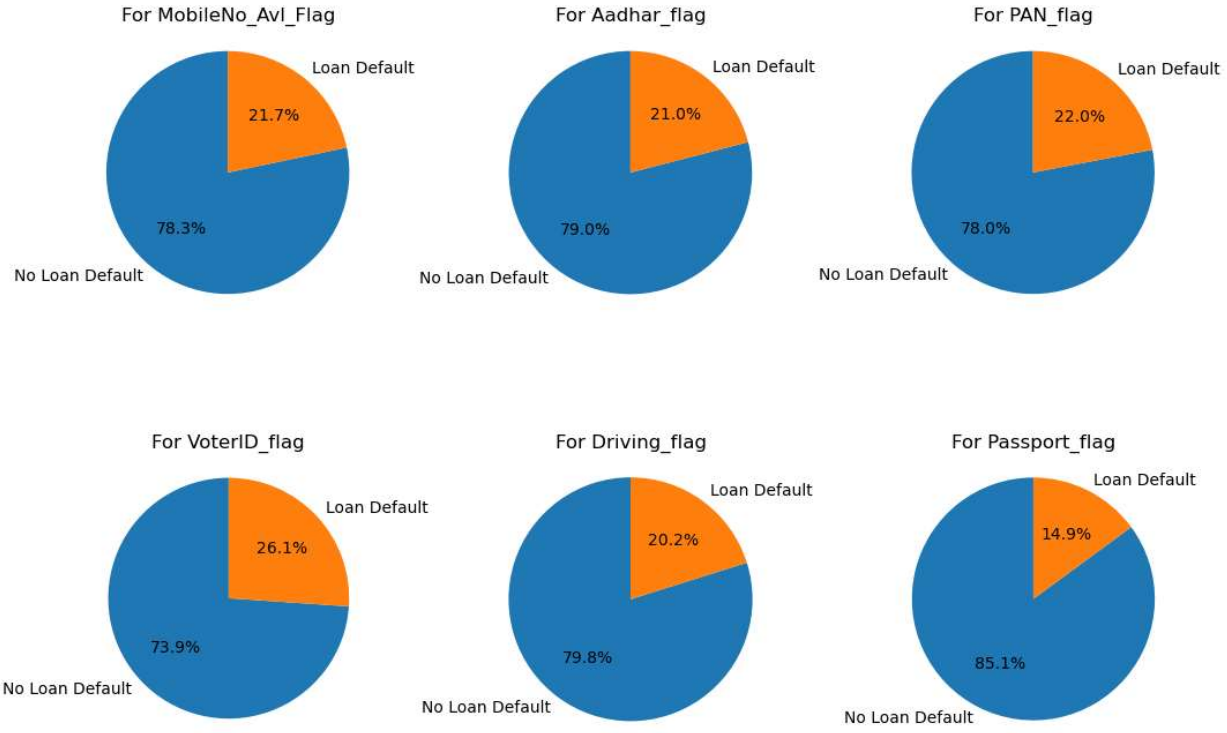
```
followed by the Voter ID and then PAN card.\
The least common proof is the passport. \
The mobile number seems to be a mandatory field, as every customer in the dataset has
```

```
MobileNo_Avl_Flag    233154
Aadhar_flag          195924
VoterID_flag          33794
PAN_flag              17621
Driving_flag           5419
Passport_flag           496
dtype: int64
```

The Aadhar card is the most popular choice for ID proof among customers, followed by the Voter ID and then PAN card.The least common proof is the passport. The mobile number seems to be a mandatory field, as every customer in the dataset has provided it.

In [74]:

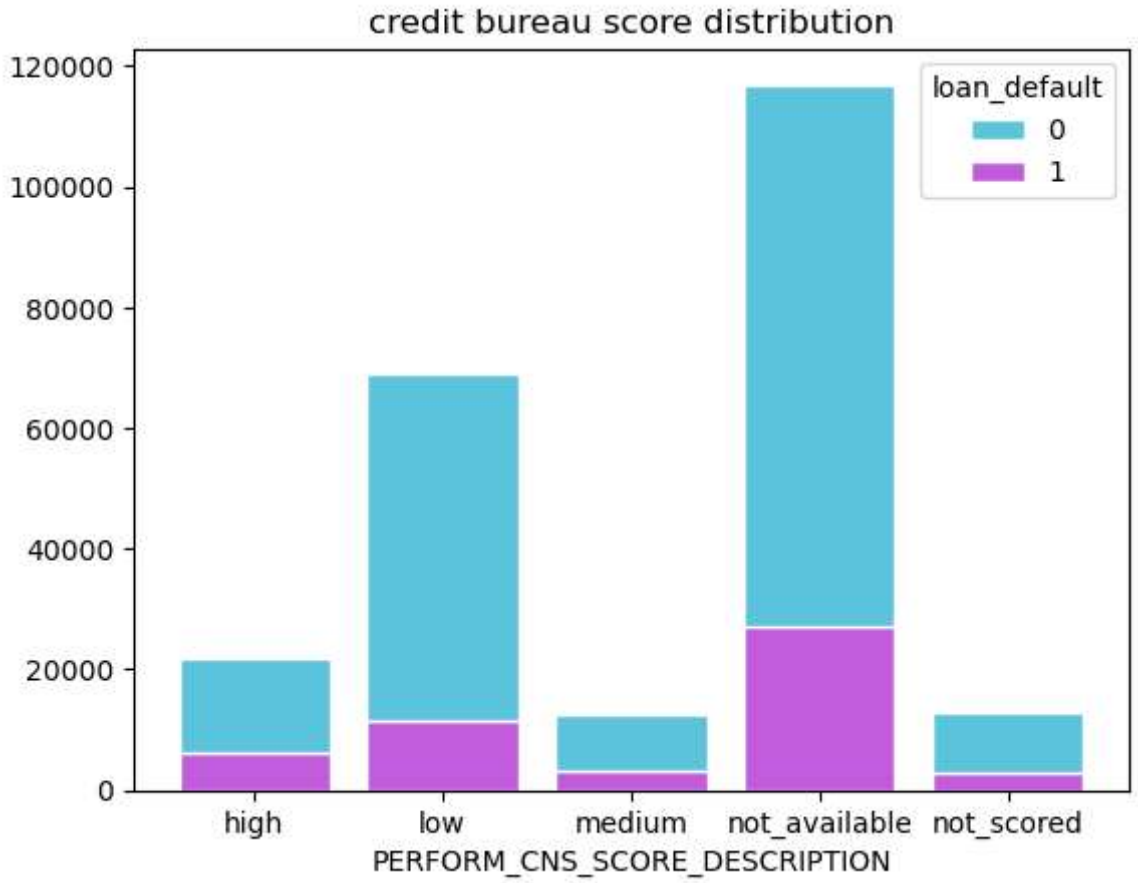
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 3, figsize=(12, 8))
plt.subplots_adjust(wspace=0.5, hspace=0.5)  # Adjust spacing between subplots

# Define the flags and corresponding titles
flags = ['MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag', 'VoterID_flag', 'Driving_flag
titles = ['MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag', 'VoterID_flag', 'Driving_fla

for flag, title, ax in zip(flags, titles, axes.flatten()):
    counts = loan[loan[flag] == 1]['loan_default'].value_counts()
    labels = ['No Loan Default', 'Loan Default']
    sizes = counts.values

    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
    ax.set_title(f'For {title}')
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



-Study the credit bureau score distribution. Compare the distribution for defaulters vs. non-defaulters. Explore in detail.

In [75]:
```
d = loan.groupby(['PERFORM_CNS_SCORE_DESCRIPTION','loan_default']).count()['UniqueID']

ax = sns.histplot(data=d , x = 'PERFORM_CNS_SCORE_DESCRIPTION' , hue = 'loan_default',
            palette=['#24b1d1', '#ae24d1'],
            edgecolor='white',
            shrink=0.8)
ax.set_title('credit bureau score distribution')
# Remove 'Count' ylabel.
ax.set_ylabel(None)
```

Out[75]:
```
Text(0, 0.5, '')
```

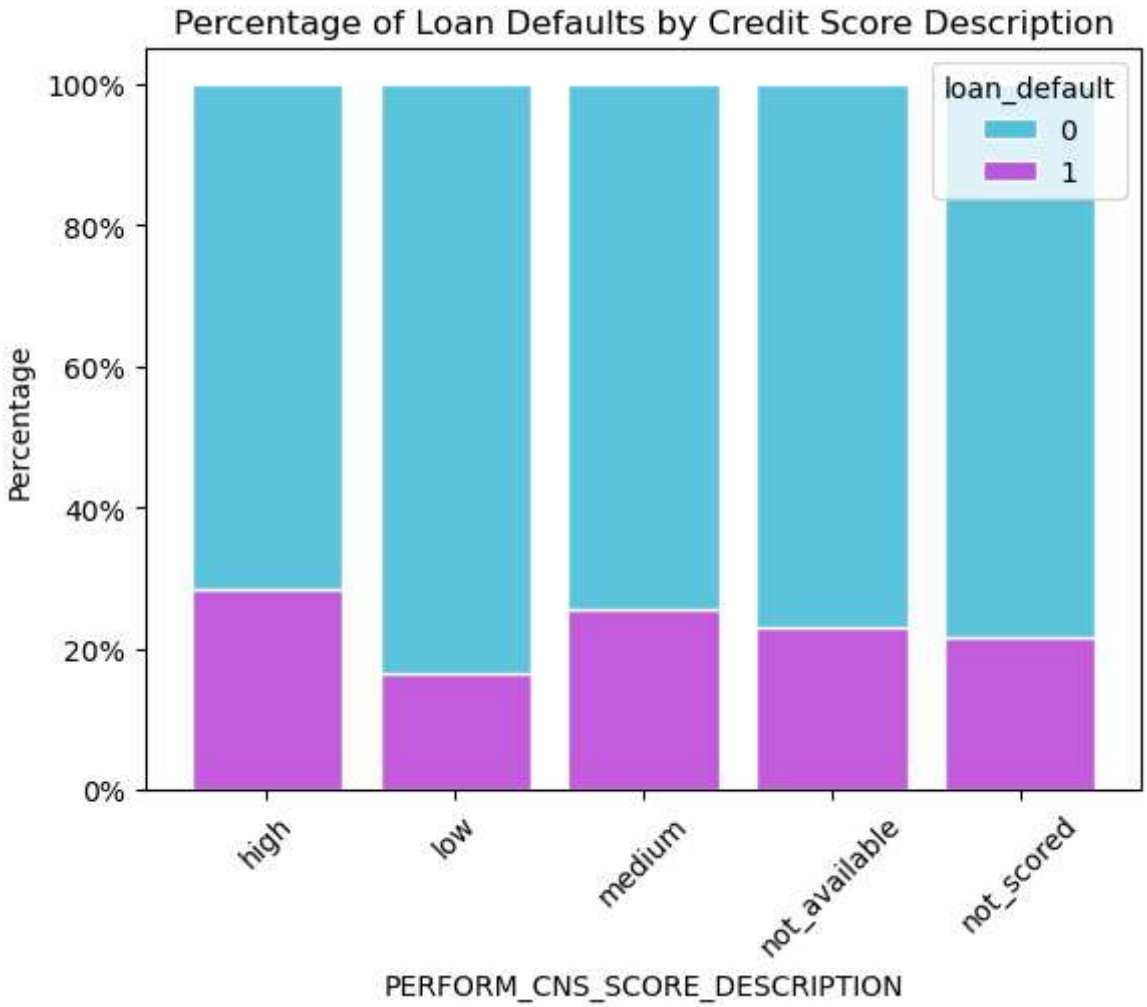


In [77]:
```
print("The credit bureau score is a valuable metric in assessing the risk of a loan ap
As expected, those with higher scores tend to default less.\
The largest segment of the dataset has no available credit score, indicating potential
new credit users.\n")
d = loan.groupby(['PERFORM_CNS_SCORE_DESCRIPTION','loan_default']).count()['UniqueID']
print(d)
# Calculate the total count for each category
total_counts = d.groupby('PERFORM_CNS_SCORE_DESCRIPTION')['count'].transform('sum')

# Calculate the percentage
d['percentage'] = (d['count'] / total_counts) * 100
```

```
ax = sns.histplot(data=d , x='PERFORM_CNS_SCORE_DESCRIPTION', hue='loan_default', mult
                  palette=['#24b1d1', '#ae24d1'],
                  edgecolor='white',
                  shrink=0.8)
ax.set_title('Percentage of Loan Defaults by Credit Score Description')

# Create a custom y-axis formatter for percentages
def percent_formatter(x, pos):
    return f'{x:.0f}%'

# Apply the custom formatter to the y-axis labels
ax.yaxis.set_major_formatter(mtick.FuncFormatter(percent_formatter))

# Set the y-axis label
ax.set_ylabel('Percentage')

plt.xticks(rotation=45)  # Rotate x-axis labels if needed
plt.show()
```

The credit bureau score is a valuable metric in assessing the risk of a loan applican t.As expected, those with higher scores tend to default less.The largest segment of t he dataset has no available credit score, indicating potential gaps in data or a larg e number of new credit users.

	PERFORM_CNS_SCORE_DESCRIPTION	loan_default	count
0	high	0	15696
1	high	1	6239
2	low	0	57638
3	low	1	11384
4	medium	0	9239
5	medium	1	3173
6	not_available	0	89898
7	not_available	1	27052
8	not_scored	0	10072
9	not_scored	1	2763



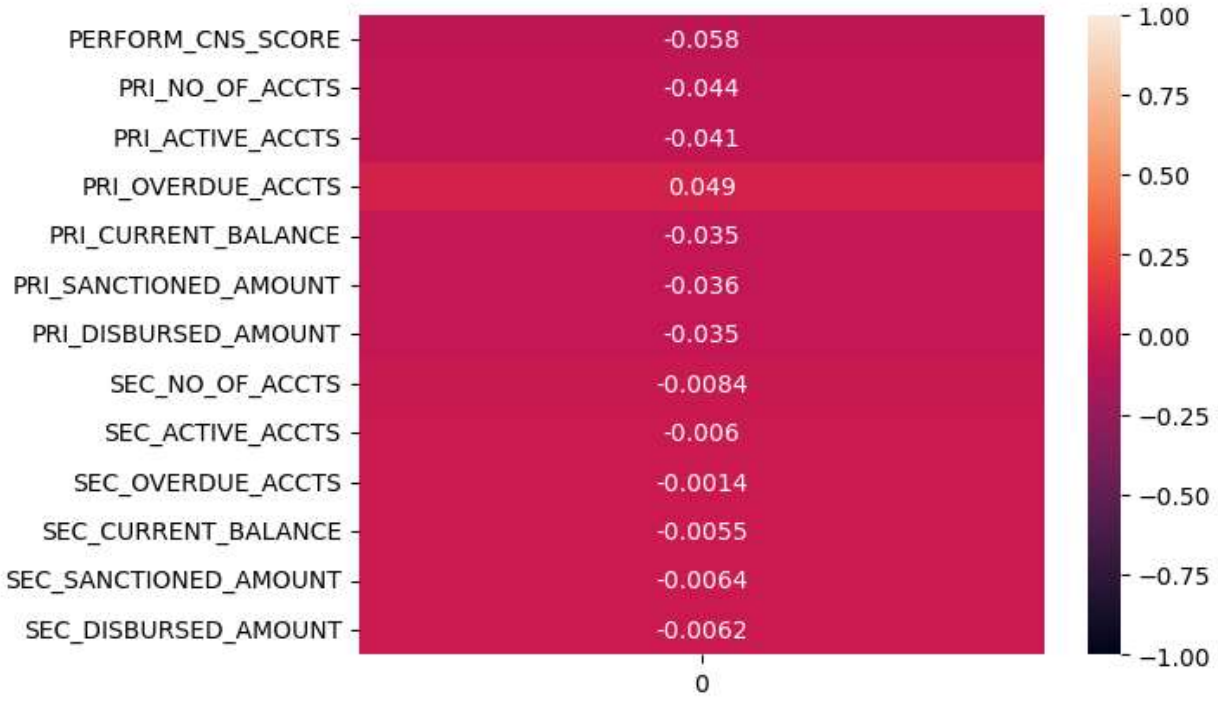
In [78]:
```
print("Most of the users withouht PERFORM_CNS_SCORE_DESCRIPTION are new credit users "
display(loan[loan['PERFORM_CNS_SCORE_DESCRIPTION']=='not_available'].AVERAGE_ACCT_AGE.
display(loan[loan['PERFORM_CNS_SCORE_DESCRIPTION']=='not_available'].PERFORM_CNS_SCORE
```

```
Most of the users withouht PERFORM_CNS_SCORE_DESCRIPTION are new credit users
0     115844
68       114
12        46
7         43
8         38
       ...
64         2
61         2
54         2
53         1
51         1
Name: AVERAGE_ACCT_AGE, Length: 67, dtype: int64
0    116950
Name: PERFORM_CNS_SCORE, dtype: int64
```

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

In [81]:
```
display(loan.iloc[:, 19:33].corrwith(loan['loan_default']))
sns.heatmap( loan.iloc[:, 19:33].corrwith(loan['loan_default']).to_frame() , vmin = -1
print ( "From the correlation analysis, it seems that primary account details have a s
 with loan default compared to secondary account details, though all the relationships
Specifically, the number of overdue primary accounts appears to be a factor that might
```

```
PERFORM_CNS_SCORE       -0.057929
PRI_NO_OF_ACCTS         -0.044472
PRI_ACTIVE_ACCTS        -0.041123
PRI_OVERDUE_ACCTS        0.049334
PRI_CURRENT_BALANCE     -0.035296
PRI_SANCTIONED_AMOUNT   -0.035503
PRI_DISBURSED_AMOUNT    -0.034505
SEC_NO_OF_ACCTS         -0.008385
SEC_ACTIVE_ACCTS        -0.005993
SEC_OVERDUE_ACCTS       -0.001371
SEC_CURRENT_BALANCE     -0.005531
SEC_SANCTIONED_AMOUNT   -0.006354
SEC_DISBURSED_AMOUNT    -0.006248
dtype: float64
From the correlation analysis, it seems that primary account details have a slightly
more considerable relationship with loan default compared to secondary account detail
s, though all the relationships are weak. Specifically, the number of overdue primary
accounts appears to be a factor that might increase the risk of default.
```



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

In [82]:
```
print("First, we look at the descriptive statistics for those variables.")
descriptive_stat = loan[['PRI_SANCTIONED_AMOUNT',  'PRI_DISBURSED_AMOUNT' , 'SEC_SANCT
print(descriptive_stat)
```

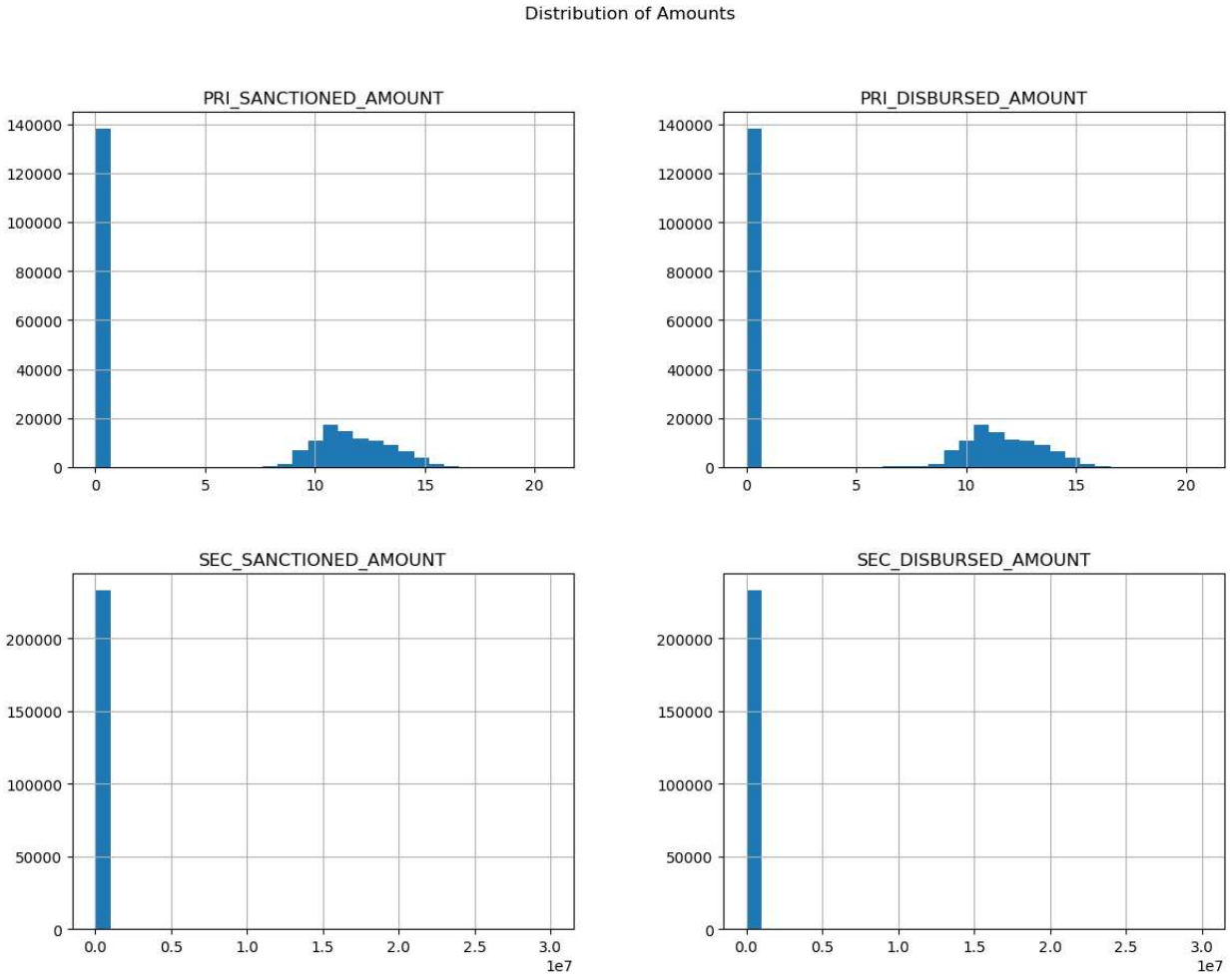
```
print("Looking at the mean values reveals a minor difference between the amounts sanct
disbursed for both Primary and Secondary accounts. The distribution of both sanctioned
for primary loans is highly skewed. This skewness is evident from the median (50th per
large standard deviation.")
```

```
First, we look at the descriptive statistics for those variables.
       PRI_SANCTIONED_AMOUNT  PRI_DISBURSED_AMOUNT  SEC_SANCTIONED_AMOUNT  \
count          233154.000000         233154.000000           2.331540e+05
mean                4.778708              4.761638           7.295923e+03
std                 5.869926              5.859382           1.831560e+05
min                 0.000000              0.000000           0.000000e+00
25%                 0.000000              0.000000           0.000000e+00
50%                 0.000000              0.000000           0.000000e+00
75%                11.042938             11.015362           0.000000e+00
max                20.723266             20.723266           3.000000e+07

       SEC_DISBURSED_AMOUNT
count          2.331540e+05
mean           7.179998e+03
std            1.825925e+05
min            0.000000e+00
25%            0.000000e+00
50%            0.000000e+00
75%            0.000000e+00
max            3.000000e+07
Looking at the mean values reveals a minor difference between the amounts sanctioned
and those actually disbursed for both Primary and Secondary accounts. The distributio
n of both sanctioned and disbursed amounts for primary loans is highly skewed. This s
kewness is evident from the median (50th percentile) being zero and the large standar
d deviation.
```

In [83]:
```
print("Let's check the average for the difference between PRI_SANCTIONED_AMOUNT and PR
print(np.mean([loan.PRI_SANCTIONED_AMOUNT  - loan.PRI_DISBURSED_AMOUNT]))
print("This value shows that the amount that was approved-sanctioned for primary loans
-disbursed by about 438 units, and they are not the same. " )

print("Now, let's check the average for the difference between SEC_SANCTIONED_AMOUNT a
print(round(np.mean([loan.SEC_SANCTIONED_AMOUNT  - loan.SEC_DISBURSED_AMOUNT]),2))
print("This value shows that the amount that was approved-sanctioned for secondary loa
disbursed by about 115 units, and they are not the same. " )
```

```
Let's check the average for the difference between PRI_SANCTIONED_AMOUNT and PRI_DISB
URSED_AMOUNT:
0.017070428977605405
This value shows that the amount that was approved-sanctioned for primary loans excee
ds the amount that was actually given out -disbursed by about 438 units, and they are
not the same.
Now, let's check the average for the difference between SEC_SANCTIONED_AMOUNT and SEC
_DISBURSED_AMOUNT:
115.93
This value shows that the amount that was approved-sanctioned for secondary loans exc
eeds the amount that was actually given out disbursed by about 115 units, and they ar
e not the same.
```

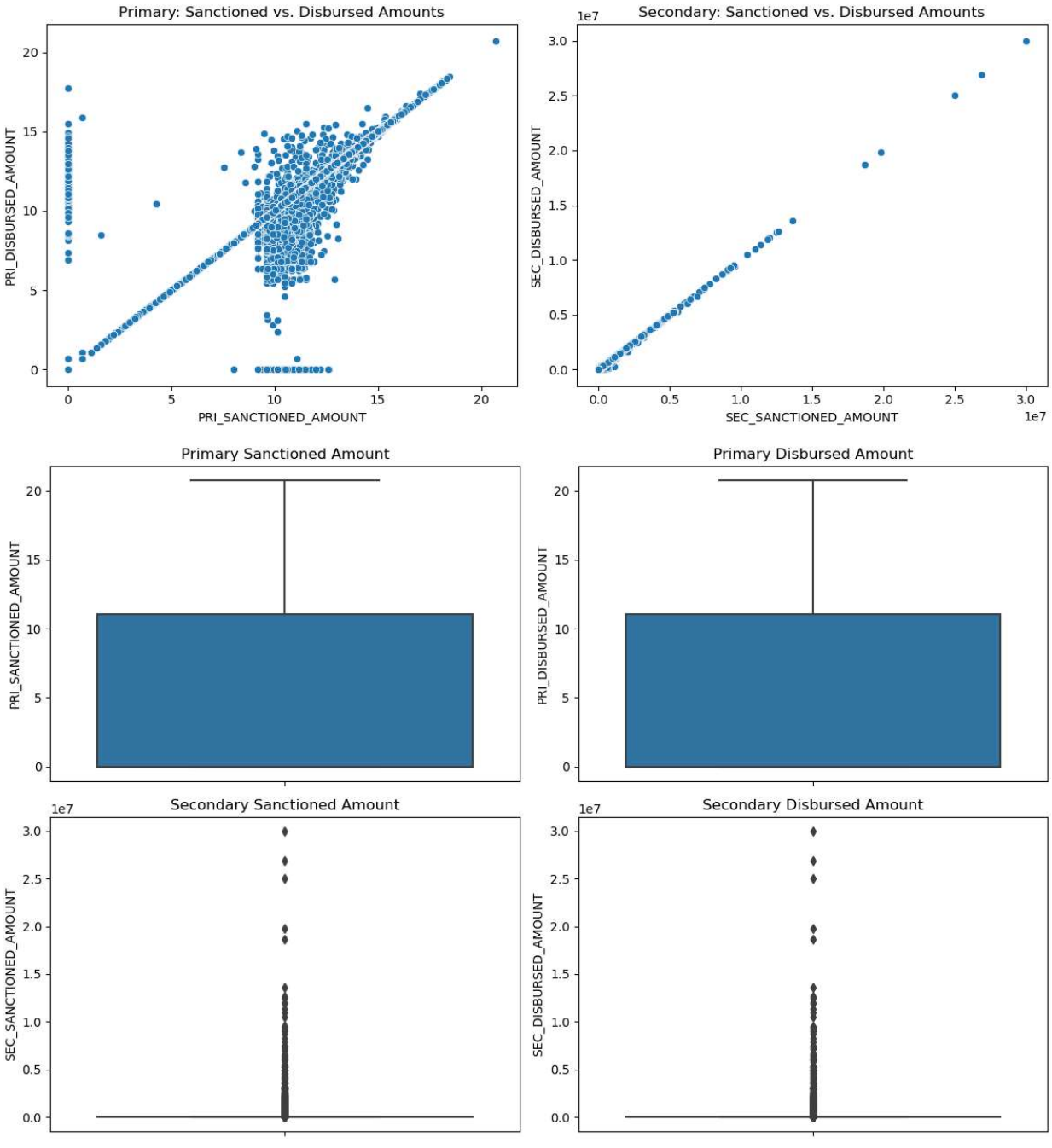
In [84]:
```
loan[['PRI_SANCTIONED_AMOUNT', 'PRI_DISBURSED_AMOUNT',
      'SEC_SANCTIONED_AMOUNT', 'SEC_DISBURSED_AMOUNT']].hist(figsize=(14, 10), bins=30
plt.suptitle('Distribution of Amounts')
plt.show()
```



In [85]:
```
print("Scatter plots show that the perfect correlation, sanctioned and disbursed amoun
Because they are so similar,we should only use one of them in our model to keep things
plt.figure(figsize=(12, 5))

# Primary Scatter Plot
plt.subplot(1, 2, 1)
ax=sns.scatterplot(x='PRI_SANCTIONED_AMOUNT', y='PRI_DISBURSED_AMOUNT', data=loan)
ax.set_title('Primary: Sanctioned vs. Disbursed Amounts')

# Secondary Scatter Plot
plt.subplot(1, 2, 2)
ax=sns.scatterplot(x='SEC_SANCTIONED_AMOUNT', y='SEC_DISBURSED_AMOUNT', data=loan)
ax.set_title('Secondary: Sanctioned vs. Disbursed Amounts')

plt.tight_layout()
plt.show()



plt.figure(figsize=(12, 8))

# Primary Boxplots
plt.subplot(2, 2, 1)
ax=sns.boxplot(y='PRI_SANCTIONED_AMOUNT', data=loan)
ax.set_title('Primary Sanctioned Amount')

plt.subplot(2, 2, 2)
ax=sns.boxplot(y='PRI_DISBURSED_AMOUNT', data=loan)
```

```
ax.set_title('Primary Disbursed Amount')

# Secondary Boxplots
plt.subplot(2, 2, 3)
ax=sns.boxplot(y='SEC_SANCTIONED_AMOUNT', data=loan)
ax.set_title('Secondary Sanctioned Amount')

plt.subplot(2, 2, 4)
ax=sns.boxplot(y='SEC_DISBURSED_AMOUNT', data=loan)
ax.set_title('Secondary Disbursed Amount')

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

Scatter plots show that the perfect correlation, sanctioned and disbursed amounts for loans move closely together. Because they are so similar,we should only use one of th em in our model to keep things clear and simple.

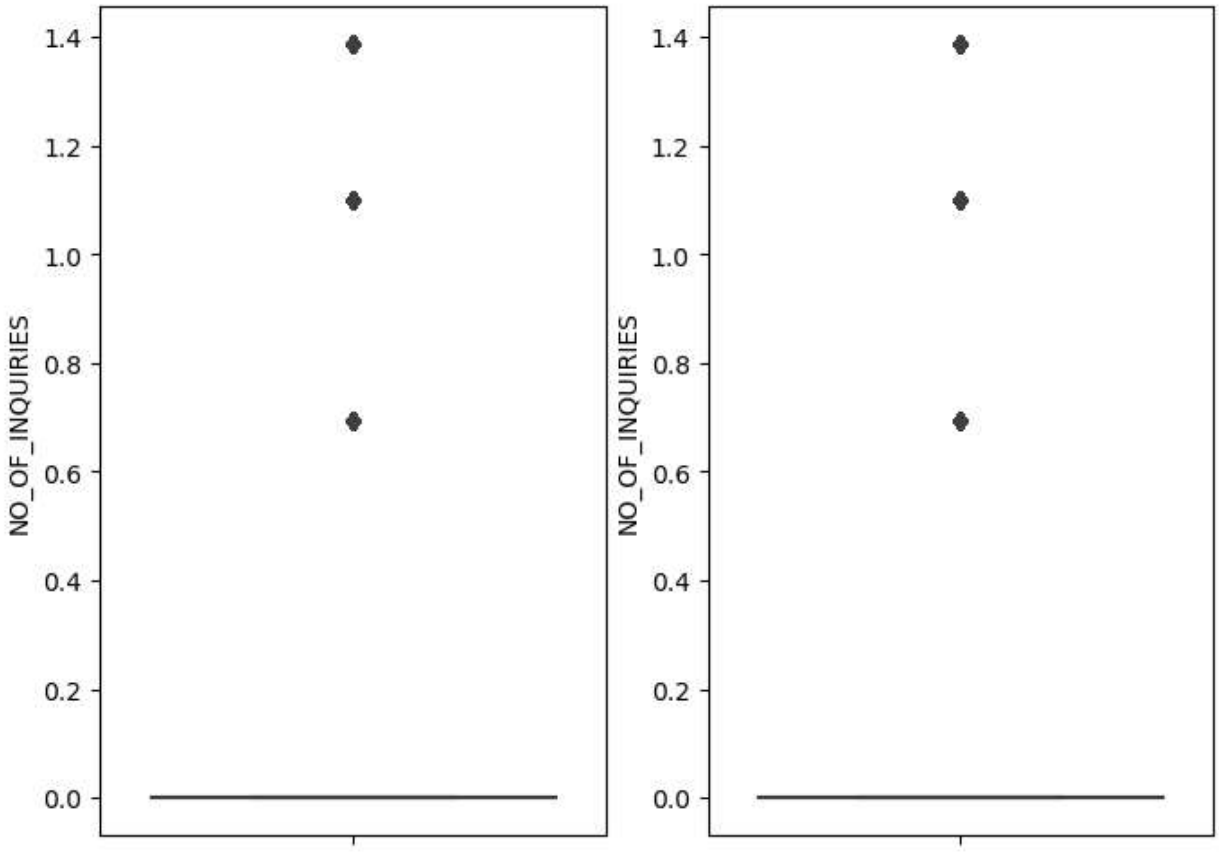


-Do customer who make higher number of enquiries end up being higher risk candidates?

In [86]:
```
print ( "Correlation between NO_OF_INQUIRIES  and Loan_Default: {}" .format (loan['NO_
fig, axes = plt.subplots(1, 2, figsize=(8, 6))
sns.boxplot(data = loan[loan.loan_default ==0] , y = 'NO_OF_INQUIRIES' , ax=axes[0])
sns.boxplot(data = loan[loan.loan_default ==1] , y = 'NO_OF_INQUIRIES' , ax=axes[1])
```

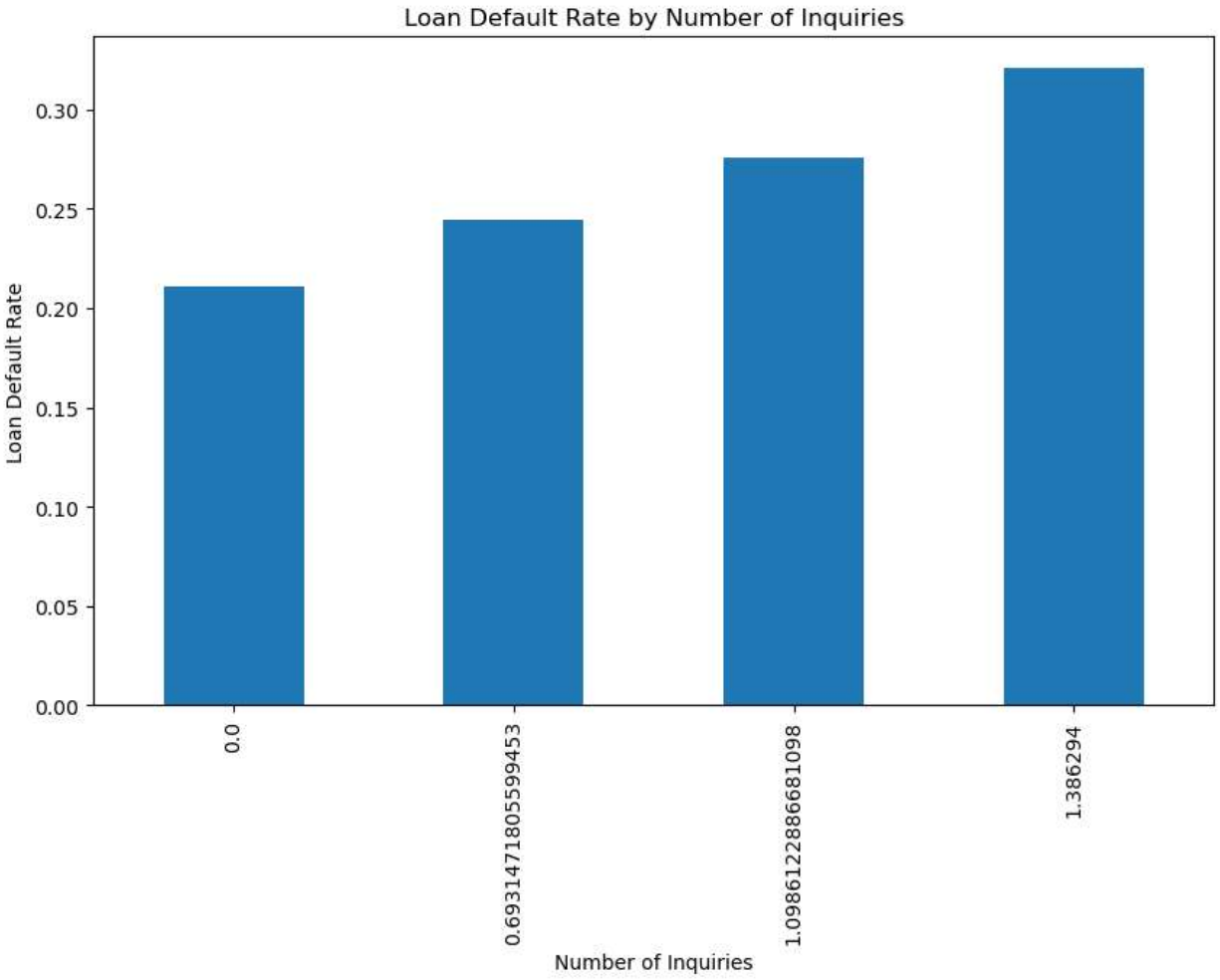
```
Correlation between NO_OF_INQUIRIES  and Loan_Default: 0.043983137086025685
```

Out[86]:
```
<AxesSubplot:ylabel='NO_OF_INQUIRIES'>
```



In [87]:
```
loan[loan.loan_default == 0 ].NO_OF_INQUIRIES.mean() , loan[loan.loan_default == 1  ].
inquiries_default_rate =loan.groupby('NO_OF_INQUIRIES')['loan_default'].mean()
print(inquiries_default_rate )
ax=inquiries_default_rate.plot(kind='bar', figsize=(10, 6))
ax.set_title('Loan Default Rate by Number of Inquiries')
plt.xlabel('Number of Inquiries')
plt.ylabel('Loan Default Rate')
plt.show()

print("Customers who make a higher number of inquiries indeed seem to have a higher li
 While there's a correlation, it doesn't necessarily mean causation.")
```

```
NO_OF_INQUIRIES
0.000000    0.210719
0.693147    0.244155
1.098612    0.275652
1.386294    0.320663
Name: loan_default, dtype: float64
```

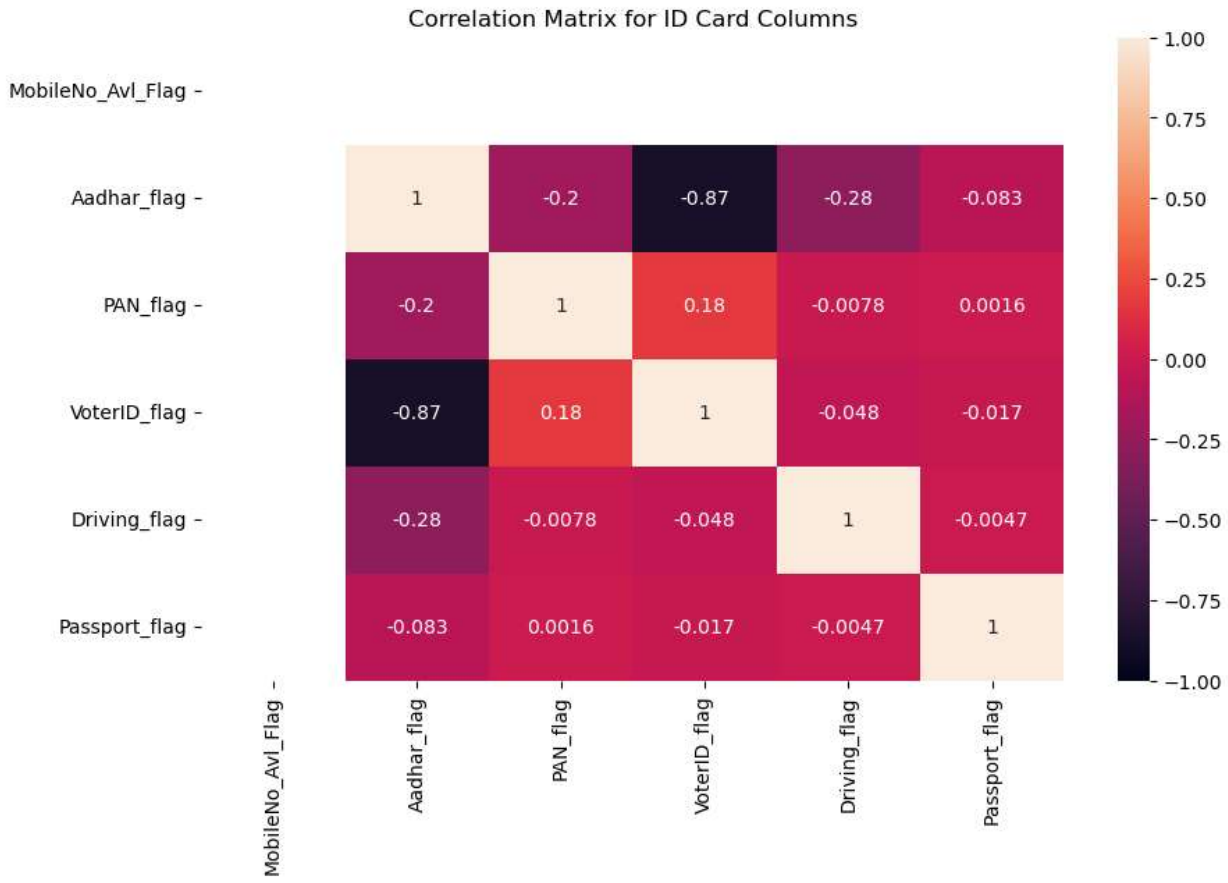



```
Customers who make a higher number of inquiries indeed seem to have a higher likeliho
od of defaulting on their loans While there's a correlation, it doesn't necessarily m
ean causation.
```

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

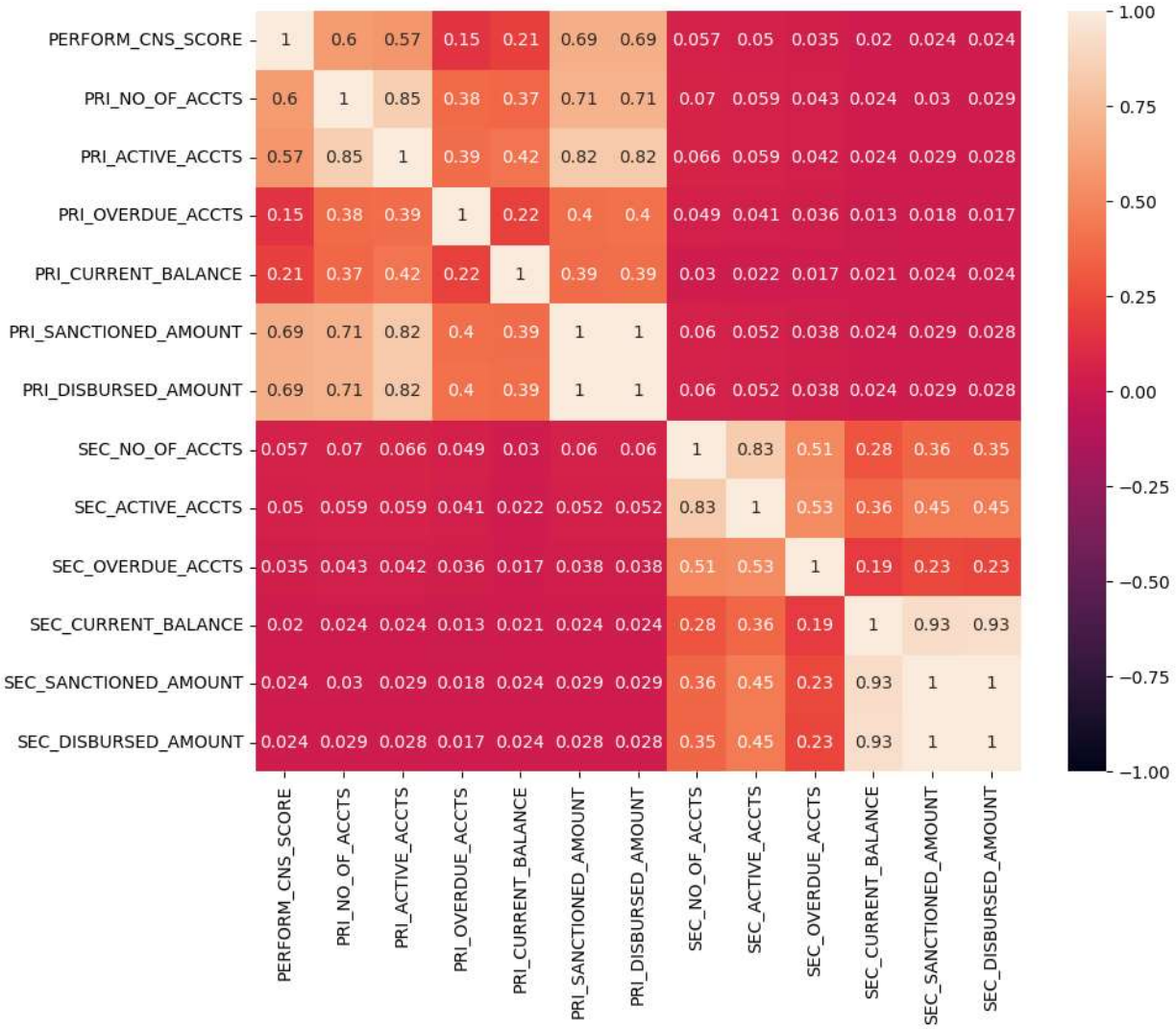
In [129…

```
id_columns = ['MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag', 'VoterID_flag', 'Driving
id_data = loan[id_columns]
correlation_matrix_id = id_data.corr()
plt.figure(figsize=(10,6))
ax=sns.heatmap(correlation_matrix_id, annot=True,  vmin=-1, vmax=1)
ax.set_title("Correlation Matrix for ID Card Columns")
plt.show()
print("Customers who have an Aadhar card are more likely to also have a Voter ID, and
Aadhar card are more likely not to have a Voter ID. I remove VoterID_flag from the mod
variables are saying the same thing.")
```

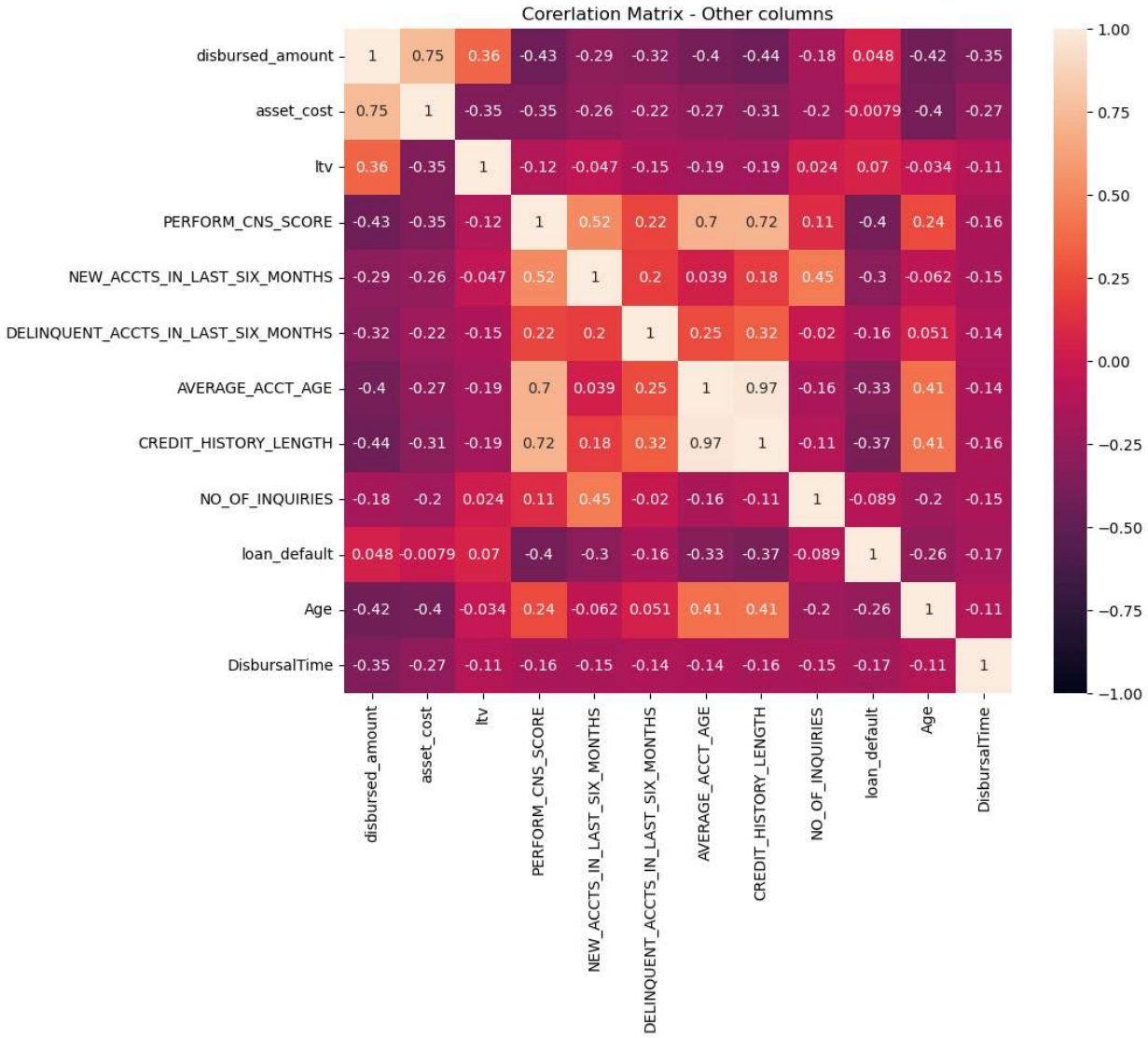


Customers who have an Aadhar card are more likely to also have a Voter ID, and customers without an Aadhar card are more likely not to have a Voter ID. I remove VoterID_flag from the model, since these two variables are saying the same thing.

In [89]:
```
loan_pri_sec_cols = loan.iloc[:,19:33]
loan_pri_sec_cols.corr()
plt.figure(figsize =(10,8))
sns.heatmap( loan_pri_sec_cols.corr(), vmin = -1 , vmax = 1 ,annot = True)
plt.title= "Corerlation Matrix - Primary and Secondary Account"
plt.show()
```



In [90]:
```
columns_to_exclude = ['UniqueID' ,'branch_id','supplier_id','manufacturer_id','Current
                      'State_ID','Employee_code_ID','VoterID_flag', 'Driving_flag', 'P
loan_for_corr = loan.drop(columns = columns_to_exclude)
loan_for_corr = loan_for_corr.drop(columns=loan_pri_sec_acc)
plt.figure(figsize =(10,8))
ax=sns.heatmap( loan_for_corr.corr().corr(), vmin = -1 , vmax = 1 ,annot = True)
ax.set_title("Corerlation Matrix - Other columns")
plt.show()
```



In [91]:
```
loan_for_corr.corr()
```

Out[91]:

	disbursed_amount	asset_cost	ltv	PERFORM_CNS_SC
disbursed_amount	1.000000	0.674991	0.472029	0.00
asset_cost	0.674991	1.000000	-0.299971	-0.05
ltv	0.472029	-0.299971	1.000000	0.08
PERFORM_CNS_SCORE	0.008275	-0.054552	0.084993	1.00
NEW_ACCTS_IN_LAST_SIX_MONTHS	0.034689	-0.031718	0.089991	0.41
DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS	0.019802	-0.012199	0.044544	0.18
AVERAGE_ACCT_AGE	-0.002443	-0.023723	0.032715	0.52
CREDIT_HISTORY_LENGTH	0.007394	-0.030683	0.054847	0.49
NO_OF_INQUIRIES	0.048842	-0.026510	0.105196	0.17
loan_default	0.090077	0.015899	0.098208	-0.05
Age	-0.059881	-0.127019	0.081120	0.16
DisbursalTime	-0.093371	-0.096920	0.012612	0.00

Export prepared data to .csv for use in Power BI to create a dashboard</Span>

In [45]:
```
loan.to_csv("../../02 Data/Prepared Data/Loan_PreparedData.csv")
```

In [134…
```
len(loan.columns)
```

Out[134]:
```
44
```

Perform logistic regression modeling, predict the outcome for the test data, and validate the results using the confusion matrix.</Span>

Base Model

In [124…
```
loan_encoded_base = pd.get_dummies(loan, columns=['Employment_Type', 'PERFORM_CNS_SCOR
loan_encoded_base = loan_encoded_base.drop('UniqueID',axis=1)
loan_encoded_base = loan_encoded_base.drop('DisbursalDate',axis=1)
loan_encoded_base = loan_encoded_base.drop('Date_of_Birth',axis=1)
```

In [125…
```
# Splitting data into training and test sets
X = loan_encoded_base.drop(columns='loan_default')  # Assuming 'loan_default' is your
y = loan_encoded_base['loan_default']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

```
# Model training

# Scaling the data (important for logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize and train the logistic regression model
model_base = LogisticRegression(max_iter=1000)
model_base.fit(X_train_scaled, y_train)
```

Out[125]:
```
LogisticRegression(max_iter=1000)
```

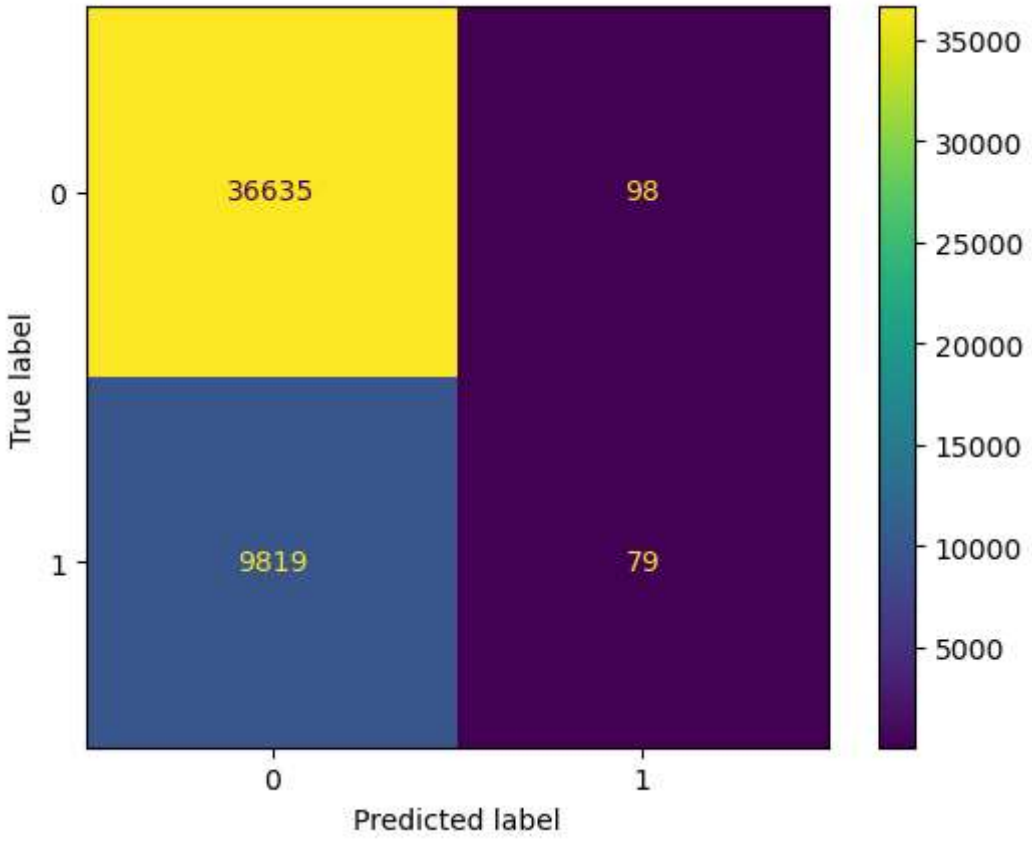
In [126…
```
# Evaluation
# Predict on the test set
y_pred = model_base.predict(X_test_scaled)

# Print the classification report
print(classification_report(y_test, y_pred))

# Print accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      1.00      0.88     36733
           1       0.45      0.01      0.02      9898

    accuracy                           0.79     46631
   macro avg       0.62      0.50      0.45     46631
weighted avg       0.72      0.79      0.70     46631

Accuracy: 0.7873303167420814
```

In [128…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Predict on the test set
y_pred = model_base.predict(X_test_scaled)

# Get the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print(cm)

# Optionally, you can visualize the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model_base.classes_)
disp.plot()
```

```
[[36635    98]
 [ 9819    79]]
```

Out[128]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a21781d1f0>
```



The model has a very high number of True Negatives, which means it is good at identifying non-defaulted loans. The number of True Positives is very low, indicating that the model struggles to correctly identify defaulted loans. The number of False Negatives is significantly high, which means the model is predicting many loans as non-defaulted when they actually are defaulted. We may need to handle imbalanced datasets to improve performance, especially if identifying defaults is crucial.

Building a Model with a Subset of Data Columns

In [112…

```
# with s subset of columns
columns_to_exclude_from_model = ['Age','PRI_SANCTIONED_AMOUNT','SEC_NO_OF_ACCTS','SEC_
                                 'SEC_CURRENT_BALANCE','SEC_SANCTIONED_AMOUNT','SEC_DI
loan_encoded_subset_subset_of_cols= loan_encoded_base.drop(columns=columns_to_exclude_
```

In [113…

```
# Splitting data into training and test sets
X = loan_encoded_subset_subset_of_cols.drop(columns='loan_default')  # Assuming 'loan_
y = loan_encoded_subset_subset_of_cols['loan_default']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=


# Model training

# Scaling the data (important for logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

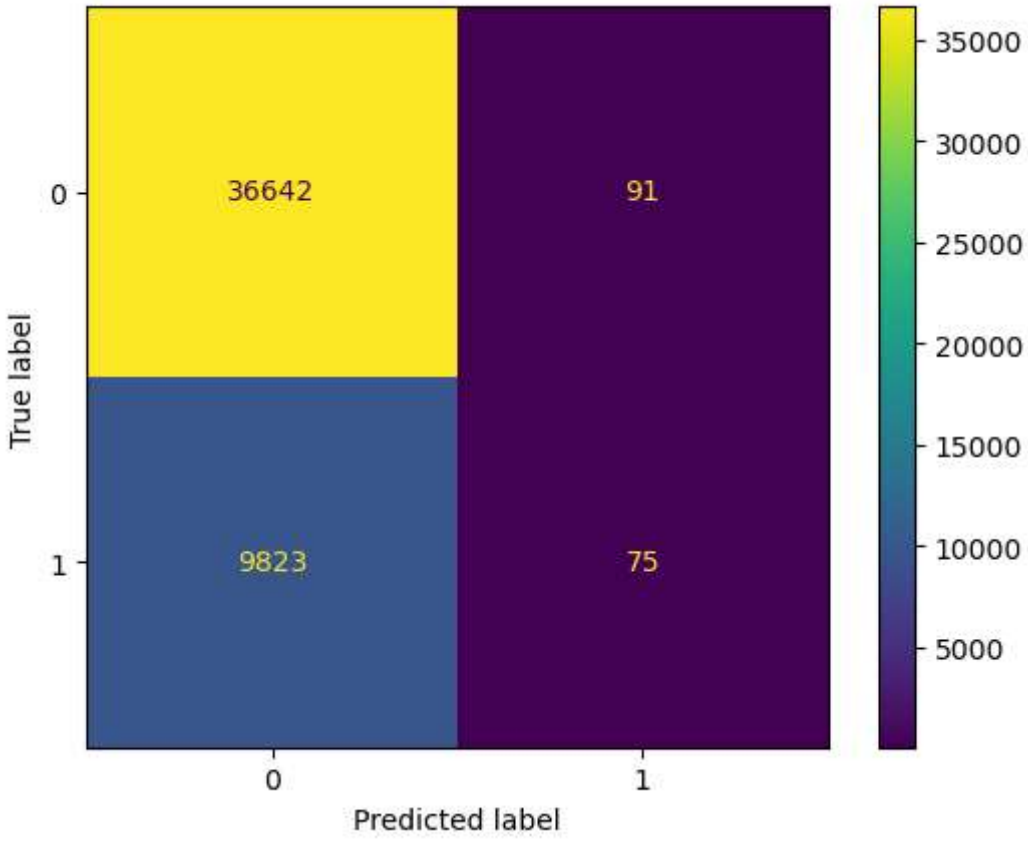
```
# Initialize and train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)
```

Out[113]:
```
LogisticRegression(max_iter=1000)
```

In [114…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Get the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print(cm)

# Optionally, you can visualize the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
disp.plot()
```

```
[[36642    91]
 [ 9823    75]]
```

Out[114]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a21e0b32e0>
```



In [103…
```
# Evaluation
# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Print the classification report
print(classification_report(y_test, y_pred))
```

```python
# Print accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      1.00      0.88     36733
           1       0.45      0.01      0.01      9898

    accuracy                           0.79     46631
   macro avg       0.62      0.50      0.45     46631
weighted avg       0.72      0.79      0.70     46631

Accuracy: 0.7873946516266004
```