🏭 Data Pre-processing, EDA, Feature Engineering 🔧 & Modeling 📉

on Vehicle Loan Default Dataset ¶

👋

🤔

👉 Financial institutions have seen significant losses due to Default of Vehicle Loans

👉 This has led to tightening up of vehicle Loan Underwriting and Increased Vehicle loan rejection rates

👉 **Accurately predict probability of loanee/borrower defaulting on a vehicle loan in first EMI-(Equated Monthly Instalments) on due date**

👉 Data pre-processing and feature engineering will be performed to prepare the dataset before it is used by the machine learning model.

🤔

👉 **Possible factors which might effect our target variable, i.e whether a Customer will default or not**

- **Credit History**
- **Employment Status**
  - Self Employed
  - Salaried -- lower chances of default
- **Loanee Age**
- **Loanee Income/Salary**
- **Loan Amount**
- **Loan Tenure**
- **Loan to Value Ratio**
- **Loan to Incom Ratio**.....so on

👉 **Demographic factors** also play vital role too, this list can go up to 5 more or 50more maybe 5000 more based on valid Hypothesis will collect data now

🤔

👉 Data Source - 1 Lonee Information

👉 Data Source - 2 Loan Information

👉 Data Source - 3 Bureau Data & Credit History

👉 For a Bank which is new there data can be collected from `Central Bureau `, every county have 1. They collect all transational information of evry in dividul WHO have credit history based on this data they provide `civil score ` or `fico score` or `crif score` or `equifax score` etc...

🤔

👉 Loanee Information (Demographic data like age, Identity proof etc)

👉 Loan Information (Disbursal details, loan to value ratio etc)

👉 Bureau data & History (Bureau score, number of active accounts, the status of other loans, credit history etc)

👉 **Going through all this data will ensure that clients/loanee capable of repayment are not rejected and important determinants can be identified which can be further used for minimising default rates**

👉 **Data Dictionary** of Colected Data

Variable Name	Description
UniqueID	Identifier for customers
loan_default	Payment default in the first EMI on due date
disbursed_amount	Amount of Loan disbursed
asset_cost	Cost of the Asset
ltv	Loan to Value of the asset
branch_id	Branch where the loan was disbursed
supplier_id	Vehicle Dealer where the loan was disbursed
manufacturer_id	Vehicle manufacturer(Hero, Honda, TVS etc.)
Current_pincode	Current pincode of the customer
Date.of.Birth	Date of birth of the customer
Employment.Type	Employment Type of the customer (Salaried/Self Employed)
DisbursalDate	Date of disbursement
State_ID	State of disbursement
Employee_code_ID	Employee of the organization who logged the disbursement
MobileNo_Avl_Flag	if Mobile no. was shared by the customer then flagged as 1
Aadhar_flag	if aadhar was shared by the customer then flagged as 1
PAN_flag	if pan was shared by the customer then flagged as 1
VoterID_flag	if voter was shared by the customer then flagged as 1
Driving_flag	if DL was shared by the customer then flagged as 1
Passport_flag	if passport was shared by the customer then flagged as 1

Variable Name	Description
PERFORM_CNS.SCORE	Bureau Score
PERFORM_CNS.SCORE.DESCRIPTION	Bureau score description
PRI.NO.OF.ACCTS	count of total loans taken by the customer at the time of disbursement
PRI.ACTIVE.ACCTS	count of active loans taken by the customer at the time of disbursement
PRI.OVERDUE.ACCTS	count of default accounts at the time of disbursement
PRI.CURRENT.BALANCE	total Principal outstanding amount of the active loans at the time of disbursement
PRI.SANCTIONED.AMOUNT	total amount that was sanctioned for all the loans at the time of disbursement
PRI.DISBURSED.AMOUNT	total amount that was disbursed for all the loans at the time of disbursement
SEC.NO.OF.ACCTS	count of total loans taken by the customer at the time of disbursement
SEC.ACTIVE.ACCTS	count of active loans taken by the customer at the time of disbursement
SEC.OVERDUE.ACCTS	count of default accounts at the time of disbursement
SEC.CURRENT.BALANCE	total Principal outstanding amount of the active loans at the time of disbursement
SEC.SANCTIONED.AMOUNT	total amount that was sanctioned for all the loans at the time of disbursement
SEC.DISBURSED.AMOUNT	total amount that was disbursed for all the loans at the time of disbursement
PRIMARY.INSTAL.AMT	EMI Amount of the primary loan
SEC.INSTAL.AMT	EMI Amount of the secondary loan
NEW.ACCTS.IN.LAST.SIX.MONTHS	New loans taken by the customer in last 6 months before the disbursment
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS	Loans defaulted in the last 6 months
AVERAGE.ACCT.AGE	Average loan tenure
CREDIT.HISTORY.LENGTH	Time since first loan
NO.OF_INQUIRIES	Enquries done by the customer for loans

👉 **Primary accounts are those which customer has taken for his personal use**

👉 **Secondary accounts are those which customer act as a co-applicant or gaurantor**

Features in capital letters are part of Bureau Data:

1. `PRI.` ==> primary account i.e. loanee was applicant
2. `SEC.` ==> loanee was co-applicant , kind of joint loan

👉 **Lonee Information-Features**

Variable Name	Description
State_ID	State of disbursement
Employee_code_ID	Employee of the organization who logged the disbursement
MobileNo_Avl_Flag	if Mobile no. was shared by the customer then flagged as 1
Aadhar_flag	if aadhar was shared by the customer then flagged as 1
PAN_flag	if pan was shared by the customer then flagged as 1
VoterID_flag	if voter was shared by the customer then flagged as 1
Driving_flag	if DL was shared by the customer then flagged as 1

Variable Name	Description
Passport_flag	if passport was shared by the customer then flagged as 1

👉 Loan Information-Features

Variable Name	Description
UniqueID	Identifier for customers
loan_default	Payment default in the first EMI on due date
disbursed_amount	Amount of Loan disbursed
asset_cost	Cost of the Asset
ltv	Loan to Value of the asset
branch_id	Branch where the loan was disbursed
supplier_id	Vehicle Dealer where the loan was disbursed
manufacturer_id	Vehicle manufacturer(Hero, Honda, TVS etc.)
Current_pincode	Current pincode of the customer
Date.of.Birth	Date of birth of the customer
Employment.Type	Employment Type of the customer (Salaried/Self Employed)
DisbursalDate	Date of disbursement

👉 Beauro Information-Features

Variable Name	Description
PERFORM_CNS.SCORE	Bureau Score
PERFORM_CNS.SCORE.DESCRIPTION	Bureau score description
PRI.NO.OF.ACCTS	count of total loans taken by the customer at the time of disbursement
PRI.ACTIVE.ACCTS	count of active loans taken by the customer at the time of disbursement
PRI.OVERDUE.ACCTS	count of default accounts at the time of disbursement
PRI.CURRENT.BALANCE	total Principal outstanding amount of the active loans at the time of disbursement
PRI.SANCTIONED.AMOUNT	total amount that was sanctioned for all the loans at the time of disbursement
PRI.DISBURSED.AMOUNT	total amount that was disbursed for all the loans at the time of disbursement
SEC.NO.OF.ACCTS	count of total loans taken by the customer at the time of disbursement
SEC.ACTIVE.ACCTS	count of active loans taken by the customer at the time of disbursement
SEC.OVERDUE.ACCTS	count of default accounts at the time of disbursement
SEC.CURRENT.BALANCE	total Principal outstanding amount of the active loans at the time of disbursement
SEC.SANCTIONED.AMOUNT	total amount that was sanctioned for all the loans at the time of disbursement
SEC.DISBURSED.AMOUNT	total amount that was disbursed for all the loans at the time of disbursement

👉 **This notebook aims to:**

- Perform **Data Exploration**
  - Observe data dict and data samples
  - Segregate variables by there Data Types
  - Univariate Categorical Analysis
  - Univarite Numerical Analysis
  - Bivariate Categorical-Categorical Analysis
  - Bivariate Continuous-Categorical Analysis
- Perform **Data Pre-Processing**
- Perform **EDA** and **Hypothesis Testing (statistical and non-statistical)** in cleaned data set
- Perform **Feature Engineering**

👉 **Importing libraries** that will be used in this notebook.

In [1]:

```
import numpy as np
import pandas as pd

#data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Ignore Warnings
import warnings
warnings.filterwarnings('ignore')
#see max columns
pd.set_option('display.max_columns', None)

sns.set_style('whitegrid')
plt.rcParams['figure.dpi']=100
```

👉 After importing libraries, **dataset that will be used will be imported**

In [2]:

```
path = 'datasets/bank-loan-data/'
train = pd.read_csv(path + 'train.csv')
test  = pd.read_csv(path + 'test.csv')
```

In [3]:

```
# --- Reading Train Dataset ---
train.head(10).style.background_gradient(cmap='Blues').set_properties(**{'font-famil
```

Out[3]:

UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Cu
420825	50578	58400	89.550000	67	22807	45	
537409	47145	65550	73.230000	67	22807	45	
417566	53278	61360	89.630000	67	22807	45	
624493	57513	66113	88.480000	67	22807	45	
539055	52378	60300	88.390000	67	22807	45	
518279	54513	61900	89.660000	67	22807	45	
529269	46349	61500	76.420000	67	22807	45	
510278	43894	61900	71.890000	67	22807	45	
490213	53713	61973	89.560000	67	22807	45	
510980	52603	61300	86.950000	67	22807	45	

👉 See all features , **descriptions** as there are more than 39 feature

👉 Merge Data Dictionary with dataset , **to get better intuition**

👉 AIM: To see sample of each feature to get some intuition woll merge Data Dictionary with dataset , **to get better intuition**

In [4]:

```
!pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.0-py2.py3-none-any.whl (250 kB)
     |████████████████████████████████| 250 kB 4.5 MB/s
Collecting et-xmlfile
  Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.0
WARNING: Running pip as the 'root' user can result in broken permissio
ns and conflicting behaviour with the system package manager. It is re
commended to use a virtual environment instead: https://pip.pypa.io/wa
rnings/venv (https://pip.pypa.io/warnings/venv)
```

In [5]:

```
#reading data dict
data_dic = pd.read_excel(path + 'data_dict.xlsx')
```

In [6]:

```
#quick eyeballing data
any_5_sample = pd.DataFrame(train.sample(5).T).reset_index()

#renaming first column from index to Variable name
any_5_sample.rename(columns = {'index':'Variable Name'},inplace=True)
```

In [7]:

```
# merging data dictionary with features to make a solid understanding
sample_with_discription = data_dic[['Variable Name','Description']].merge(any_5_samp
```

In [8]:

```
sample_with_discription.style.set_properties(**{'font-family': 'Segoe UI'}).hide_inc
```

Out[8]:

Variable Name	Description	116753	158930	128362	16430	60206
UniqueID	Identifier for customers	483690	645051	474272	528018	42146
loan_default	Payment default in the first EMI on due date	1	0	1	1	0
disbursed_amount	Amount of Loan disbursed	45349	52578	51303	47349	5791
asset_cost	Cost of the Asset	65000	69621	69182	74036	6743
ltv	Loan to Value of the asset	72.310000	78.280000	76.610000	66.180000	87.49000
branch_id	Branch where the loan was disbursed	136	29	77	10	13
supplier_id	Vehicle Dealer where the loan was	24159	23939	24287	22482	2223

Eyebolling data

- **branch_id** name is not given, Anonymized
- **supplier_id** (vehicle dealer)name is not given, Anonymized
- **manufacturer_id** name is not given, Anonymized
- **Current_pincode** NaN values can be seen
- **Date.of.Birth** last two number of year are given -- short form year like 22 can be 2022 or 1922
- **DisbursalDate** last two number of year are given -- short form year like 22 can be 2022 or 1922
- **PRI** used for primary account
- **SEC** used for secondary account
- **DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS** losn default in last 6 month very important feature

Conclusion👓

- We first have some loan features
- some features related to branch, supplier and manufacturer of asset then
- more information on loan itself such as employee who disbursed loan, flags on what ID proofs were available
- some bureau data information

LTV -- Loan to Value of asset || idealy formula becomes || sometime other charges can be added to it

$$LTV = \frac{disbursedAmount}{assetCost}$$

In [9]:

```
def dataset_shape_info(dataset):
    # --- Print Dataset Info ---
    print('\033[36m\033[1m'+'.: Dataset Info :.')
    print('\033[0m\033[36m*' * 20)
    print('\033[0m'+'Total Rows:'+'\033[36m\033[1m', dataset.shape[0])
    print('\033[0m'+'Total Columns:'+'\033[36m\033[1m', dataset.shape[1])
    print('\033[0m\033[36m*' * 20)
    print('\n')

# --- Print Dataset Detail ---
    print('\033[1m'+'.: Dataset Details :.')
    print('\033[0m\033[36m*' * 22 +'\033[0m')
    dataset.info(memory_usage = False)
```

In [10]:

```
dataset_shape_info(train)
```

```
.: Dataset Info :.
********************
Total Rows: 233154
Total Columns: 41
********************


.: Dataset Details :.
**********************
<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  object
 9   Employment.Type                      225493 non-null  object
 10  DisbursalDate                        233154 non-null  object
 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: float64(1), int64(34), object(6)
```

In [11]:

```
dataset_shape_info(test)
```

```
.: Dataset Info :.
********************
Total Rows: 112392
Total Columns: 40
********************


.: Dataset Details :.
**********************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112392 entries, 0 to 112391
Data columns (total 40 columns):
 #   Column                               Non-Null Count   Dtype
---  ------                               --------------   -----
 0   UniqueID                             112392 non-null  int64
 1   disbursed_amount                     112392 non-null  int64
 2   asset_cost                           112392 non-null  int64
 3   ltv                                  112392 non-null  float64
 4   branch_id                            112392 non-null  int64
 5   supplier_id                          112392 non-null  int64
 6   manufacturer_id                      112392 non-null  int64
 7   Current_pincode_ID                   112392 non-null  int64
 8   Date.of.Birth                        112392 non-null  object
 9   Employment.Type                      108949 non-null  object
 10  DisbursalDate                        112392 non-null  object
 11  State_ID                             112392 non-null  int64
 12  Employee_code_ID                     112392 non-null  int64
 13  MobileNo_Avl_Flag                    112392 non-null  int64
 14  Aadhar_flag                          112392 non-null  int64
 15  PAN_flag                             112392 non-null  int64
 16  VoterID_flag                         112392 non-null  int64
 17  Driving_flag                         112392 non-null  int64
 18  Passport_flag                        112392 non-null  int64
 19  PERFORM_CNS.SCORE                    112392 non-null  int64
 20  PERFORM_CNS.SCORE.DESCRIPTION        112392 non-null  object
 21  PRI.NO.OF.ACCTS                      112392 non-null  int64
 22  PRI.ACTIVE.ACCTS                     112392 non-null  int64
 23  PRI.OVERDUE.ACCTS                    112392 non-null  int64
 24  PRI.CURRENT.BALANCE                  112392 non-null  int64
 25  PRI.SANCTIONED.AMOUNT                112392 non-null  int64
 26  PRI.DISBURSED.AMOUNT                 112392 non-null  int64
 27  SEC.NO.OF.ACCTS                      112392 non-null  int64
 28  SEC.ACTIVE.ACCTS                     112392 non-null  int64
 29  SEC.OVERDUE.ACCTS                    112392 non-null  int64
 30  SEC.CURRENT.BALANCE                  112392 non-null  int64
 31  SEC.SANCTIONED.AMOUNT                112392 non-null  int64
 32  SEC.DISBURSED.AMOUNT                 112392 non-null  int64
 33  PRIMARY.INSTAL.AMT                   112392 non-null  int64
 34  SEC.INSTAL.AMT                       112392 non-null  int64
 35  NEW.ACCTS.IN.LAST.SIX.MONTHS         112392 non-null  int64
 36  DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS  112392 non-null  int64
 37  AVERAGE.ACCT.AGE                     112392 non-null  object
 38  CREDIT.HISTORY.LENGTH                112392 non-null  object
 39  NO.OF_INQUIRIES                      112392 non-null  int64
dtypes: float64(1), int64(33), object(6)
```

🔠

👉 There are different ways to deel and explore both (Continious & Catergorical) variables so

👉 Segrigating them is very important

In [12]:

```
#if category exist then there distribution and nuniques in general
print('\033[36m*' * 29+'\033[0m')
for each_column in train.columns:
    each_column_nunique = train[each_column].nunique()
    print(f"Column: [ {each_column} ] nuniques are: {each_column_nunique}\n")

    #if category are less then 3 print them with count
    if each_column_nunique <= 3:
        print(sorted(train[each_column].value_counts().to_dict().items()),'\n')
    print('\033[36m*' * 29+'\033[0m')
```

```
*****************************
Column: [ UniqueID ] nuniques are: 233154

*****************************
Column: [ disbursed_amount ] nuniques are: 24565

*****************************
Column: [ asset_cost ] nuniques are: 46252

*****************************
Column: [ ltv ] nuniques are: 6579

*****************************
Column: [ branch_id ] nuniques are: 82

*****************************
Column: [ supplier_id ] nuniques are: 2953

*****************************
```

EDA and feature engineering

1. Separating Categorical Feature
2. Separating Numerical Features
3. Separating Date Time Features
4. Separating ID Features

In [13]:

```
#target
target = train['loan_default']

###Features
target_features = ['loan_default']

id_features = ['UniqueID','branch_id','supplier_id','Current_pincode','Employee_code

date_time_features = ['Date.of.Birth','DisbursalDate','CREDIT.HISTORY.LENGTH','AVERA

categorical_features = ['Employment.Type','PERFORM_CNS.SCORE.DESCRIPTION','manufactu
                        'Aadhar_flag','PAN_flag','VoterID_flag','Driving_flag','Pass

segrigated_features = target_features + id_features + date_time_features + categoric
numerical_features = [particular_colum for particular_colum in train.columns if part
```

NOTE

We will EDA now based on these features and will try to get every required information out of these features

In [14]:

```
pd.isnull(train['branch_id']).sum()
```

Out[14]:

```
0
```

In [15]:

```
import matplotlib.ticker as ticker
import seaborn as sns
```

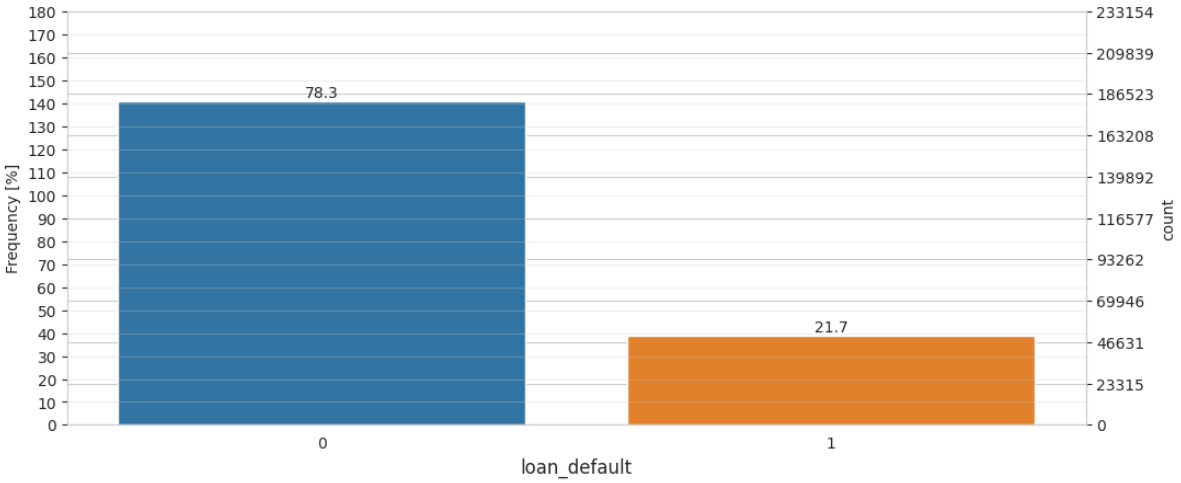
In [16]:

```
#function to visualise { Categorical VAriables }
def cat_var_eda(data,features):
    """
    [ Univariate Analysis For Categorical Variables ]
        Will take a group of variables-[CATEGORY] and will plot or print:
            1. Bar Plot
            2. value_counts
    """
    #setting figure size and all
    fig_size = len(features)
    fig = plt.figure(figsize=(12*fig_size,5),dpi=100)
    fig.canvas.draw()
    plt.tight_layout(pad=1.08,h_pad=None,w_pad=100000,rect=None)

    for index,values in enumerate(features):
        plt.subplot(1, fig_size, index + 1)
        ax = sns.countplot(x=values, data=data, orient='h') #horizontal countplot
        plt.xlabel(f"{values}", fontsize=12)

        not_null_count = data.shape[0] - pd.isnull(data[values]).sum()

        #twin axis formation
        ax2 = ax.twinx()
        #count axis on right, frequency on left
        ax2.yaxis.tick_left()
        ax.yaxis.tick_right()
        #also switching labels over
        ax.yaxis.set_label_position('right')
        ax2.yaxis.set_label_position('left')

        ax2.set_ylabel('Frequency [%]')

        for p in ax.patches:
            x = p.get_bbox().get_points()[:,0]
            y = p.get_bbox().get_points()[1,1]
            ax.annotate('{:.1f}'.format(100. * y / not_null_count),
                        (x.mean(),y), ha='center', va='bottom') #setting text allign

        #using a LinearLocator to ensure correct number of ticks
        ax.yaxis.set_major_locator(ticker.LinearLocator(11))
        ax.set_ylim(0,not_null_count)
        ax2.set_ylim(0,180)
        #tick spacing of 10
        ax2.yaxis.set_major_locator(ticker.MultipleLocator(10))
        ax2.grid(alpha=0.3)

        plt.show
```

loan_default --> Target

In [17]:

```
#target variable
cat_var_eda(train,['loan_default'])
```



In [18]:

```
train.shape[0]
```

Out[18]:

```
233154
```

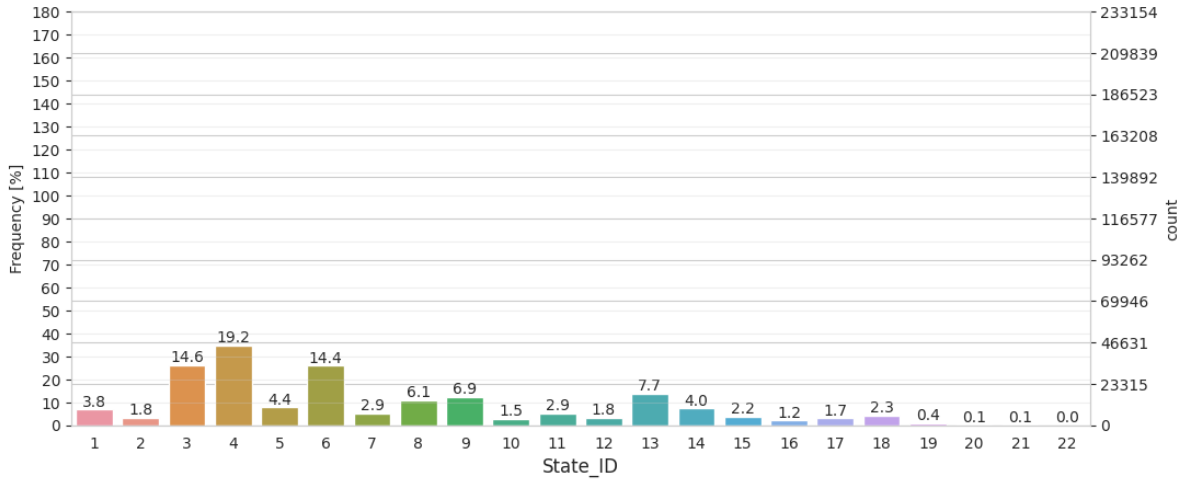
Imbalanced classification as expected from a loan default use case

- 21.7% ==> defaulter out of 233154 customers of train data

State_ID

In [19]:

```
#State_ID
cat_var_eda(train,['State_ID'])
```



Obsevation :

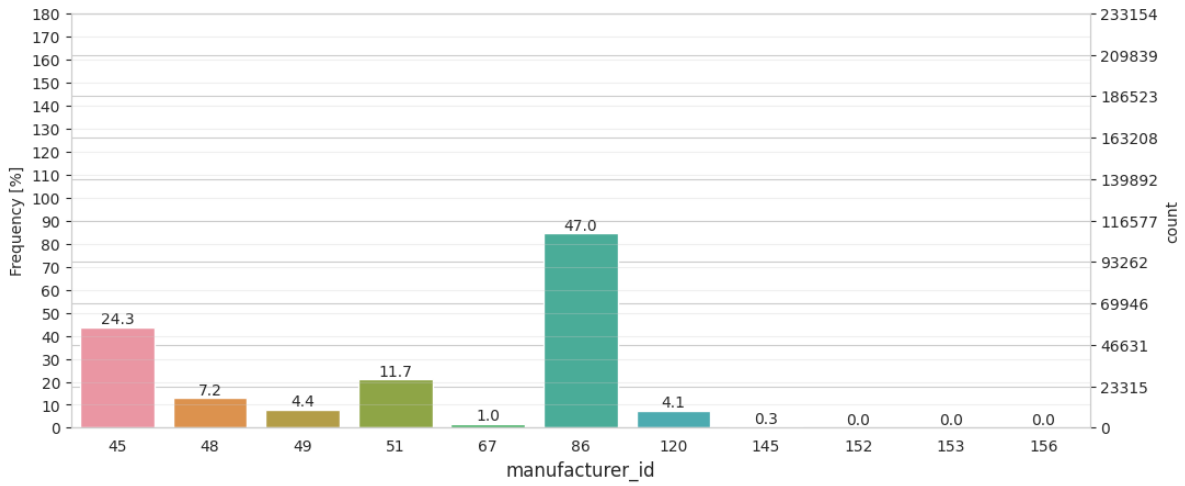
Most loan are taken from customer belonging to state 4,3,6,13,9 and 8

As most of the earning comes out of some of the state as not all state have same earning and industry and stufff stuff.

manufacturer_id

In [20]:

```
#manufacturer_id
cat_var_eda(train,['manufacturer_id'])
```



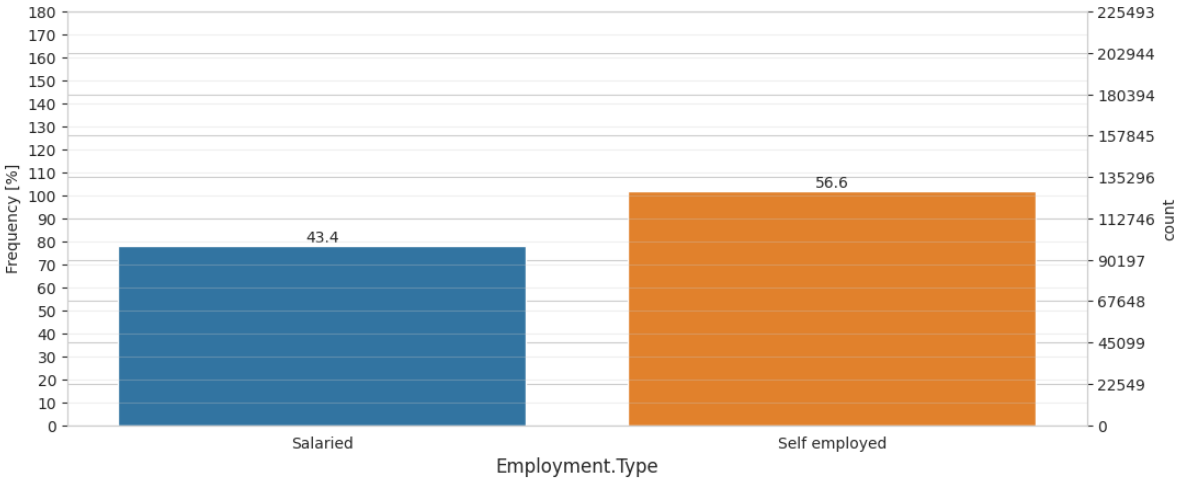
Observation :

- manufacturer_id - 120,49 have almost similar number of customers
- manufacturer_id - 86,45 have more than 70% of market share combined
- manufacturer_id - 86,45,51 have more than 90% of market share combined
- manufacturer_id - 152,153,156 have 0% of market share

Employment.Type

In [21]:

```
#Employment.Type
cat_var_eda(train,['Employment.Type'])
```



Observation:

More customers belong to self employed category based on this plot

There are some missing values as well but not significant in this case

In [22]:

```
#3% null value not much significant as discussed
(train['Employment.Type'].isnull().sum() / train.shape[0])*100
```

Out[22]:

```
3.2858110948128703
```

Identification Document Flags

Either id was given or not

In [23]:

```
#using melt as we have more then 1 id variables
diff_flags_uniques = pd.melt(frame=train, value_vars=['MobileNo_Avl_Flag','Aadhar_fl
                                                      'VoterID_flag','Driving_flag',
diff_flags_uniques.head(5)
```

Out[23]:

	variable	value
0	MobileNo_Avl_Flag	1
1	MobileNo_Avl_Flag	1
2	MobileNo_Avl_Flag	1
3	MobileNo_Avl_Flag	1
4	MobileNo_Avl_Flag	1

In [24]:

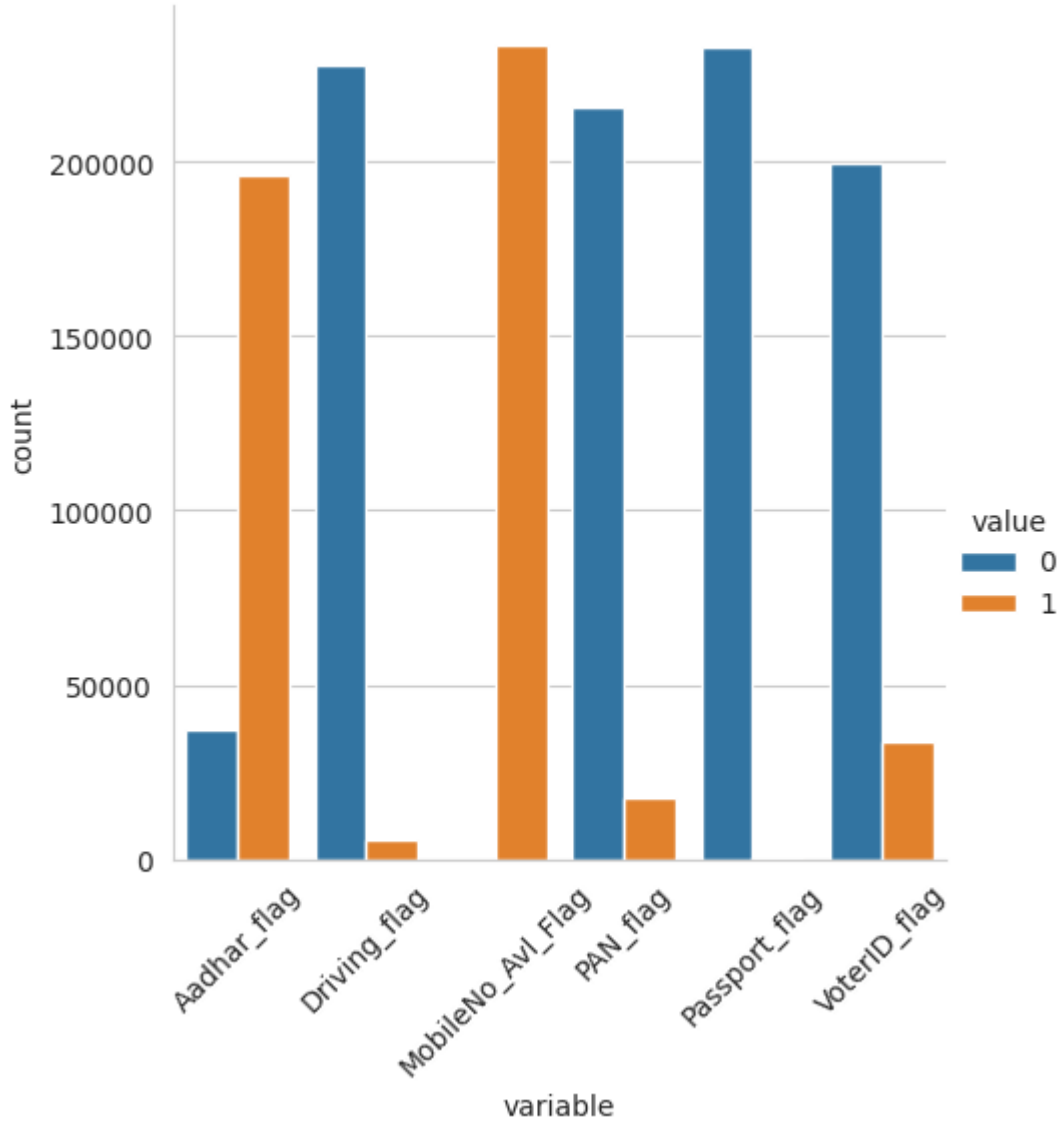
```
df_diff_flags_uniques = pd.DataFrame(diff_flags_uniques.groupby(['variable','value']
.rename(columns={'value':'count'}).reset_index()
df_diff_flags_uniques.head(5)
```

Out[24]:

	variable	value	count
0	Aadhar_flag	0	37230
1	Aadhar_flag	1	195924
2	Driving_flag	0	227735
3	Driving_flag	1	5419
4	MobileNo_Avl_Flag	1	233154

In [25]:

```
#visualizing
sns.catplot(x='variable', y='count', hue='value', data=df_diff_flags_uniques, kind='

plt.xticks(rotation=45)
plt.show()
```



Observation :

- Imbalanced distrubution
- Aadhar is most popular ID proof medium
  - Unlikely to be of much significance for target

Using Kernel Density Estimation -KDE plot

In [26]:

```
#this is a list of all features/variaibles which we have to usnderstand
print(numerical_features)
```

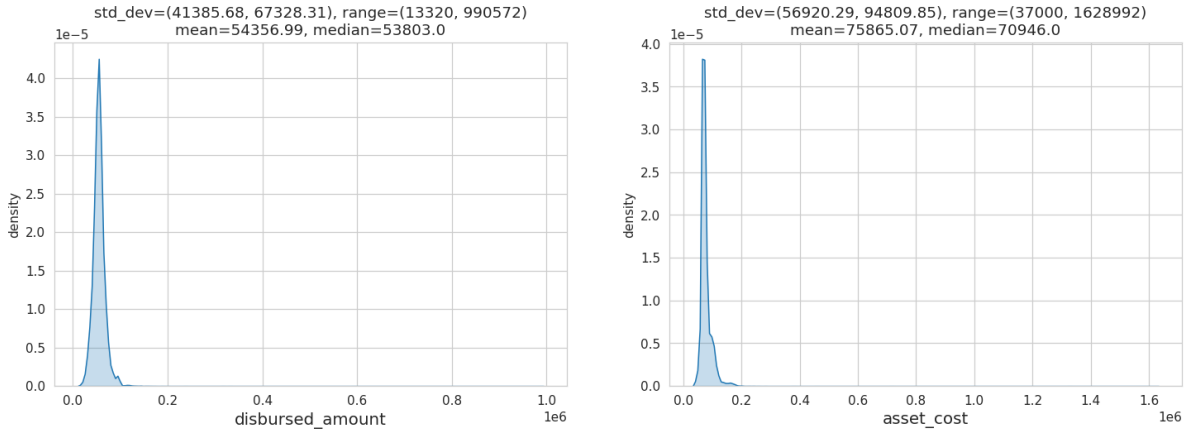
```
['disbursed_amount', 'asset_cost', 'ltv', 'Current_pincode_ID', 'PERFO
RM_CNS.SCORE', 'PRI.NO.OF.ACCTS', 'PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACC
TS', 'PRI.CURRENT.BALANCE', 'PRI.SANCTIONED.AMOUNT', 'PRI.DISBURSED.AM
OUNT', 'SEC.NO.OF.ACCTS', 'SEC.ACTIVE.ACCTS', 'SEC.OVERDUE.ACCTS', 'SE
C.CURRENT.BALANCE', 'SEC.SANCTIONED.AMOUNT', 'SEC.DISBURSED.AMOUNT',
'PRIMARY.INSTAL.AMT', 'SEC.INSTAL.AMT', 'NEW.ACCTS.IN.LAST.SIX.MONTH
S', 'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'NO.OF_INQUIRIES']
```

In [27]:

```
#fucntoin Univariate Numerical Variables Analysis
def num_var_eda(data,features,log=False,dpi=110):
    fig_size = len(features)
    plt.figure(figsize=(8*fig_size,5),dpi=dpi)

    for index,values in enumerate(features):
        #calculating descriptives of variable
        minimum = data[values].min()
        maximum = data[values].max()
        mean    = data[values].mean()
        median  = data[values].median()
        st_dev  = data[values].std()
       #calculating points of one standard deviation
        points = mean - st_dev,mean + st_dev

        #plotting variable with every information
        plt.subplot(1,fig_size,index + 1)
        ax = sns.kdeplot(data[values],shade=True) #KDE plot

        if log == True:
            ax.set_xscale('log')
        else:
            pass

        plt.xlabel(f'{values}',fontsize=13)
        plt.ylabel('density')
        plt.title(f'std_dev={(round(points[0],2),round(points[1],2))}, range={(round
```

In [28]:

```
#without log
num_var_eda(train,['disbursed_amount','asset_cost'])
```



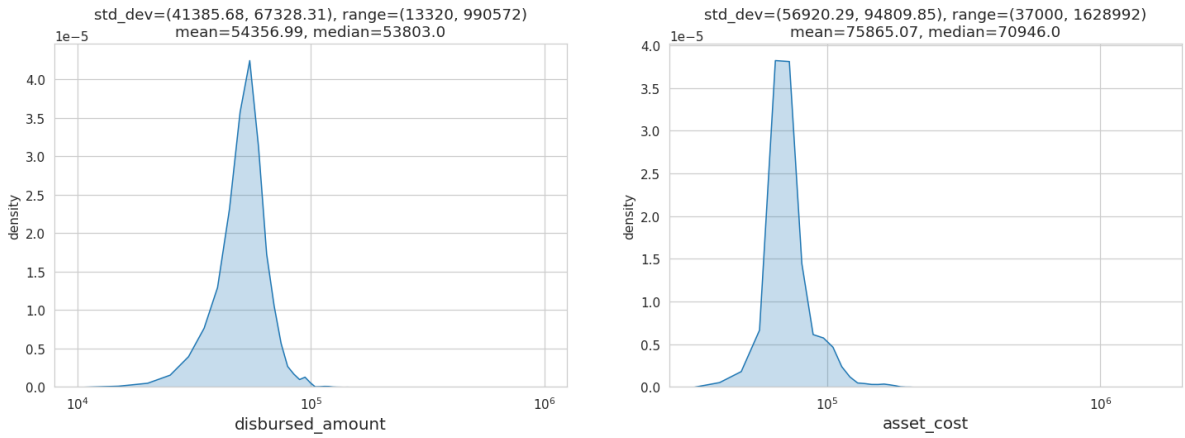
Observation

- asset_cost is more then disbursed_amount it is a valid output ==> most of customer do some down payment they never take loan on initeal amount

Both KDE plot are Right Skewed here are some outliers, will apply log scaling on x-axis

In [29]:

```
#with log
num_var_eda(train,['disbursed_amount','asset_cost'],log=True)
```



Observation :

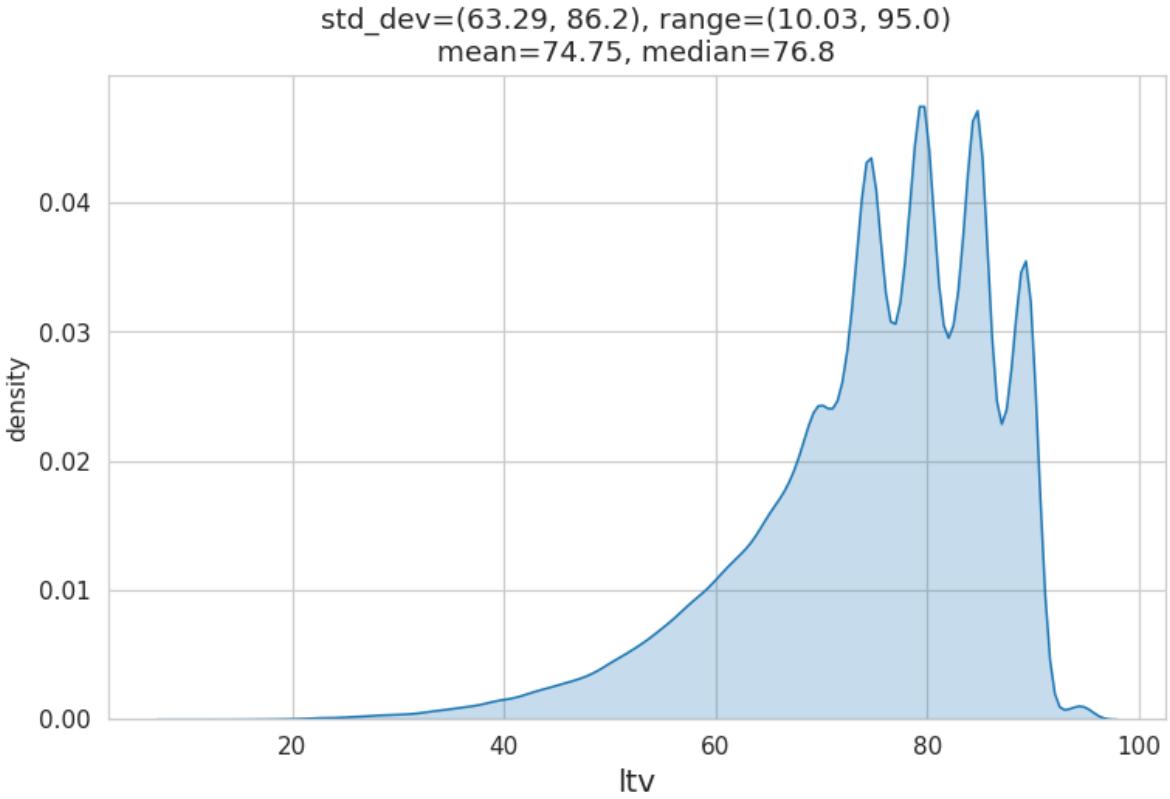
- Most of disbursed_amoun & asset_cost values lie between appx-10000 to 100000 which is expected from a two wheeler loan
- Invariably some down payment is done and that is why asset cost has higher values as compared to disbursed amount

Customer generaly dont take all money in loan, there is somthing downpament involved

NOTE :
Box-plot can be used here

In [30]:

```
num_var_eda(train,['ltv'],log=False)
```

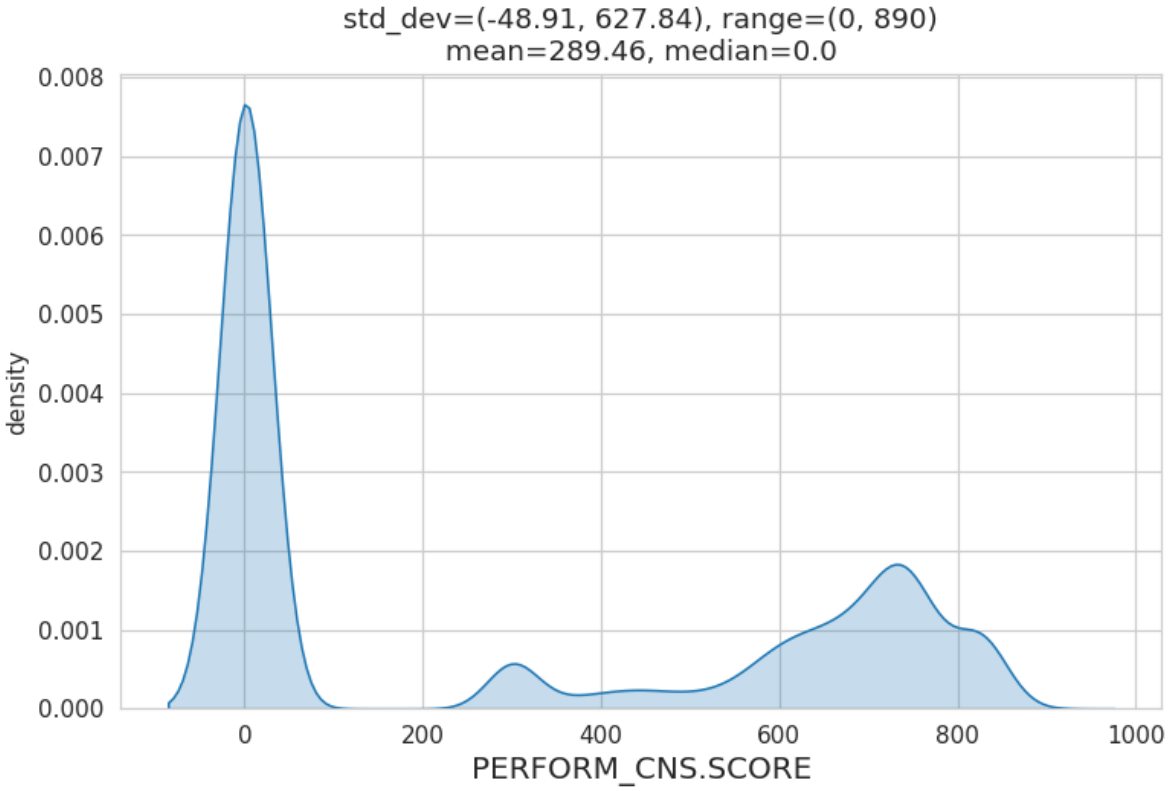


Observation

- KDE is Left Skewed[bell on right side] and non-smooth -- hand of monster

In [31]:

```
#column mens no beauro data for these peoples
num_var_eda(train,['PERFORM_CNS.SCORE'])
```



Observation :

- Most customers don't have any score, meaning they do not have any credit history.
- Related variable although non-numeric yet closely related is `cns score description`

In [32]:

```
train['PERFORM_CNS.SCORE'].value_counts()
```

Out[32]:

```
0      116950
300      8776
738      8662
825      7393
15       3765
        ...
863         1
834         1
838         1
837         1
867         1
Name: PERFORM_CNS.SCORE, Length: 573, dtype: int64
```

#this feature is categorial in natures but most related to 'PERFORM_CNS.SCORE' so I am exploring it now
train['PERFORM_CNS.SCORE.DESCRIPTION'].value_counts(normalize=True)

Vechile loans are generaly low ammount so this validates that most of customer do not have any previous credit history moany of user can be from villages

In [33]:

```
#this feature is categorial in natures but most related to 'PERFORM_CNS.SCORE' so I
train['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
```

Hunch was right as we can see clearly that most of users do not possess a bureau history

Will `group by this category and check average of score for each category and to interpret it better

In [34]:

```
#summary for each category
customer_score_summ = pd.DataFrame(train.groupby('PERFORM_CNS.SCORE.DESCRIPTION')['P

with pd.option_context('display.max_rows',None):
    display(customer_score_summ)
```

	PERFORM_CNS.SCORE.DESCRIPTION	PERFORM_CNS.SCORE
0	A-Very Low Risk	827.662631
1	B-Very Low Risk	774.183893
2	C-Very Low Risk	741.899221
3	D-Very Low Risk	715.958091
4	E-Low Risk	691.517437
5	F-Low Risk	666.064467
6	G-Low Risk	640.853561
7	H-Medium Risk	617.031947
8	I-Medium Risk	586.832644
9	J-High Risk	549.876734
10	K-High Risk	440.594177
11	L-Very High Risk	326.601411
12	M-Very High Risk	300.000000
16	Not Scored: No Updates available in last 36 mo...	18.000000
17	Not Scored: Not Enough Info available on the c...	17.000000
15	Not Scored: No Activity seen on the customer (...	16.000000
19	Not Scored: Sufficient History Not Available	15.000000
18	Not Scored: Only a Guarantor	14.000000
14	Not Scored: More than 50 active Accounts found	11.000000
13	No Bureau History Available	0.000000

Observation :

- Bureau score is higher for customers who are less likely to default and [above categories as you can see are created on basis of risk associated with each customer]

Exploring Primary Loan Features from Bureau Data

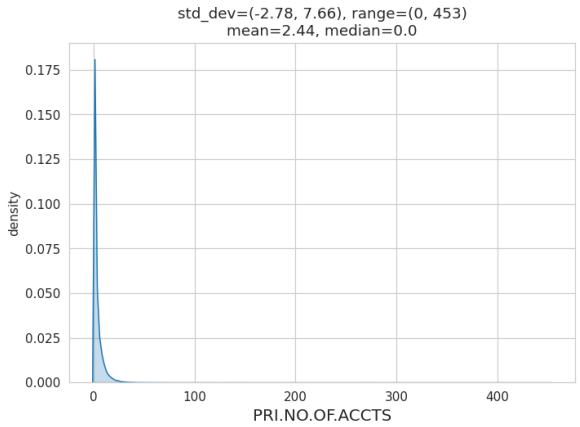
Loan taken by user itself features

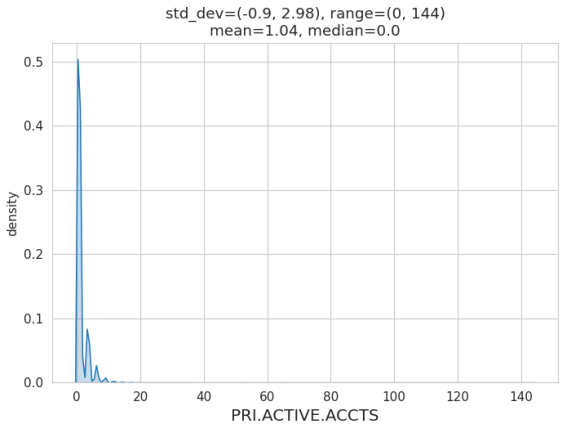
In [35]:

```
primary_loan_features = ['PRI.NO.OF.ACCTS','PRI.ACTIVE.ACCTS','PRI.OVERDUE.ACCTS','P
                         'PRI.DISBURSED.AMOUNT']
```

In [36]:

```
#total accounts & active accounts for each customer
num_var_eda(train,['PRI.NO.OF.ACCTS','PRI.ACTIVE.ACCTS'], log=False)
```





In [37]:

```
#not continious in nature
train['PRI.NO.OF.ACCTS'].head()
```

Out[37]:

```
0    0
1    1
2    0
3    3
4    0
Name: PRI.NO.OF.ACCTS, dtype: int64
```

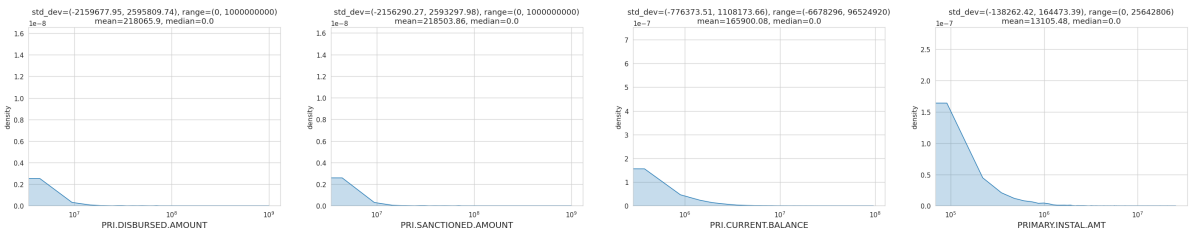
Observation

These variables are not continious that is why you see see above plotes

We can use `Histogram in this case`

In [38]:

```
#visualising Primary loan features
num_var_eda(train,['PRI.DISBURSED.AMOUNT','PRI.SANCTIONED.AMOUNT','PRI.CURRENT.BALAN
```



Observation

- All are Right Squed

Check-1

- I have history of 10 loans ==> `PRI.ACTIVE.ACCTS`
- Active loans in this time are say 4 ==> `PRI.NO.OF.ACCTS`

So `PRI.ACTIVE.ACCTS` cannot be greater then `PRI.NO.OF.ACCTS`

In [39]:

```
#Check 1
train[train['PRI.NO.OF.ACCTS'] < train['PRI.ACTIVE.ACCTS']]
```

Out[39]:

UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_

Nothing means there is no problem in our data

In [40]:

```
#sanity 2
train[train['PRI.NO.OF.ACCTS'] < train['PRI.OVERDUE.ACCTS']]
```

Out[40]:

UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_

In [41]:

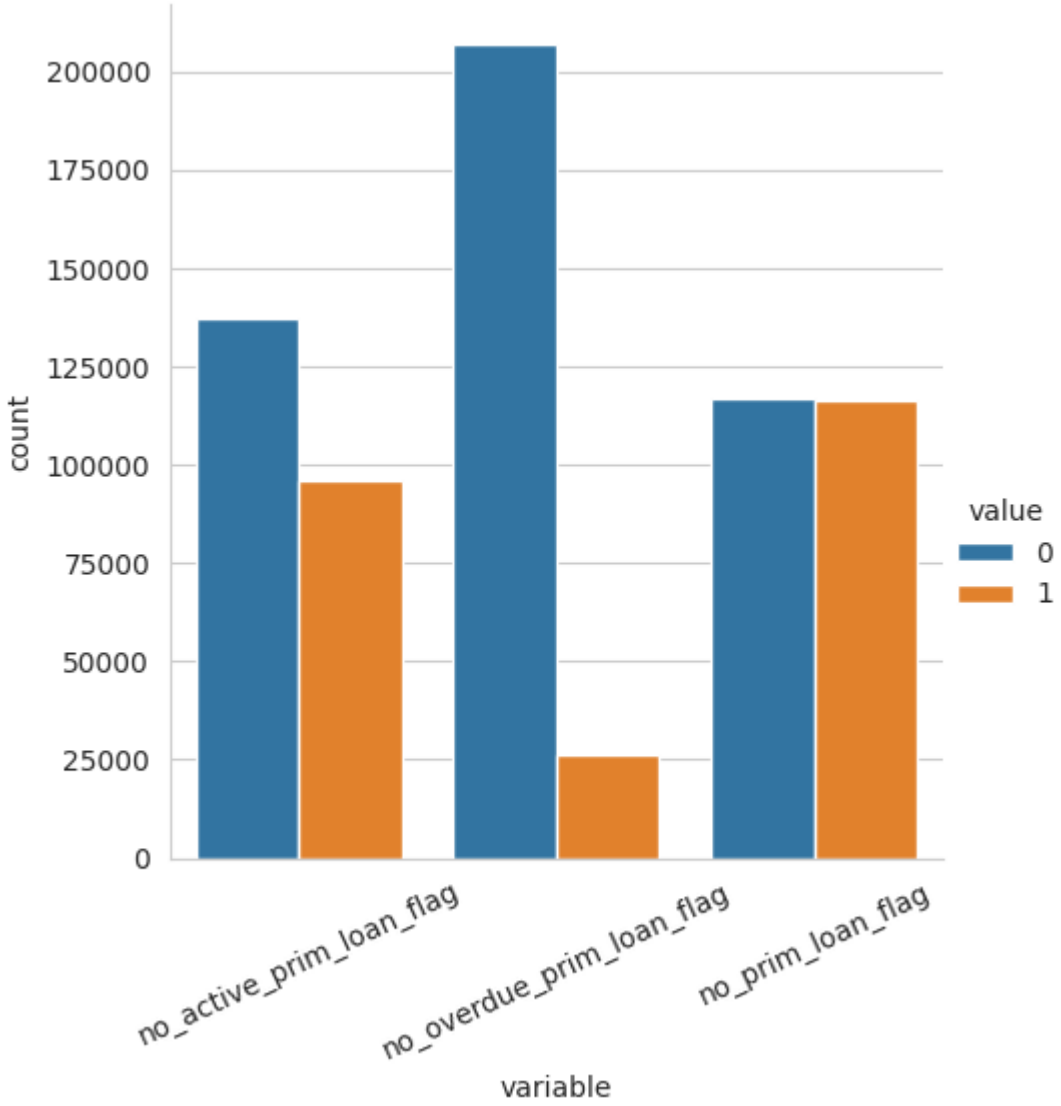
```
#checking how many of loan holders have primary account information
train['no_prim_loan_flag'] = train['PRI.NO.OF.ACCTS'].mask(train['PRI.NO.OF.ACCTS']

#checking how many customers have active & Overdue accounts
train['no_active_prim_loan_flag'] = train['PRI.ACTIVE.ACCTS'].mask(train['PRI.ACTIVE
train['no_overdue_prim_loan_flag'] = train['PRI.OVERDUE.ACCTS'].mask(train['PRI.OVER
```

In [42]:

```
#collectively visualise above 3 variables
df_uniques = pd.melt(frame=train, value_vars=['no_prim_loan_flag','no_active_prim_lo
df_uniques = pd.DataFrame(df_uniques.groupby(['variable', 'value'])['value'].count()
.rename(columns={'value': 'count'}).reset_index()
```

In [43]:

```
#catplot
sns.catplot(x='variable', y='count', hue='value', data=df_uniques, kind='bar')
plt.xticks(rotation=25)
plt.show()
```



Observation

- 50% user have no primary account which validates 'PERFORM_CNS.SCORE' outcome which can be seen in down shell

In [44]:

```
train['PERFORM_CNS.SCORE'].value_counts()
```

Out[44]:

```
0      116950
300      8776
738      8662
825      7393
15       3765
        ...
863         1
834         1
838         1
837         1
867         1
Name: PERFORM_CNS.SCORE, Length: 573, dtype: int64
```

no_prim_loan_flag ==> people who have primary account == 1

In [45]:

```
#user who dont have no_primary_loan_accounts what is there score description
train[train['no_prim_loan_flag']==0]['PERFORM_CNS.SCORE.DESCRIPTION'].value_counts(r
```

Out[45]:

```
No Bureau History Available    1.0
Name: PERFORM_CNS.SCORE.DESCRIPTION, dtype: float64
```

Concluding

From primary loan information and basic_beauro infromation can say 50% of customers dont have any credit history, we have validated it up till now

Exploring Secondary Loan information

In [46]:

```
sec_loan_features = ['SEC.NO.OF.ACCTS','SEC.ACTIVE.ACCTS','SEC.OVERDUE.ACCTS','SEC.C
                     'SEC.SANCTIONED.AMOUNT','SEC.DISBURSED.AMOUNT','SEC.INSTAL.AMT'
```

In [47]:

```
train['SEC.DISBURSED.AMOUNT'].value_counts(normalize=True)
```

Out[47]:

```
0         0.984114
50000     0.000253
100000    0.000202
200000    0.000154
40000     0.000133
            ...
49233     0.000004
557000    0.000004
163204    0.000004
295998    0.000004
49246     0.000004
Name: SEC.DISBURSED.AMOUNT, Length: 2553, dtype: float64
```

Observation :

- Secondary loan doesn't have much information as it is 0 for 98% customers
  Only 1% of customers have Secondary data so for now it is not usefull to analyse this data

In [48]:

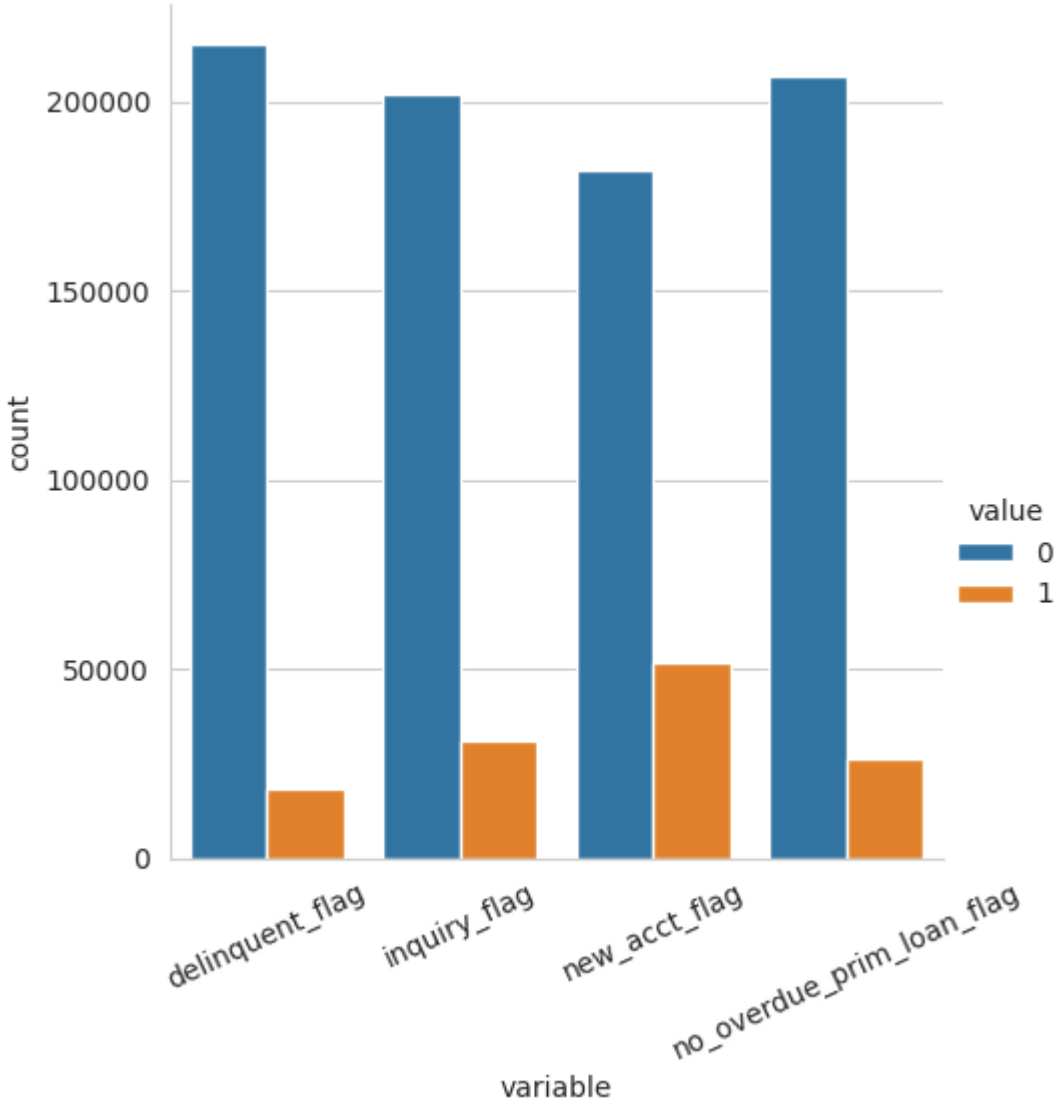
```
#checking other flags from bureau data
train['new_acct_flag']   = train['NEW.ACCTS.IN.LAST.SIX.MONTHS'].mask(train['NEW.ACC
train['delinquent_flag'] = train['DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS'].mask(train['
train['inquiry_flag']    = train['NO.OF_INQUIRIES'].mask(train['NO.OF_INQUIRIES'] >
```

In [49]:

```
#plotting same information
df_uniques = pd.melt(frame=train,value_vars=['new_acct_flag','delinquent_flag','inqu
df_uniques = pd.DataFrame(df_uniques.groupby(['variable','value'])['value'].count())
.rename(columns={'value': 'count'}).reset_index()
```

In [50]:

```
#catplot
sns.catplot(x='variable', y='count', hue='value', data=df_uniques, kind='bar')
plt.xticks(rotation=25)
plt.show()
```



Observation
From above analysis we can say 50% user dont have previous account information and then for rest of 50% hae this information which can be inderstood from above plot

Exploring id_features

Let us explore ID features now and see if we have some interesting information

In [51]:

```
#list of all id features
id_features
```

Out[51]:

```
['UniqueID', 'branch_id', 'supplier_id', 'Current_pincode', 'Employee_
code_ID']
```

In [52]:

```
#uniqueid ==> unique for each customer must be same as numbe of rows in data
train['UniqueID'].nunique(), train.shape[0]
```

Out[52]:

```
(233154, 233154)
```

Observation ==> no repeated customer found

In [53]:

```
# employ
train.Employee_code_ID.nunique()
```

Out[53]:

```
3270
```

Observation
Employ who have dispursed loan in any manny are 3270

In [54]:

```
#[[ Do all employees work in a single branch????? ]]
train[['branch_id','Employee_code_ID']].drop_duplicates().groupby('Employee_code_ID'
reset_index()['branch_id'].value_counts()
```

Out[54]:

```
1    3109
2     161
Name: branch_id, dtype: int64
```

Observatin 161-Employs are mapped to multiple branches oops, Problem found

- maybe some employ are working from 2 different branches
- maybe employs are mapped to multiple branches

- maybe 2 branches are having a single person who is responsible for despursing loan
- it can totally be anomaly

`pincode`

`if pincode have any sequence??`

In [55]:

```
train.Current_pincode_ID.head()
```

Out[55]:

```
0    1441
1    1502
2    1497
3    1501
4    1495
Name: Current_pincode_ID, dtype: int64
```

There are randome number not pincode, our task is to find out if these number make any sense
Sharing pincode can reveal location of customer there migh be any clause underwhich bank cannot share exact pin code, but still there will be some paterns as 1441 will be near to 1442 and kinde of thing, lets find out

In [56]:

```
#range of number for every branch_id
train.groupby('branch_id').agg(min_pin=('Current_pincode_ID',min),max_pin=('Current_
```

Out[56]:

branch_id	min_pin	max_pin
1	4906	6146
2	1593	2398
3	1	604
5	3299	3588
7	5725	5858
...	...	...
257	936	970
258	125	178
259	216	258
260	4153	4288
261	179	214

82 rows × 2 columns

`Observation`
On basis of above table `Current_pincode_ID` has a sequence, they are not randomly assigned numbers
`Current_pincode_ID` is important feature for model building

To make this finding more impact full lets do same with 'supplier_id'

In [57]:

```
#range of number for every supplier_id
train.groupby('supplier_id').agg(min_pin=('Current_pincode_ID',min),max_pin=('Currer
```

Out[57]:

supplier_id	min_pin	max_pin
10524	5368	5425
12311	5368	5426
12312	5064	5221
12374	5108	5219
12441	5368	5476
...	...	...
24794	6982	6982
24797	727	776
24799	104	104
24802	1467	1471
24803	5471	5473

2953 rows × 2 columns

`Observation`

- Suppliers are generaly takes reasons so above table patterns validated out findings that `Current_pincode_ID` has a sequence

In [58]:

```
#range of number for every Employee_code_ID
train.groupby('Employee_code_ID').agg(min_pin=('Current_pincode_ID',min),max_pin=('C
```

Out[58]:

Employee_code_ID	min_pin	max_pin
1	3	100
3	5939	6010
4	2679	2769
5	5060	5114
7	6158	6379
...	...	...
3791	820	828
3792	3003	3003
3793	4638	4638
3794	248	248
3795	3675	3675

3270 rows × 2 columns

Observation

See Eemploy with id 1 works with 3-100 pin reason customers only

Concluding

All employees and suppliers are generally operating in a single location but only ids for pin have meaning

Checking frequency of each branch ID, Supplier ID & Employee ID

In [59]:

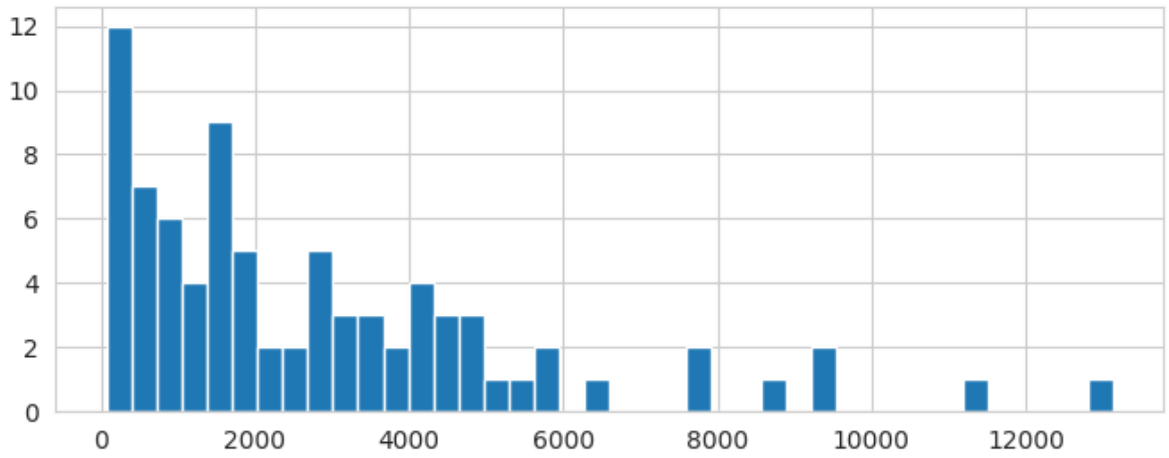
```
train['branch_id'].value_counts()
```

Out[59]:

```
2      13138
67     11328
3       9230
5       9218
36      8832
       ...
217      183
261      176
84       156
111       89
158       69
Name: branch_id, Length: 82, dtype: int64
```

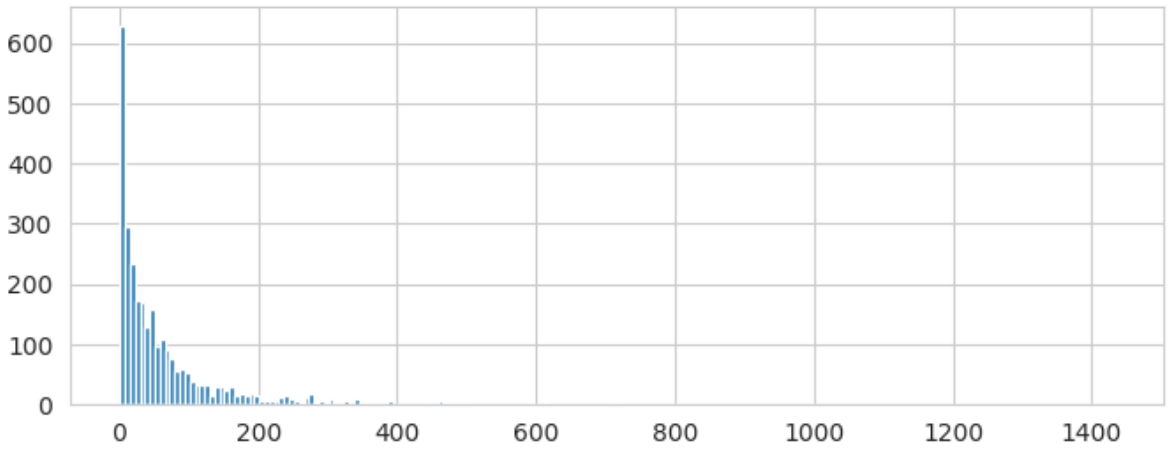
In [60]:

```
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['branch_id'].value_counts(),bins=40)

plt.show()
```



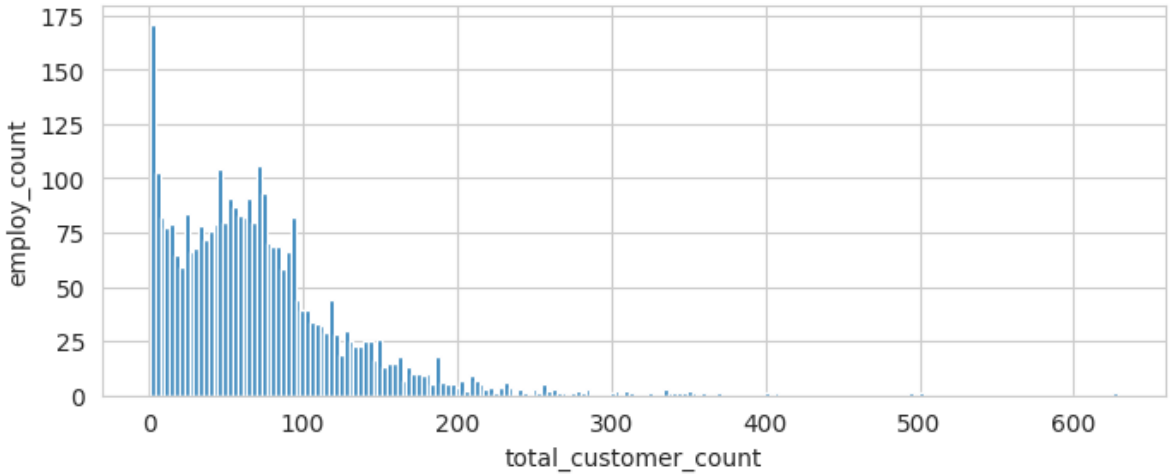
In [61]:

```
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['supplier_id'].value_counts().values,bins=200,log=False)
plt.show()
```



In [62]:

```
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['Employee_code_ID'].value_counts().values,bins=200,log=False)


plt.xlabel('total_customer_count')
plt.ylabel('employ_count')
plt.show()
```



Most of employ are working with single loan

Exploring Date Time Features

In [63]:

```
#all datetime features list
date_time_features
```

Out[63]:

```
['Date.of.Birth', 'DisbursalDate', 'CREDIT.HISTORY.LENGTH', 'AVERAGE.A
CCT.AGE']
```

```
 DisbursalDate
```

In [64]:

```
train['DisbursalDate'] = pd.to_datetime(train['DisbursalDate'], format='%d-%m-%y') #
test['DisbursalDate']  = pd.to_datetime(test['DisbursalDate'], format='%d-%m-%y')

#Date.of.Birth feature
train['Date.of.Birth'] = pd.to_datetime(train['Date.of.Birth'], format='%d-%m-%y')
test['Date.of.Birth']  = pd.to_datetime(test['Date.of.Birth'], format='%d-%m-%y')
```

In [65]:

```
#train
train['DisbursalDate'].describe()
```

Out[65]:

```
count                  233154
unique                     84
top       2018-10-31 00:00:00
freq                     8826
first     2018-08-01 00:00:00
last      2018-10-31 00:00:00
Name: DisbursalDate, dtype: object
```

In [66]:

```
#test
test['DisbursalDate'].describe()
```

Out[66]:

```
count                  112392
unique                     27
top       2018-11-15 00:00:00
freq                     7803
first     2018-11-03 00:00:00
last      2018-11-30 00:00:00
Name: DisbursalDate, dtype: object
```

Observation

- All `DisbursalDates` in test dataset lie in month of November 2018
- All `DisbursalDates` in test dataset lie in future of all `DisbursalDates` in train dataset

In [67]:

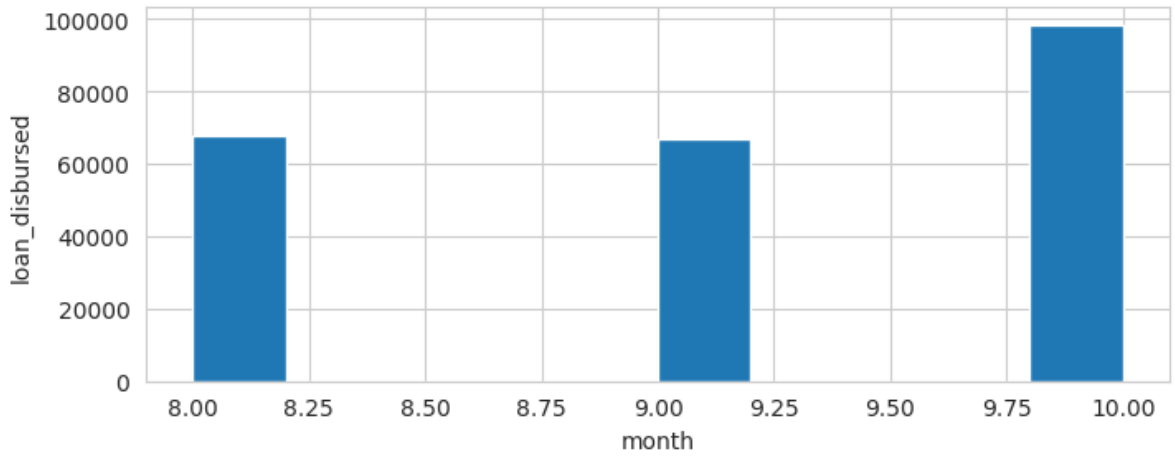
```
# finding age in years
train['age_in_days'] = (train.DisbursalDate - train['Date.of.Birth']).dt.days
train['age_in_years'] = train['age_in_days']/365.25 #every 4 year have 1 more extra
```

In [68]:

```
#extracting more features out of disbursal date
train['disbursal_day']       = train.DisbursalDate.dt.day
train['disbursal_month']     = train.DisbursalDate.dt.month
train['disbursed_dayofweek'] = train.DisbursalDate.dt.dayofweek
```

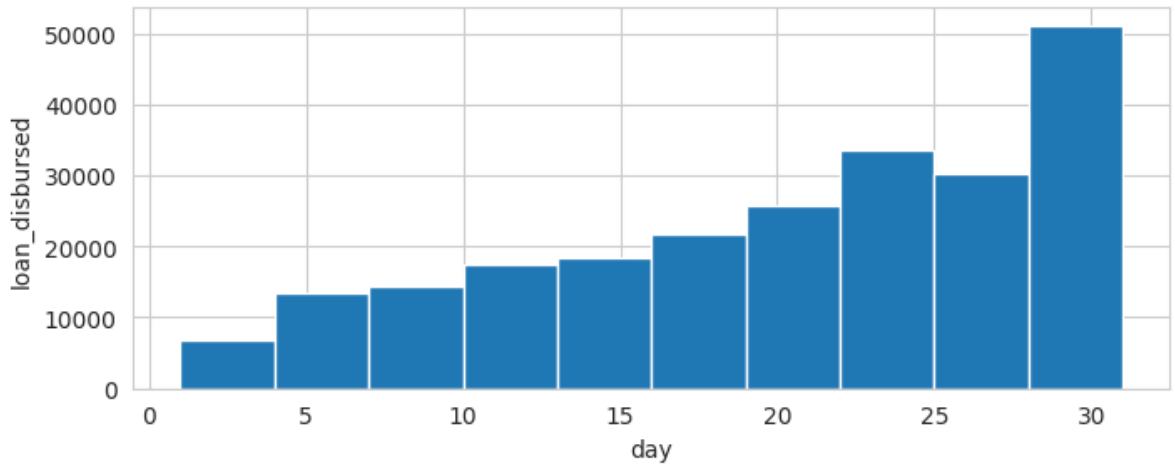
In [69]:

```
#Disbursal Month Distribution
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['disbursal_month'])

plt.xlabel('month')
plt.ylabel('loan_disbursed')
plt.show()
```



In [70]:

```
#disbursal Day Distribution
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['disbursal_day'])

plt.xlabel('day')
plt.ylabel('loan_disbursed')
plt.show()
```

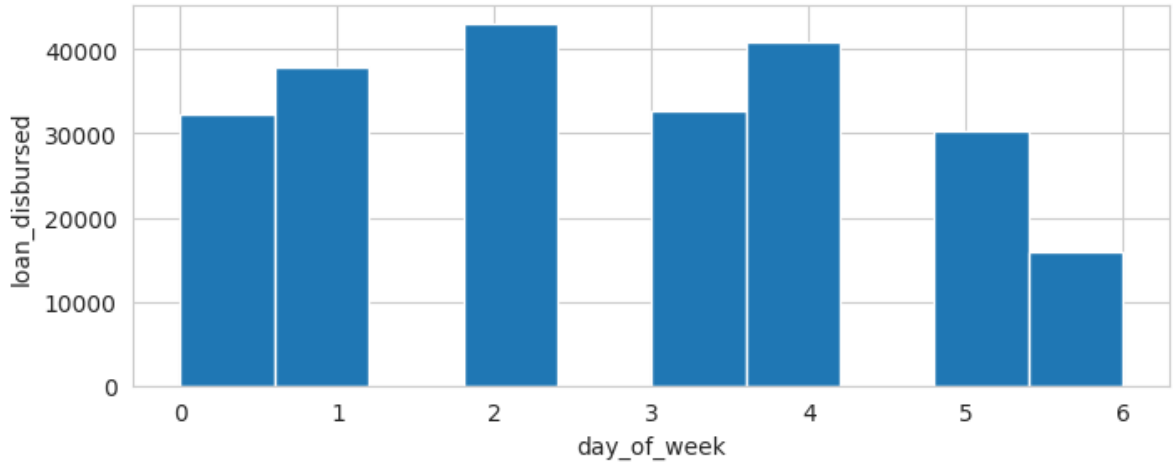


Observation

Loan generally get disbursed on last day of month, general practice, graph validates that so no further invest

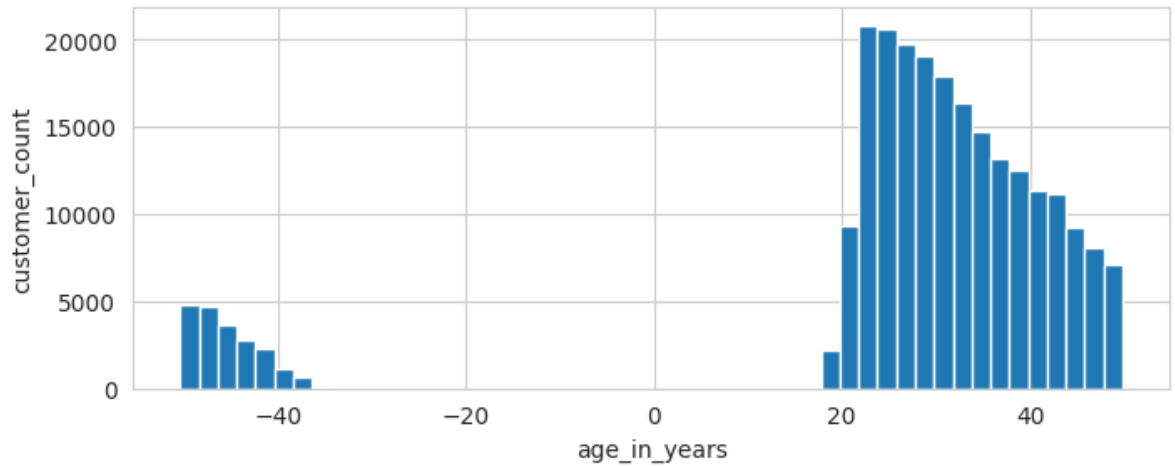
In [71]:

```
#Disbursal Day of Week Distribution
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['disbursed_dayofweek'])

plt.xlabel('day_of_week')
plt.ylabel('loan_disbursed')
plt.show()
```



In [72]:

```
#distribution of age
plt.figure(figsize=(8,3),dpi=100)
plt.hist(np.round(train['age_in_years'],5),bins=50)

plt.xlabel('age_in_years')
plt.ylabel('customer_count')
plt.show()
```



Observation :

Customer less then 18 cant get a loan on themself, so plot validates it we are going good and +ve

User in age bracket of 22 - 30 takes most of two wheler loan

Date.of.Birth'

In [73]:

```
train['Date.of.Birth'].describe()
```

Out[73]:

```
count                  233154
unique                  15433
top       1988-01-01 00:00:00
freq                     2173
first     1969-01-01 00:00:00
last      2068-12-31 00:00:00
Name: Date.of.Birth, dtype: object
```

In [74]:

```
from datetime import timedelta, date

print(date(year=2019,month=1,day=1))
print(timedelta(days=365.25*100))
```

```
2019-01-01
36525 days, 0:00:00
```

In [75]:

```
#100 years
36525/365.25
```

Out[75]:

```
100.0
```

In [76]:

```
# if date of birth is more then 2019
filter_date = train['Date.of.Birth'].dt.date > date(year=2019,month=1,day=1)
train.loc[filter_date,'Date.of.Birth'] -= timedelta(days=365.25*100) #100 years
```

In [77]:

```
train['Date.of.Birth'].describe()
```

Out[77]:

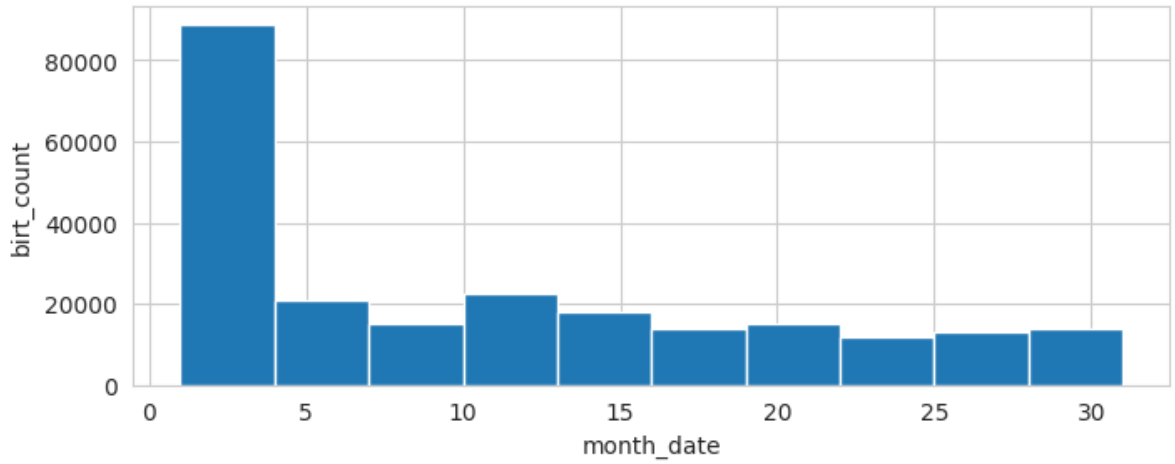
```
count                  233154
unique                  15433
top       1988-01-01 00:00:00
freq                     2173
first     1949-09-15 00:00:00
last      2000-10-20 00:00:00
Name: Date.of.Birth, dtype: object
```

In [78]:

```
#birth Day and month exploration
train['birth_day'] = train['Date.of.Birth'].dt.day
train['birth_month'] = train['Date.of.Birth'].dt.month
```

In [79]:

```
#distribution of birth day
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['birth_day'])

plt.xlabel('month_date')
plt.ylabel('birt_count')
plt.show()
```

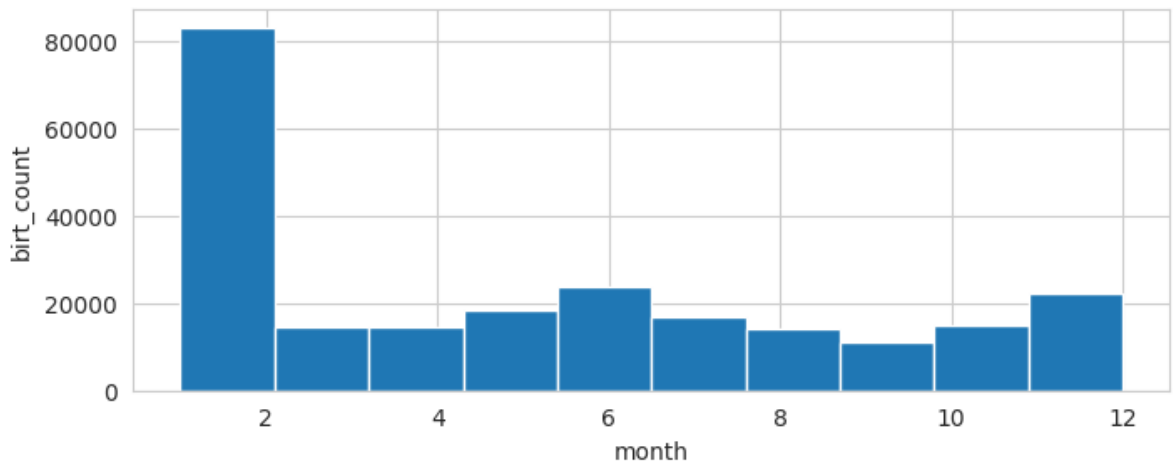


Observation

How it possible that most of user have birt date of 1-3, lets get deep into it

In [80]:

```
#distribution of birth day
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['birth_month'])

plt.xlabel('month')
plt.ylabel('birt_count')
plt.show()
```



Observation

Most of user are born on 1-jan we can conclude, there is some problem

Lets see how it effects target variable, **Are these customer defaulting more**

For this I will `Digress` to multivariated analysis

In [81]:

```
#finding loan default %tg for customers whose birth_month and birth_day is 1
train[(train.birth_day == 1) & (train.birth_month == 1)]['loan_default'].mean()
```

Out[81]:

```
0.2572111467217122
```

`Observation`

There are 25% of such user whose birth_month and birth_day is 1 and they are defaulter

In [82]:

```
#overall default rate
train['loan_default'].mean()
```

Out[82]:

```
0.2170711203753742
```

`Observation`

Overall default rate is less i.e. 22% appx

`Concluding`

User whose birth_month and birth_day is 1 they will be defaulter 25% of time

`CREDIT.HISTORY.LENGTH` and `AVERAGE.ACCT.AGE`

Converting these features in number now

In [83]:

```
train['CREDIT.HISTORY.LENGTH']
```

Out[83]:

```
0           0yrs 0mon
1          1yrs 11mon
2           0yrs 0mon
3           1yrs 3mon
4           0yrs 0mon
             ...
233149      3yrs 3mon
233150      0yrs 6mon
233151      0yrs 0mon
233152      0yrs 0mon
233153      0yrs 0mon
Name: CREDIT.HISTORY.LENGTH, Length: 233154, dtype: object
```

Exploring this above data is hard so converting it into number

In [84]:

```
import re

#using regex to extract number from text
def map_credit_history_to_months(row_data):
    re_data = list(map(int, re.findall(r'\d+',row_data)))
    return re_data[0]*12 + re_data[1] #extract first number mult by 12 add months wi
```

In [85]:

```
#applying above function
train['credit_history_in_months'] = train['CREDIT.HISTORY.LENGTH'].apply(map_credit_
train['avg_acct_age_in_months'] = train['AVERAGE.ACCT.AGE'].apply(map_credit_history
```

In [86]:

```
#credit_history_in_months
train['credit_history_in_months']
```

Out[86]:

```
0          0
1         23
2          0
3         15
4          0
          ..
233149    39
233150     6
233151     0
233152     0
233153     0
Name: credit_history_in_months, Length: 233154, dtype: int64
```

In [87]:

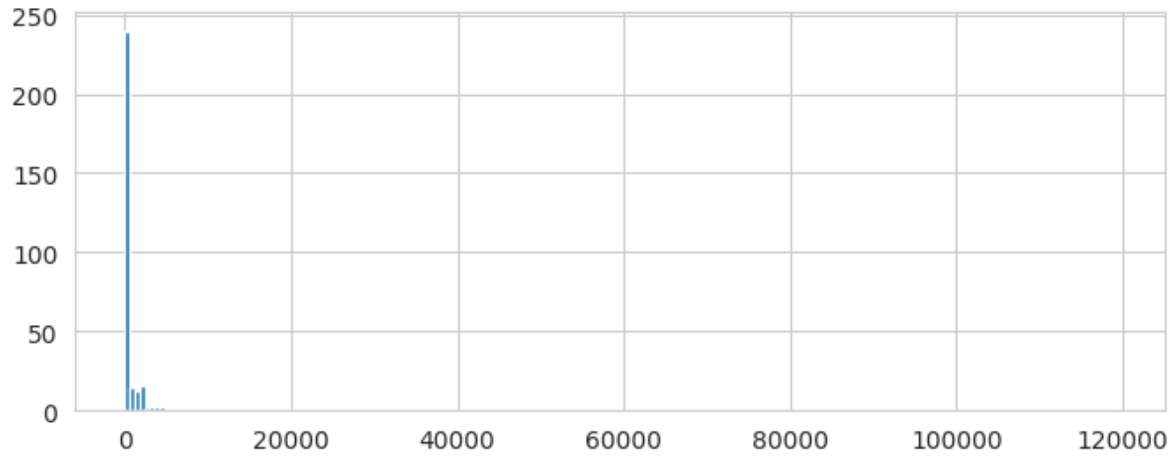
```
train.credit_history_in_months.describe()
```

Out[87]:

```
count    233154.000000
mean         16.252404
std          28.581255
min           0.000000
25%           0.000000
50%           0.000000
75%          24.000000
max         468.000000
Name: credit_history_in_months, dtype: float64
```

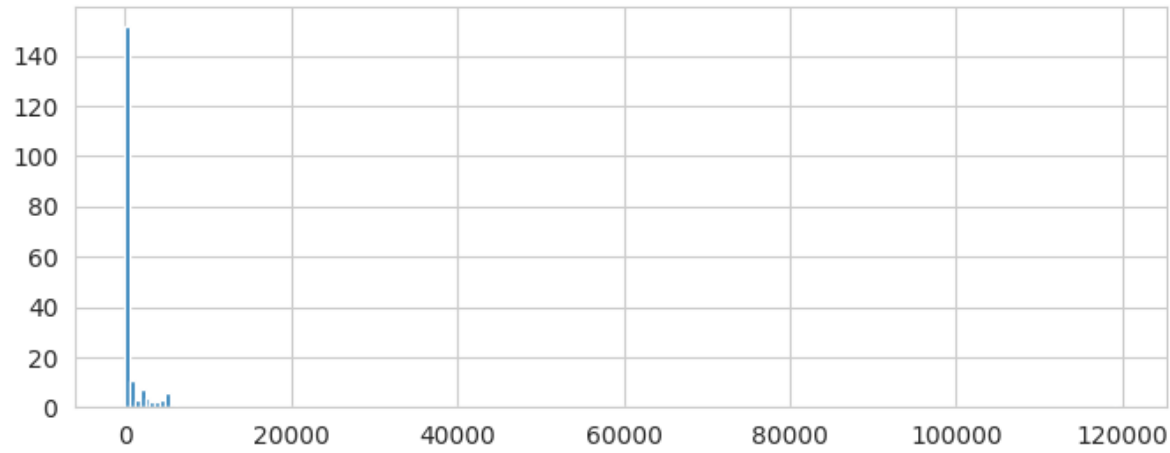
In [88]:

```
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['credit_history_in_months'].value_counts().values,bins=200)

plt.show()
```



In [89]:

```
plt.figure(figsize=(8,3),dpi=100)
plt.hist(train['avg_acct_age_in_months'].value_counts().values,bins=200)

plt.show()
```



In [90]:

```
from scipy.stats import chi2_contingency
```

Applying chi^2 test to see if there is significan relationship between `target` and other `categorical` variable

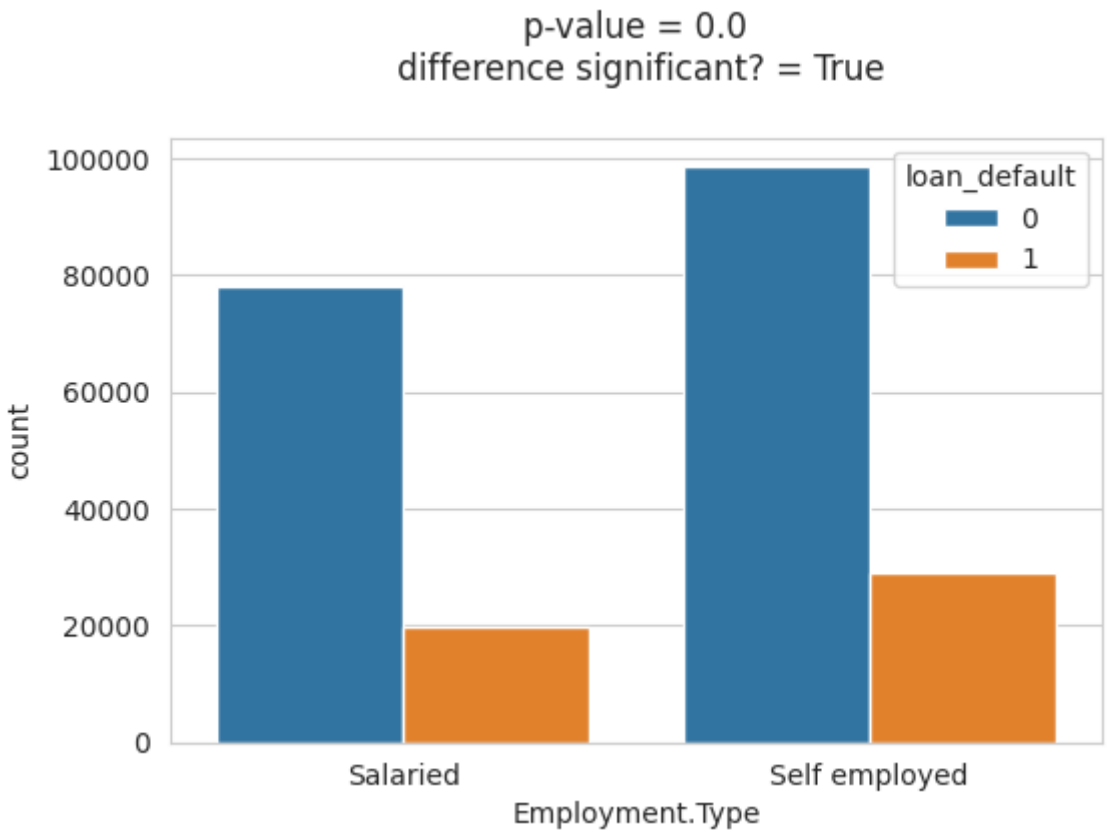
- we are creating a stacked bar plot too, simply to compare

In [91]:

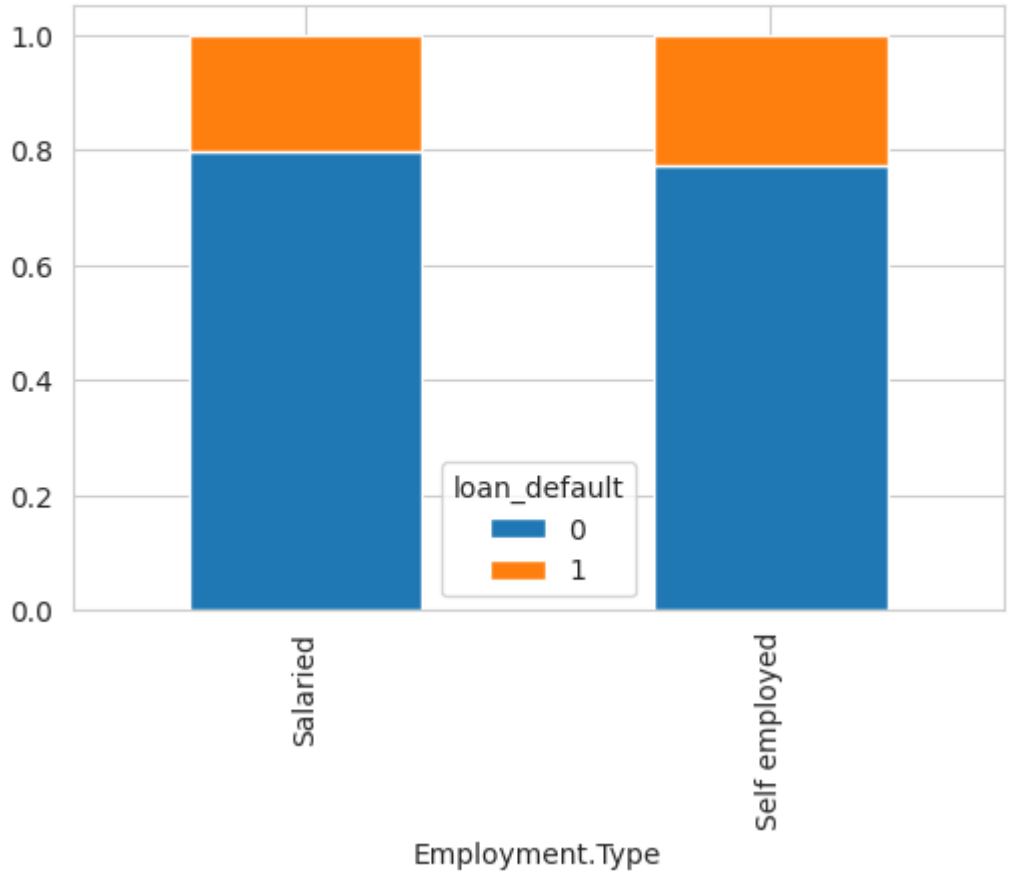
```
def biv_cat_cat_analysis(data,target_,cat,sort=False):
    data_copy = data.copy()

    if sort == True:
        df = data[[cat,target_]].groupby([cat][target_].count().reset_index(name='cc
        data_copy = data_copy.merge(df, on=cat, how='left')
        data = data_copy.sort_values(by='count', ascending=False)
        data.drop(cat,axis=1, inplace=True)
        data[cat] = data['count'].rank(ascending=False)
    else:
        pass
    data = data[[cat,target_]][:]

    #crosstab
    table = pd.crosstab(data[target_], data[cat],)
    f_obs = np.array([table.iloc[0][:].values, table.iloc[1][:].values])

    #### Performing chi2-test ####
    chi, p, dof, expected = chi2_contingency(f_obs)
    if p < 0.05:
        sig = True
    else:
        sig = False

    sns.countplot(x=cat, hue=target_, data=data)
    plt.title(f"p-value = {round(p,8)}\n difference significant? = {sig}\n")

    df = data.groupby(cat)[target_].value_counts(normalize=True).unstack()
    df.plot(kind='bar', stacked='True',title=str(df))
    int_level = data[cat].value_counts()
```

In [92]:

```
biv_cat_cat_analysis(train,target_='loan_default',cat='Employment.Type')
```



loan_default 0 1
Employment.Type
Salaried 0.796542 0.203458
Self employed 0.772343 0.227657

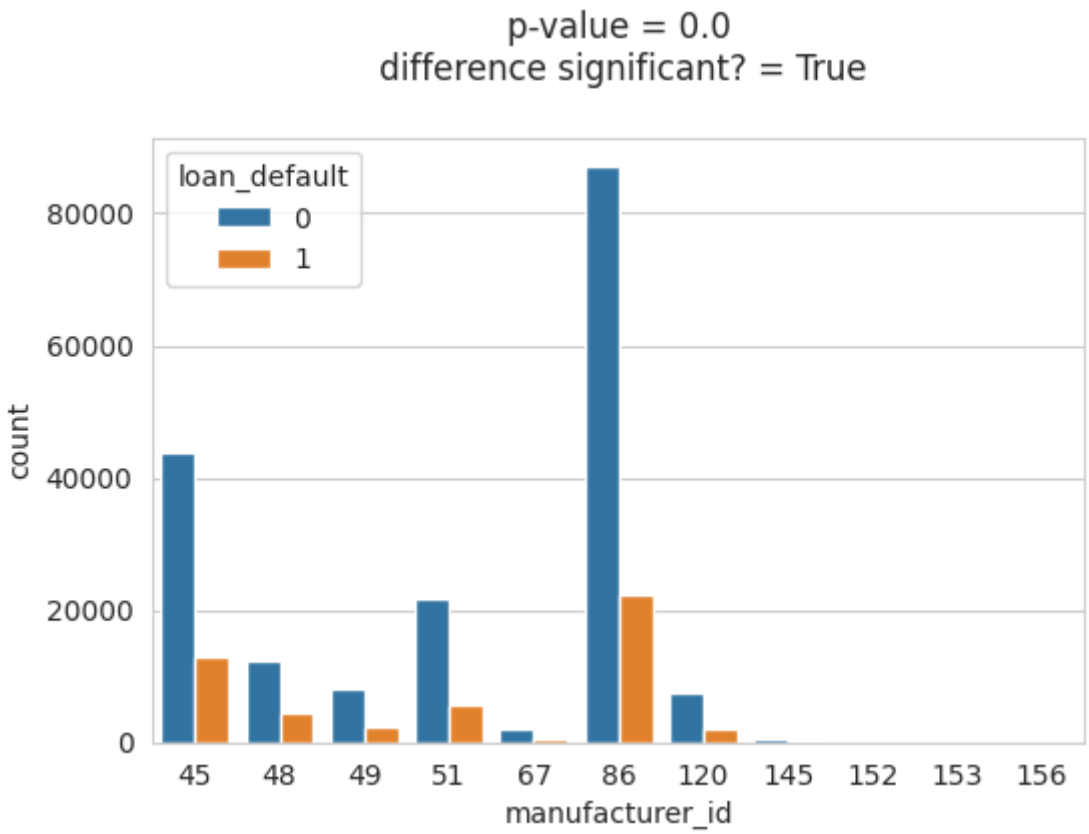


`Observation`

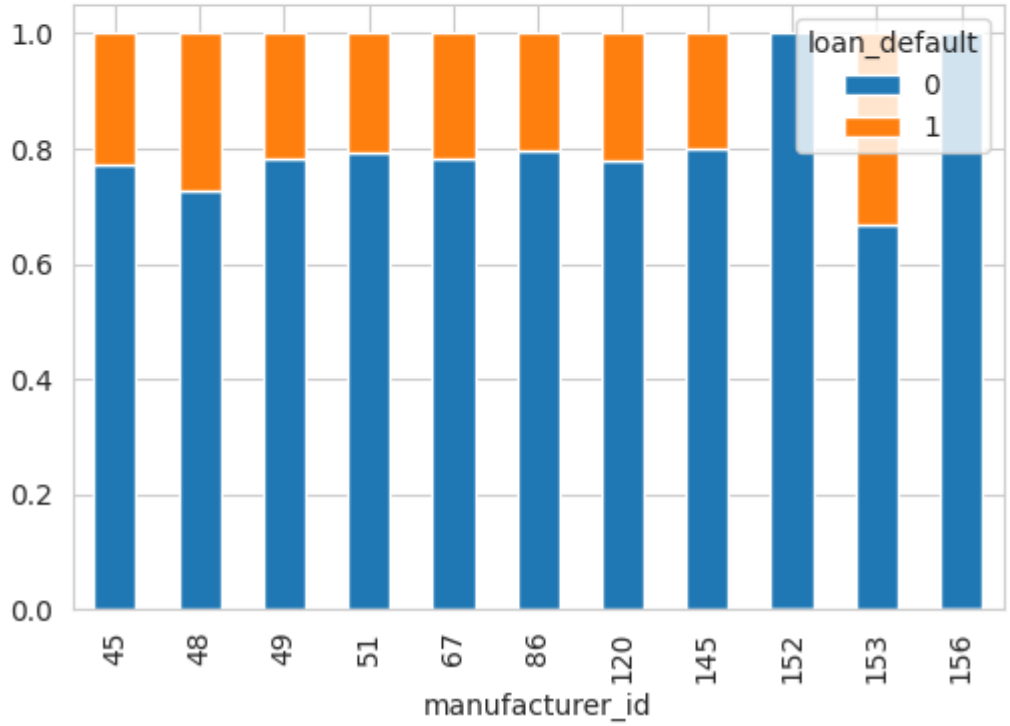
Out Hypothesis was that self employed people defaults more and that was true as 23%appx time self employed people dont pay there loans or defaults which is 3% more then salaried people

In [93]:

```
biv_cat_cat_analysis(train,target_='loan_default',cat='manufacturer_id')
```



loan_default	0	1
manufacturer_id		
45	0.771501	0.228499
48	0.727469	0.272531
49	0.781213	0.218787
51	0.792053	0.207947
67	0.782536	0.217464
86	0.795406	0.204594
120	0.779250	0.220750
145	0.799486	0.200514
152	1.000000	NaN
153	0.666667	0.333333
156	1.000000	NaN



`Observation`

Major Manufacturer are with id-86&45

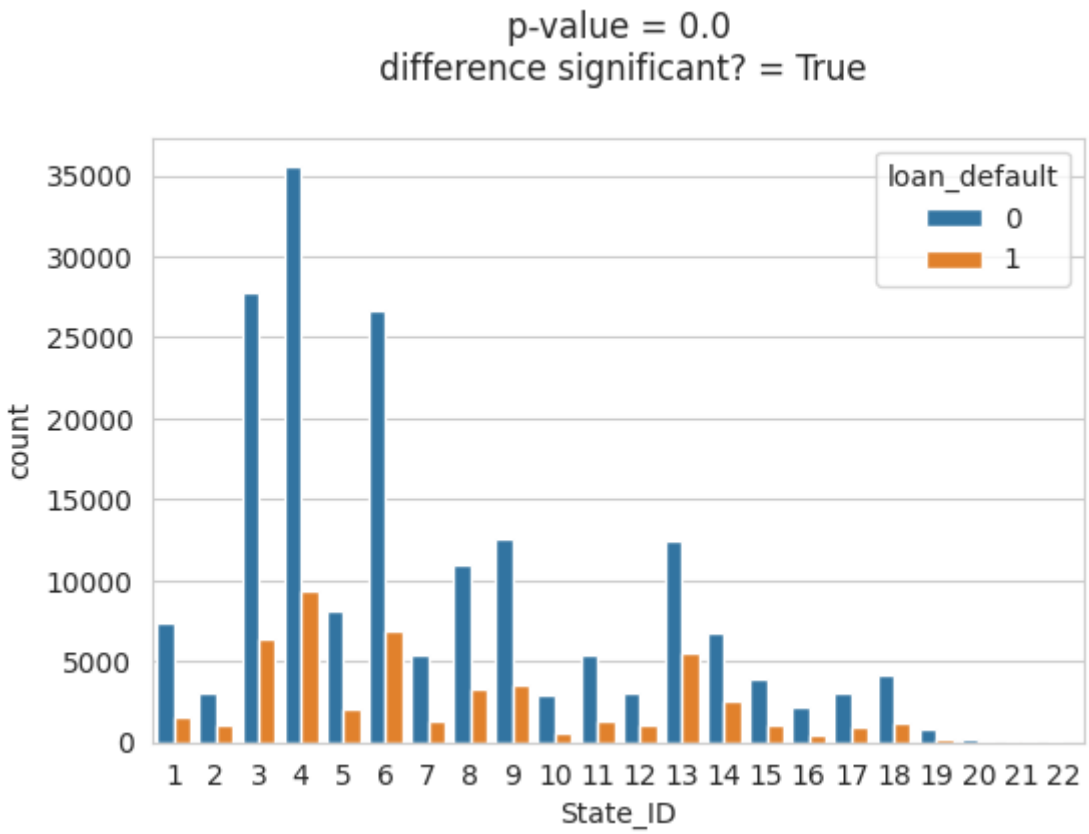
- id-45 have 23%appx default rate
- id-86 have 20% dafault rate -- major of major manufacturer have least default rate

`NaN values` can be seen as they are very small manufacturer

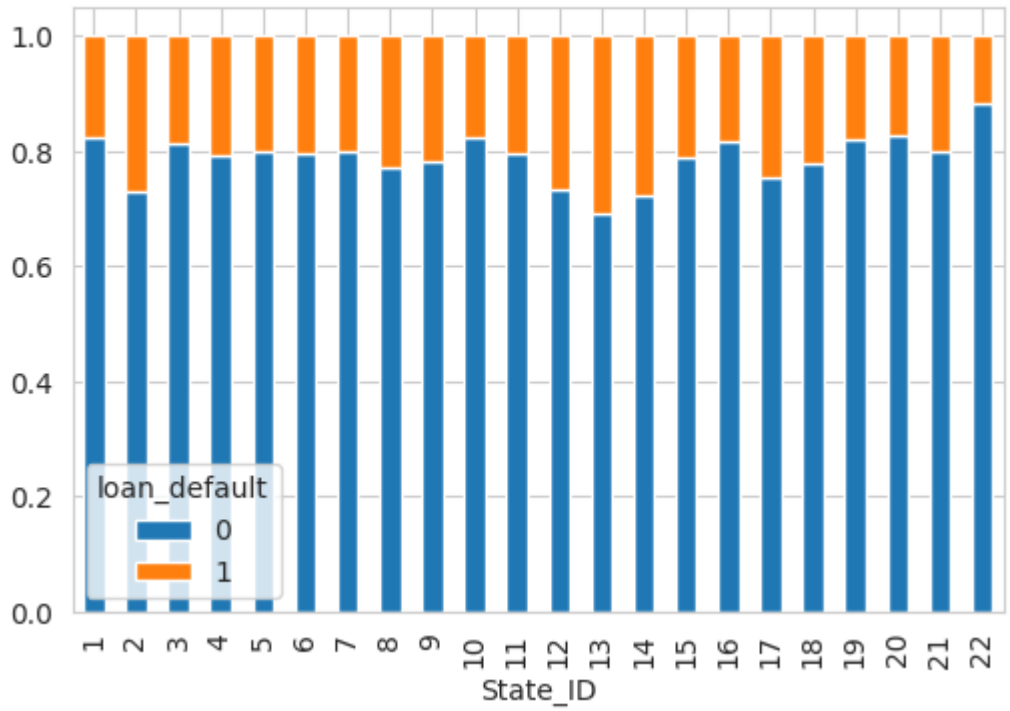
`Conclusion`

In [94]:

```
biv_cat_cat_analysis(train,target_='loan_default',cat='State_ID')
```



loan_default	0	1
State_ID		
1	0.822851	0.177149
2	0.728606	0.271394
3	0.813809	0.186191
4	0.792155	0.207845
5	0.801218	0.198782
6	0.794359	0.205641
7	0.798261	0.201739
8	0.770515	0.229485
9	0.782050	0.217950
10	0.824411	0.175589
11	0.795715	0.204285
12	0.734442	0.265558
13	0.693413	0.306587
14	0.724134	0.275866
15	0.788473	0.211527
16	0.816387	0.183613
17	0.754197	0.245803
18	0.779933	0.220067
19	0.819324	0.180676
20	0.827027	0.172973
21	0.801282	0.198718
22	0.881579	0.118421



In [95]:

```
from numpy import sqrt, abs, round
from scipy.stats import t as t_dist
from scipy.stats import norm
```

In [96]:

```
#list of all numerical_features
print(numerical_features)
```

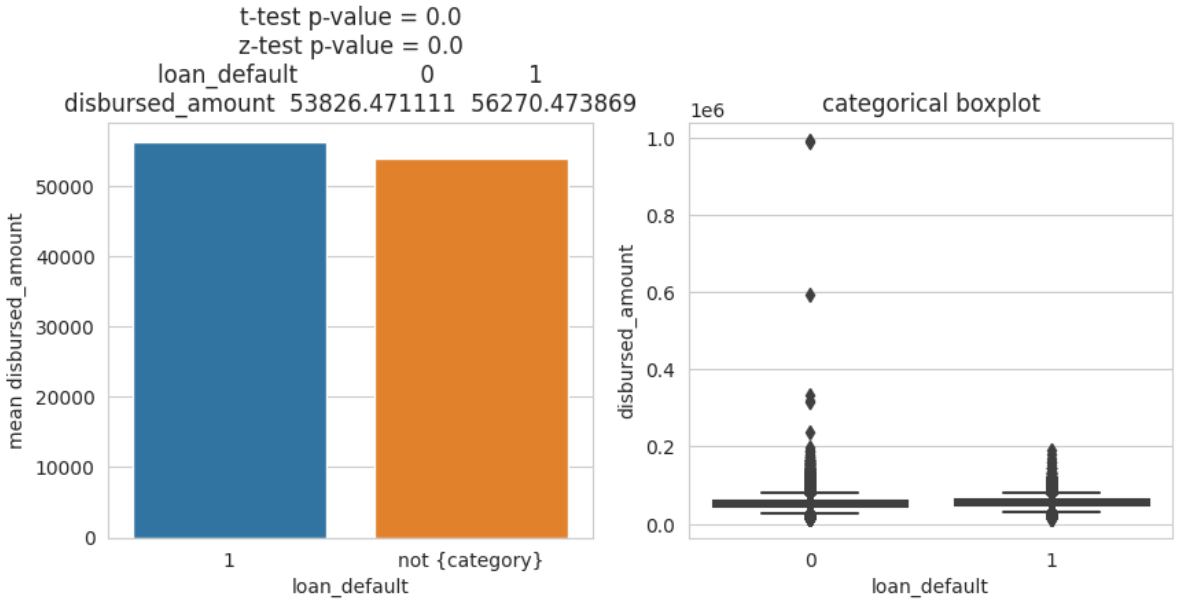
```
['disbursed_amount', 'asset_cost', 'ltv', 'Current_pincode_ID', 'PERFO
RM_CNS.SCORE', 'PRI.NO.OF.ACCTS', 'PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACC
TS', 'PRI.CURRENT.BALANCE', 'PRI.SANCTIONED.AMOUNT', 'PRI.DISBURSED.AM
OUNT', 'SEC.NO.OF.ACCTS', 'SEC.ACTIVE.ACCTS', 'SEC.OVERDUE.ACCTS', 'SE
C.CURRENT.BALANCE', 'SEC.SANCTIONED.AMOUNT', 'SEC.DISBURSED.AMOUNT',
'PRIMARY.INSTAL.AMT', 'SEC.INSTAL.AMT', 'NEW.ACCTS.IN.LAST.SIX.MONTH
S', 'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'NO.OF_INQUIRIES']
```

In [97]:

```
def two_sample_T(X1, X2, sd, sd1, n1, n2):
    ovr_sd = sqrt(sd**2/n1 + sd1**2/n2)
    t = (X1 - X2)/ovr_sd
    df = n1+n2-2
    pval = 2*(1 - t_dist.cdf(abs(t),df))
    return pval

def two_sample_Z(X1, X2, sigma, sigma1, n1, n2):
    ovr_sigma = sqrt(sigma**2/n1 + sigma1**2/n2)
    z = (X1 - X2)/ovr_sigma
    pval = 2*(1 - norm.cdf(abs(z)))
    return pval

# Bivariate Cont Cat Exploration Function
def bi_cat_con_analysis(data, continious_var, catategorical_var, category):
    #creating 2 samples
    x1 = data[continious_var][data[catategorical_var]==category][:]
    x2 = data[continious_var][~(data[catategorical_var]==category)][:]

    n1, n2 = x1.shape[0], x2.shape[0]
    m1, m2 = x1.mean(), x2.mean()
    std1, std2 = x1.std(), x2.std()

    #p-values calculation
    t_p_val = two_sample_T(m1, m2, std1, std2, n1, n2)
    z_p_val = two_sample_Z(m1, m2, std1, std2, n1, n2)

    #pivot table
    table = pd.pivot_table(data=data, values=continious_var, columns=catategorical_v

    #plotting
    plt.figure(figsize = (10,4),dpi=100)
    plt.subplot(1,2,1)
    sns.barplot([str(category),'not {category}'], [m1, m2])
    plt.ylabel(f'mean {continious_var}')
    plt.xlabel(catategorical_var)
    plt.title(f't-test p-value = {t_p_val}\nz-test p-value = {z_p_val}\n{table}')

  # boxplot
    plt.subplot(1,2,2)
    sns.boxplot(x=catategorical_var, y=continious_var, data=data)
    plt.title('categorical boxplot')
```

In [98]:

```
bi_cat_con_analysis(train, 'disbursed_amount', 'loan_default', 1)
```



Observation

p-value for both test is 0 i.e `loan_default` have a significan relationship with `disbursed_amount`

There are some outliers which are effecting out visualization lets remove them first

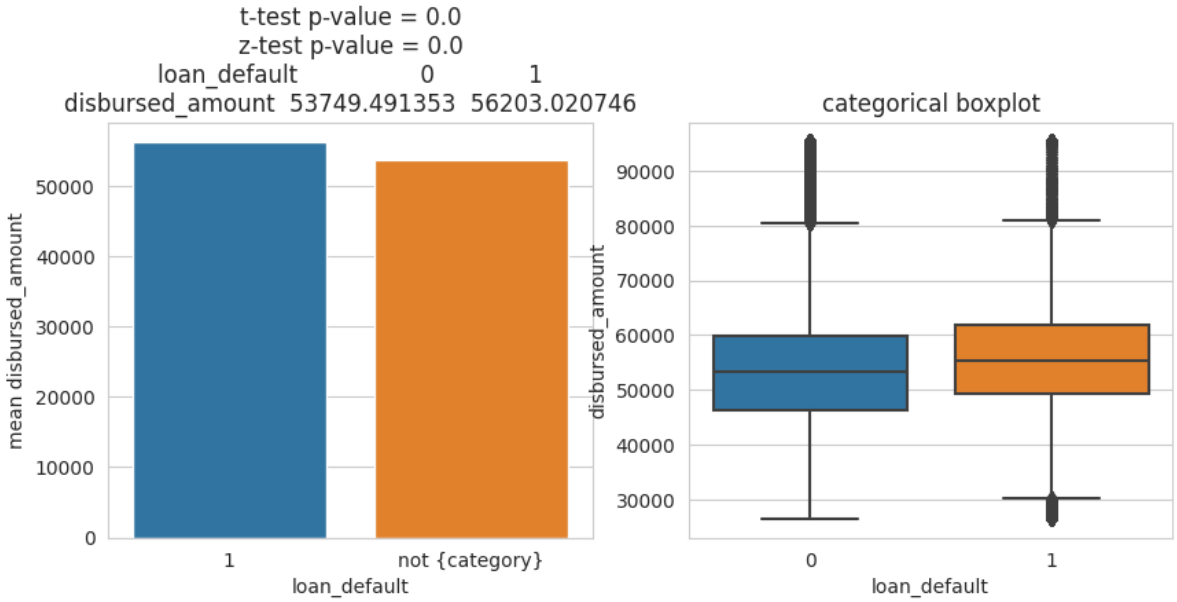
- see `average disbursed_amount` for 0-53826 and 1-5670 ==> but it is not vissible in 2nd-plot which is due to outliers

In [99]:

```
#removing outliers
upper_limit = np.percentile(train.disbursed_amount.values, 99) #after 99%tile
lower_limit = np.percentile(train.disbursed_amount.values, 1) #after 1%tile

#cliping
train['disbursed_amount'][train['disbursed_amount'] > upper_limit] = upper_limit #i
train['disbursed_amount'][train['disbursed_amount'] < lower_limit] = lower_limit
```

In [100]:

```
bi_cat_con_analysis(train, 'disbursed_amount', 'loan_default',1)
```



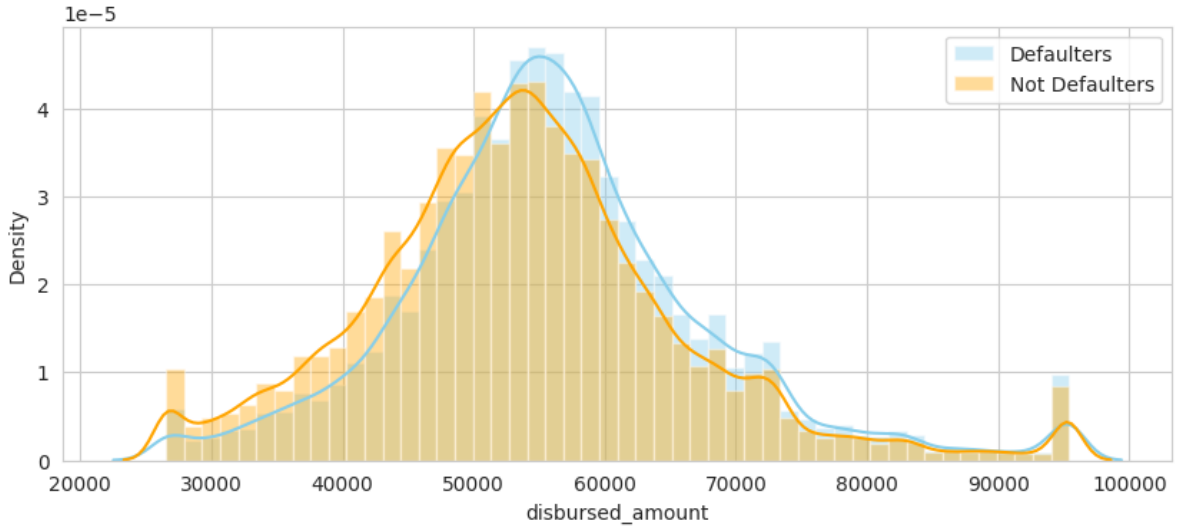
Observation

see `disbursed_amount` of defaulter-1 is higher then `disbursed_amount` of not-default-0

Now Ploting Distribution Plot for Defaulter and Non-Defaulter

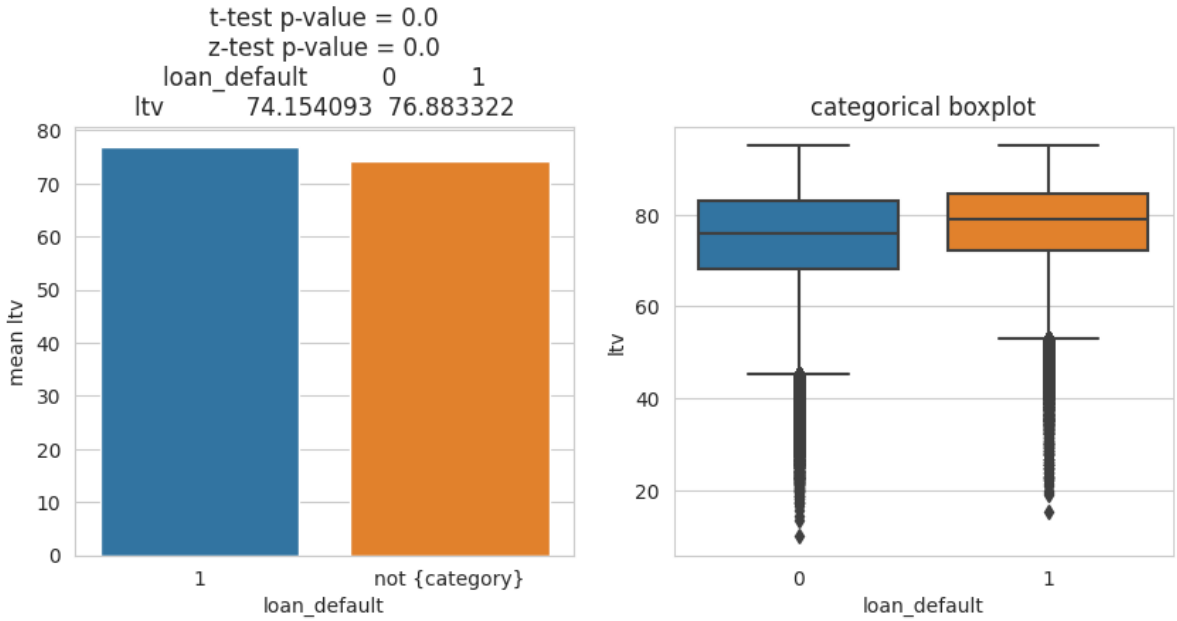
In [101]:

```
#alternate way of visualising disbursed_amount distribution for defaulters & non def
plt.figure(figsize = (10,4),dpi=100)
sns.distplot(train["disbursed_amount"][train['loan_default'] == 1] , color='skyblue'
sns.distplot(train["disbursed_amount"][train['loan_default'] == 0] , color='orange',

plt.legend()
plt.show()
```



In [102]:

```
bi_cat_con_analysis(train, 'ltv', 'loan_default', 1)
```



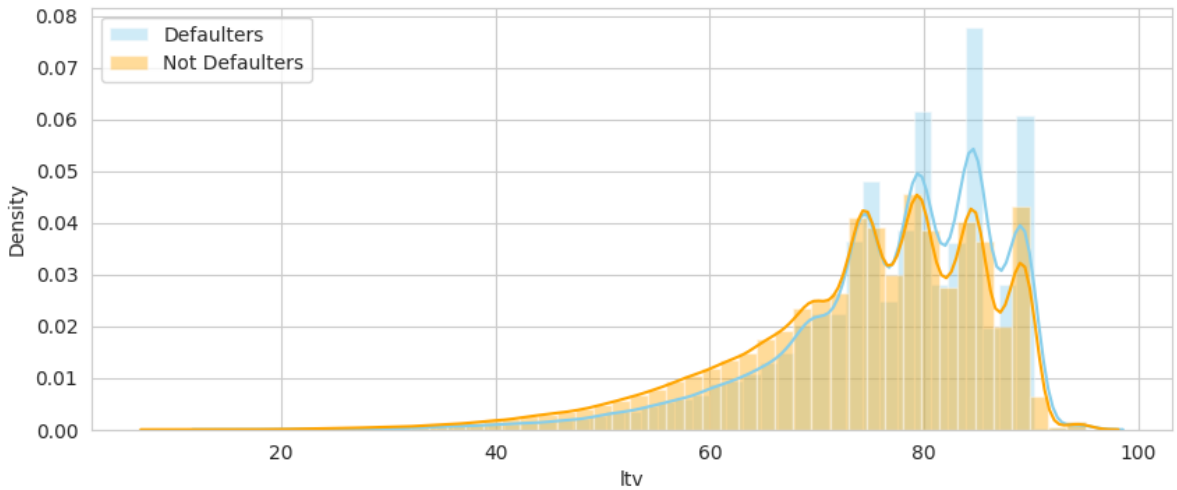
Observation

`ltv` is higher for defaults-1, see `average itv values` and in 2nd-plot

It mean if user have taken loan of rs10 and takes loan amount say rs9 with rs1 downpament then this user is more prone to default

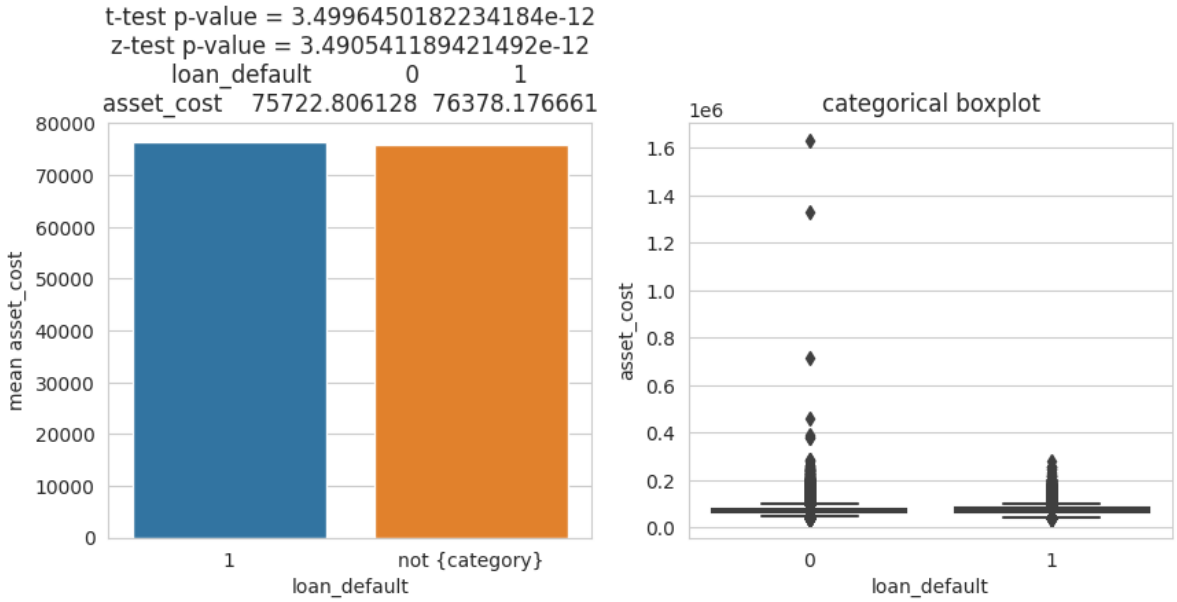
In [103]:

```
#alternate way of visualising ltv distribution for defaulters & non defaulters
plt.figure(figsize=(10,4),dpi=100)
sns.distplot(train["ltv"][train['loan_default'] == 1] , color='skyblue', label='Defa
sns.distplot(train["ltv"][train['loan_default'] == 0] , color='orange', label='Not D

plt.legend()
plt.show()
```



In [104]:

```
bi_cat_con_analysis(train, 'asset_cost', 'loan_default', 1)
```

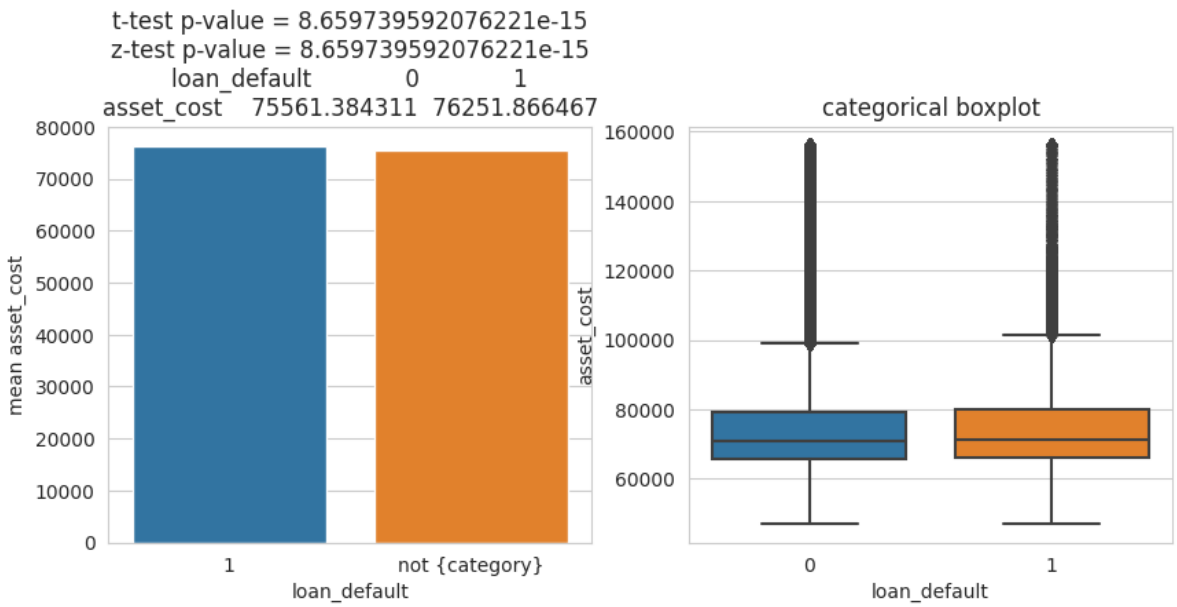


In [105]:

```
#removing outliers
upper_limit = np.percentile(train.asset_cost.values, 99) #after 99%tile
lower_limit = np.percentile(train.asset_cost.values, 1) #after 1%tile

#cliping
train['asset_cost'][train['asset_cost'] > upper_limit] = upper_limit #if value is gr
train['asset_cost'][train['asset_cost'] < lower_limit] = lower_limit
```

In [106]:

```
bi_cat_con_analysis(train, 'asset_cost', 'loan_default', 1)
```

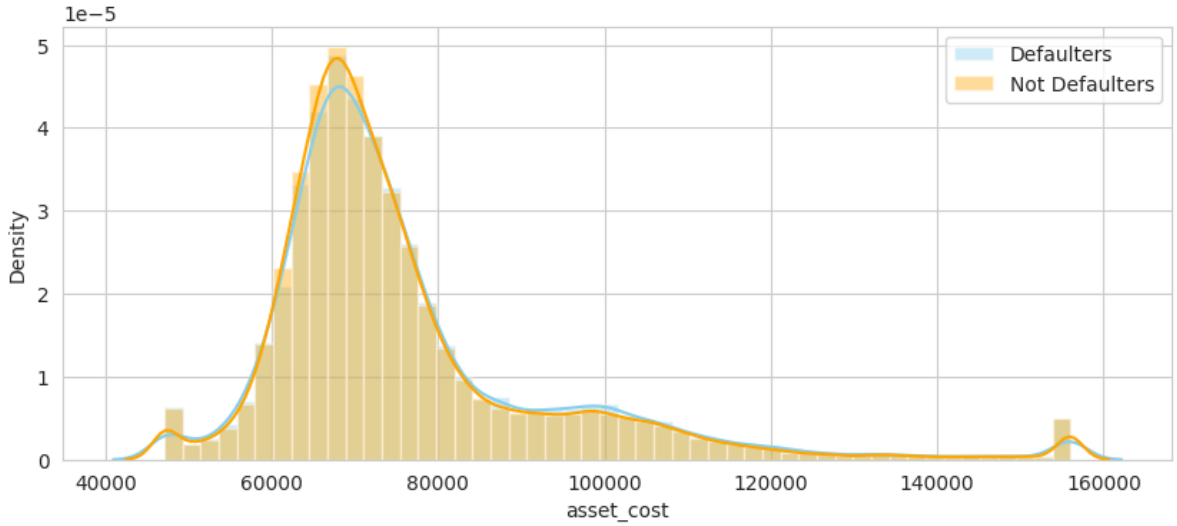


Observation

asset_cost is not that effective feature

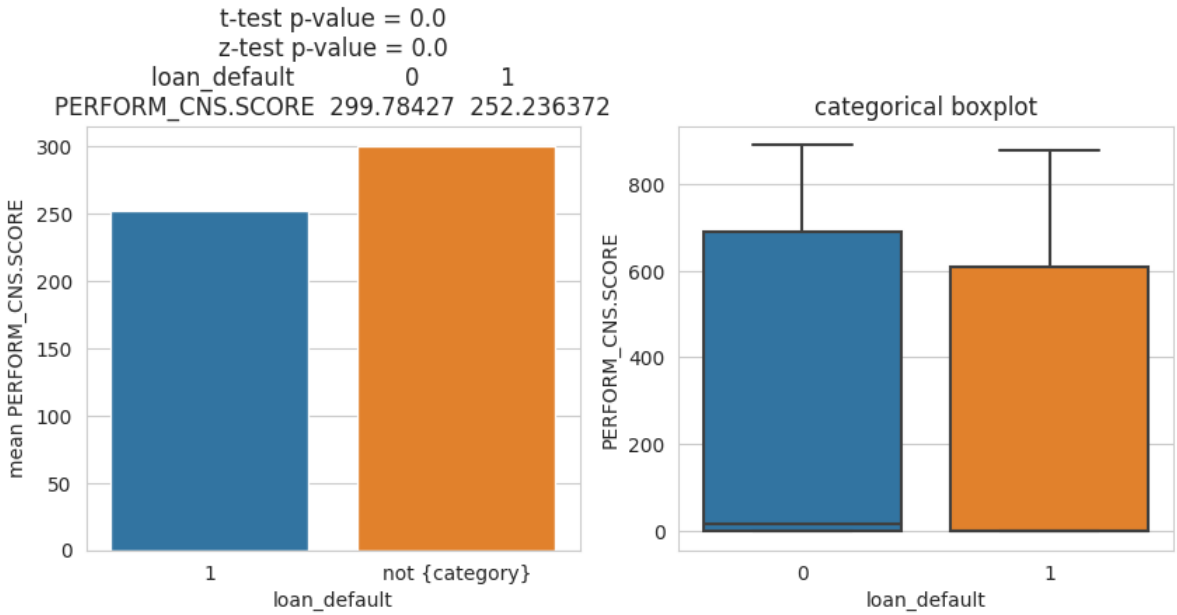
In [107]:

```
#alternate way of visualising asset_cost distribution for defaulters & non defaulter
plt.figure(figsize=(10,4),dpi=100)
sns.distplot(train['asset_cost'][train['loan_default'] == 1] , color='skyblue', labe
sns.distplot(train['asset_cost'][train['loan_default'] == 0] , color='orange', label

plt.legend()
plt.show()
```



In [108]:

```
bi_cat_con_analysis(train, 'PERFORM_CNS.SCORE', 'loan_default', 1)
```



Observation

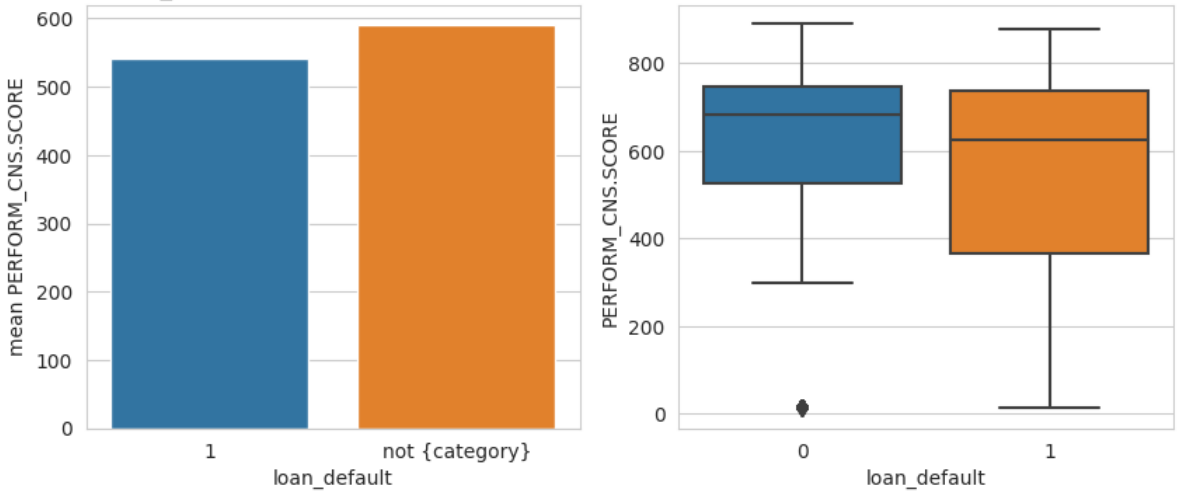
By our previous analysis we know most of these values are 0, lets plot this after removing entries having 0 in them

In [109]:

```
#performance_cns_score vs loan default with 0s removed
bi_cat_con_analysis(train[train['PERFORM_CNS.SCORE'] > 0], 'PERFORM_CNS.SCORE', 'loa
```

t-test p-value = 0.0
z-test p-value = 0.0

loan_default	0	1
PERFORM_CNS.SCORE	590.679691	541.870835

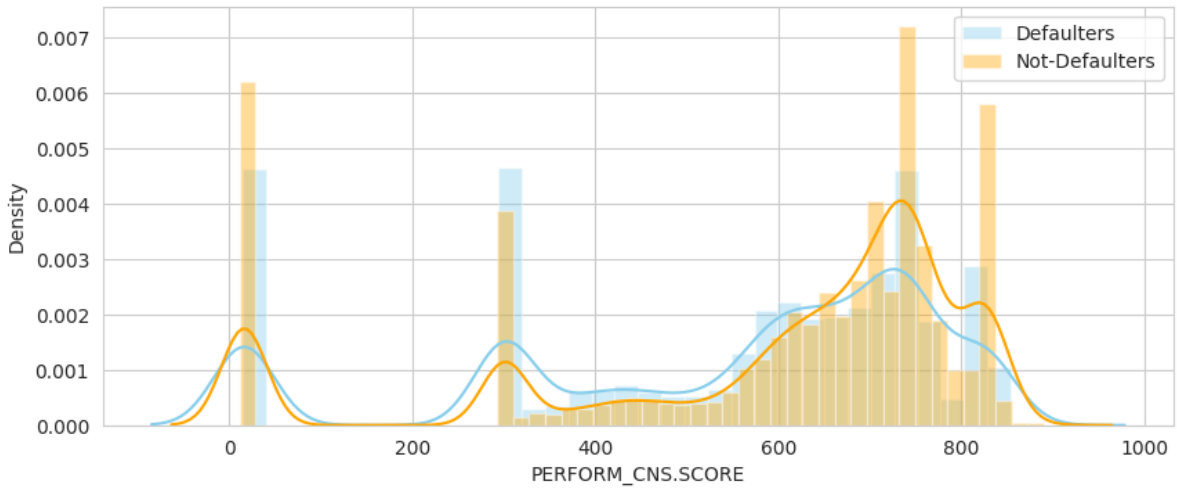


Observation

For non defaulter we have high value of `PERFORM_CNS.SCORE` i.e 0-591 appx

In [110]:

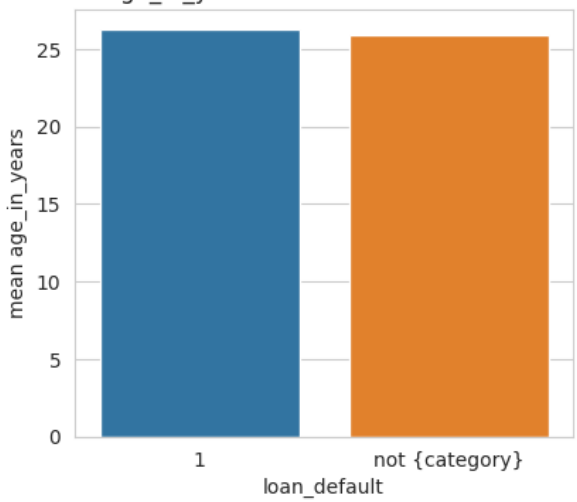
```
#alternate way of visualising PERFORM_CNS.SCORE distribution for defaulters & non de
plt.figure(figsize=(10,4),dpi=100)
sns.distplot(train['PERFORM_CNS.SCORE'][(train['loan_default'] == 1) & (train['PERFC
sns.distplot(train['PERFORM_CNS.SCORE'][(train['loan_default'] == 0) & (train['PERFC

plt.legend()
plt.show()
```

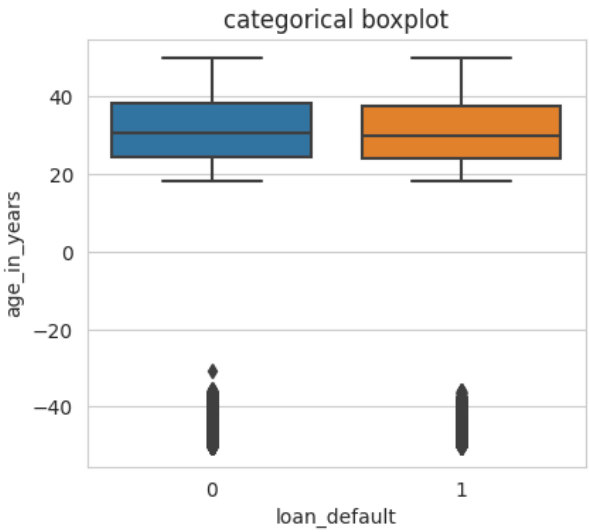


In [111]:

```
#age_in_years vs loan default with 0s removed
bi_cat_con_analysis(train, 'age_in_years', 'loan_default', 1)
```

```
t-test p-value = 0.0003695364157214609
z-test p-value = 0.00036946289979744584
loan_default        0        1
age_in_years  25.863074  26.261733
```



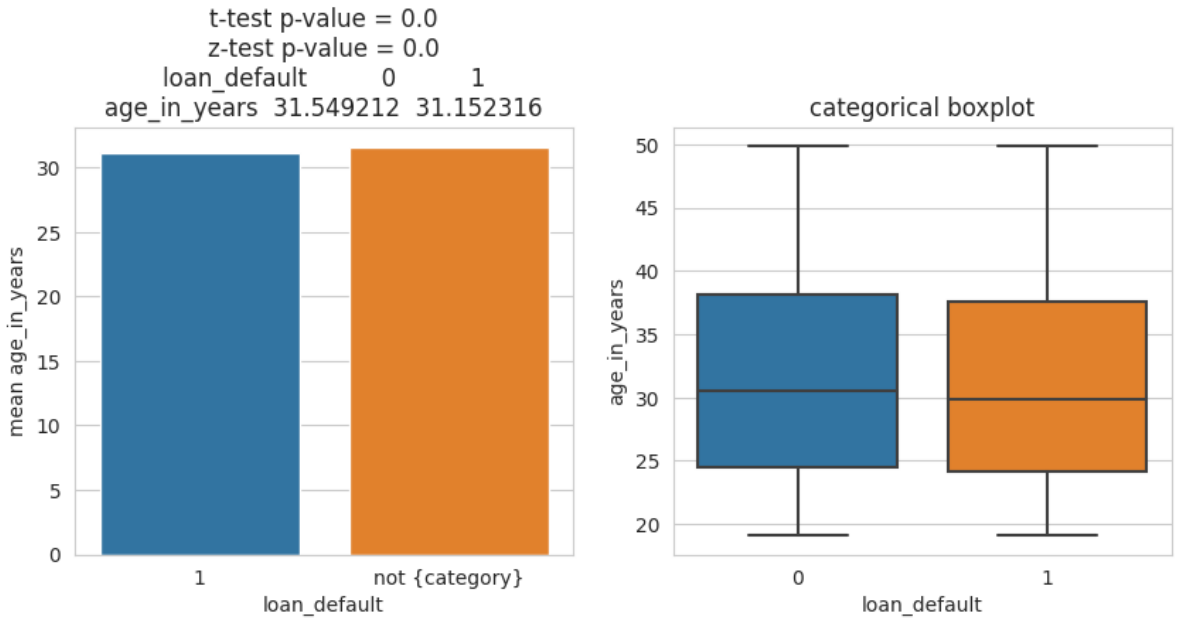


In [112]:

```
#removing outliers
upper_limit = np.percentile(train.age_in_years.values, 100) #after 100%tile
lower_limit = np.percentile(train.age_in_years.values, 9) #after 9%tile

#cliping
train['age_in_years'][train['age_in_years'] > upper_limit] = upper_limit #if value i
train['age_in_years'][train['age_in_years'] < lower_limit] = lower_limit
```

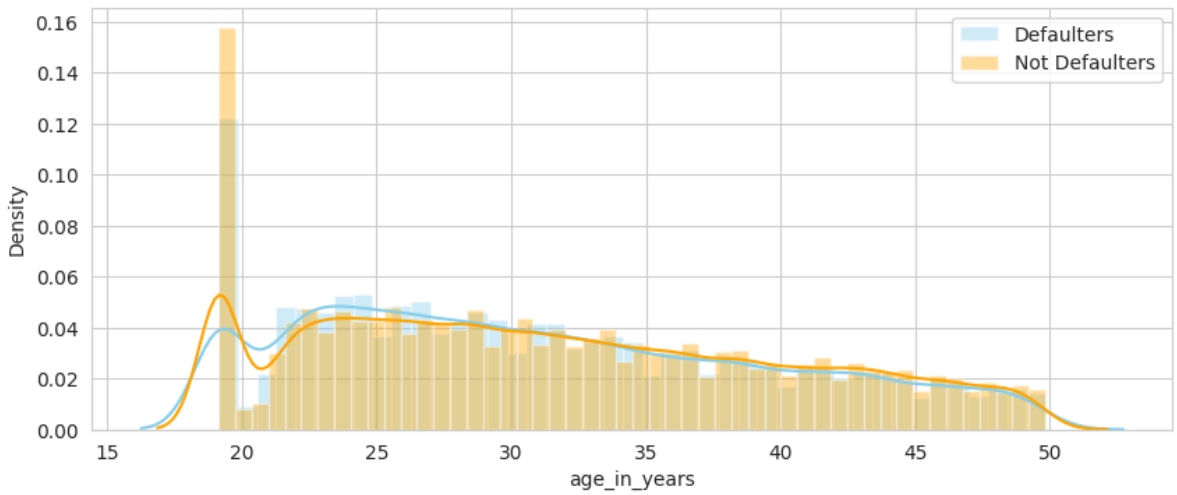
In [113]:

```
#age_in_years vs loan default with 0s removed
bi_cat_con_analysis(train, 'age_in_years', 'loan_default', 1)
```



In [114]:

```
#alternate way of visualising age_in_years distribution for defaulters & non default
plt.figure(figsize=(10,4),dpi=100)
sns.distplot(train['age_in_years'][train['loan_default'] == 1] , color='skyblue', la
sns.distplot(train['age_in_years'][train['loan_default'] == 0] , color='orange', lab

plt.legend()
plt.show()
```



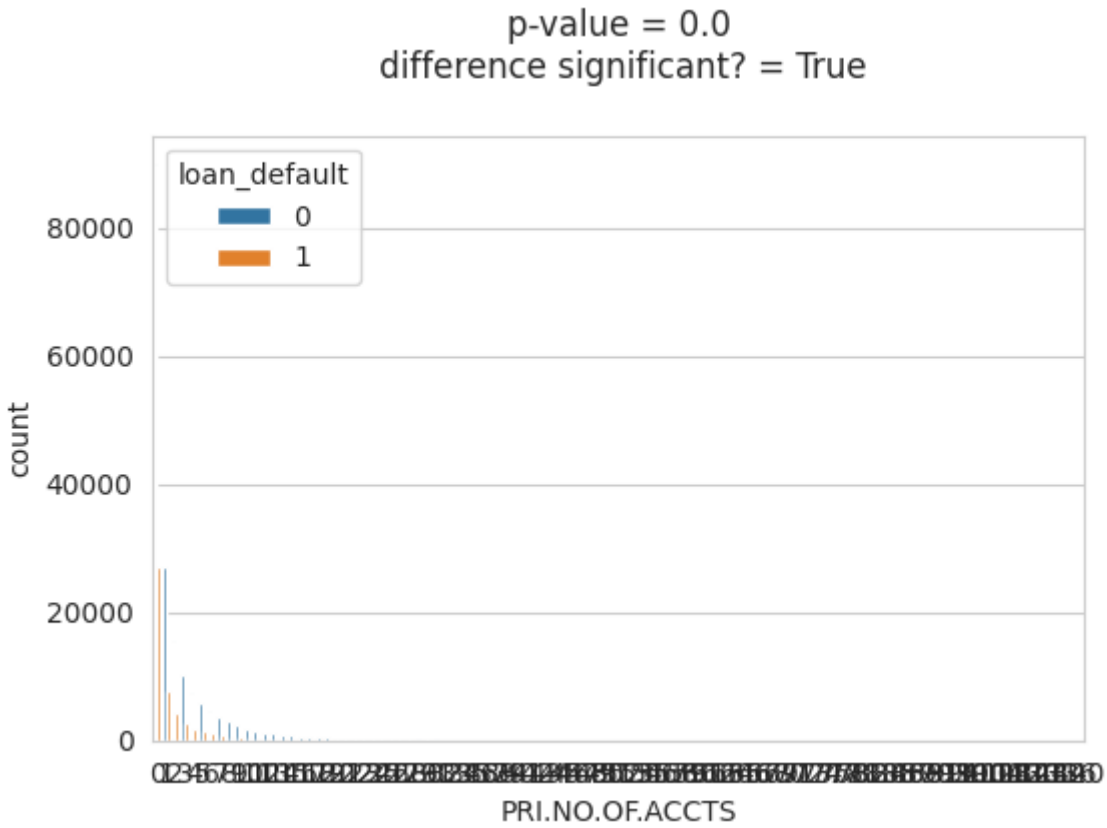
# of primary accounts vs default rate

In [115]:

```
# function removing outliers from primary account
def kick_outlieres(data,feature):
    ulimit = np.percentile(train.credit_history_in_months.values, 99)
    data[feature][data[feature] > ulimit] = ulimit
    return data

df =  kick_outlieres(train,'PRI.NO.OF.ACCTS')
```

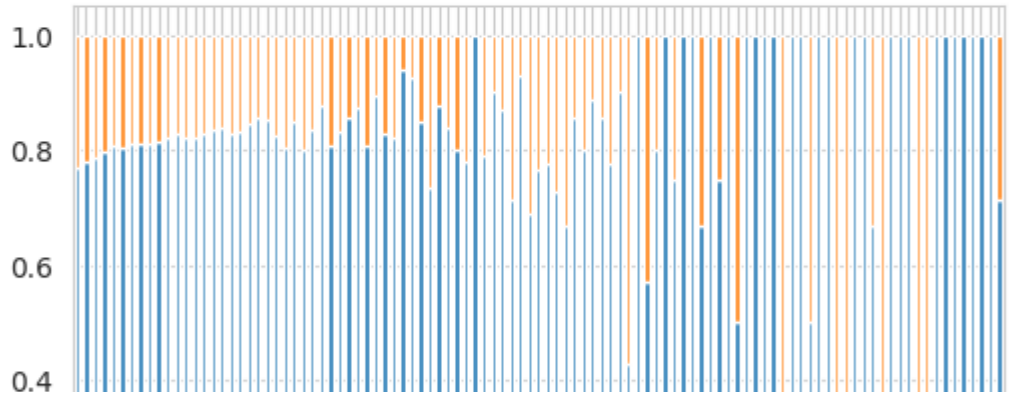
In [116]:

```
# Observing Default rate vs primary number of accounts
biv_cat_cat_analysis(train, 'loan_default', 'PRI.NO.OF.ACCTS' )
```



loan_default	0	1
PRI.NO.OF.ACCTS		
0	0.768687	0.231313
1	0.778804	0.221196
2	0.785230	0.214770
3	0.797311	0.202689
4	0.808216	0.191784
...	...	...
124	1.000000	NaN
131	1.000000	NaN
132	1.000000	NaN
136	1.000000	NaN
140	0.714286	0.285714

[103 rows x 2 columns]



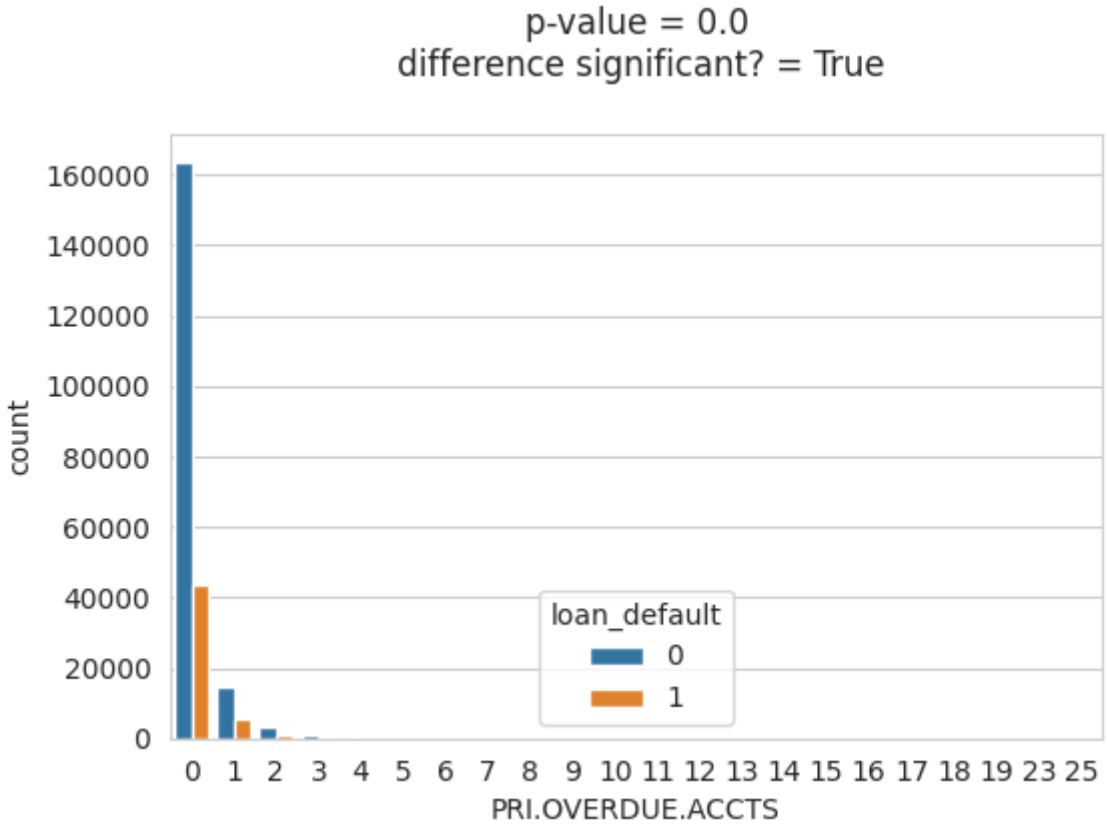
`Observation`

If `PRI.NO.OF.ACCTS` are more then 1 then we can say customer have take lone previously and what we can see is if person have taken loan previously, he/she is less prone to defaulting

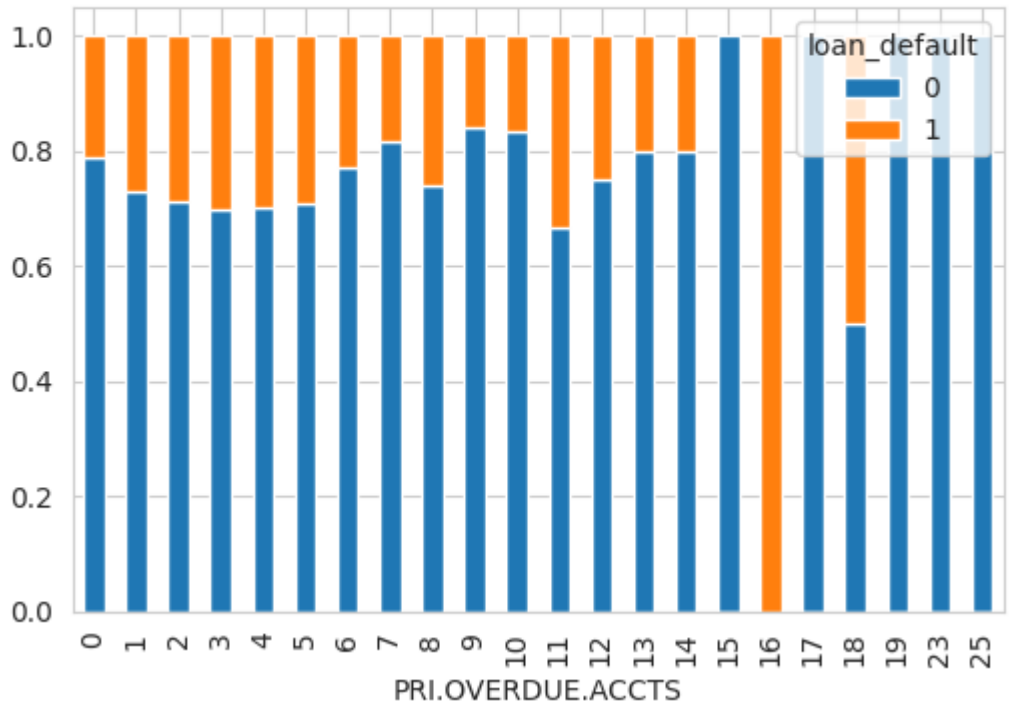
`Binning` can be used to see these graphs more clearly but still we can get intution

In [117]:

```
# Similarly for overdue accounts vs loan default rate
biv_cat_cat_analysis(train, 'loan_default', 'PRI.OVERDUE.ACCTS' )
```



loan_default	0	1
PRI.OVERDUE.ACCTS		
0	0.790177	0.209823
1	0.730496	0.269504
2	0.711530	0.288470
3	0.699667	0.300333
4	0.702970	0.297030
5	0.710843	0.289157
6	0.770833	0.229167
7	0.815789	0.184211
8	0.740741	0.259259
9	0.840000	0.160000
10	0.833333	0.166667
11	0.666667	0.333333
12	0.750000	0.250000
13	0.800000	0.200000
14	0.800000	0.200000
15	1.000000	NaN
16	NaN	1.000000
17	1.000000	NaN
18	0.500000	0.500000
19	1.000000	NaN
23	1.000000	NaN
25	1.000000	NaN

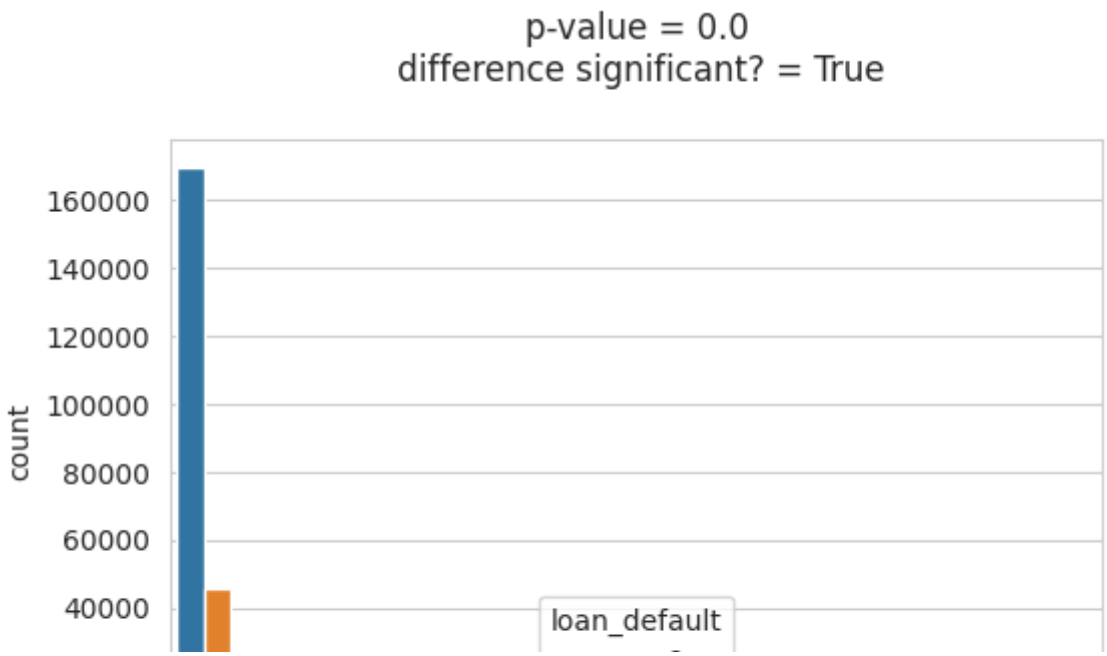


Observation

As `PRI.OVERDUE.ACCTS` are increasing defaults are incresing to

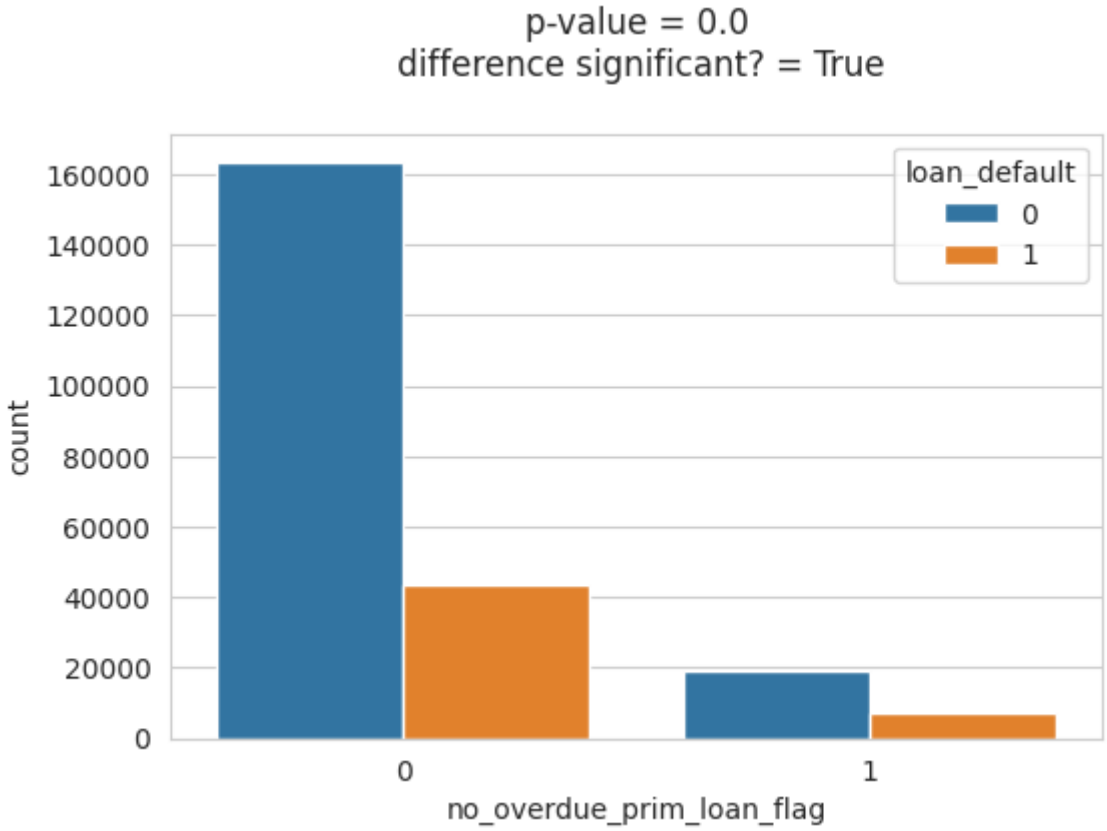
In [118]:

```
# Delinquent accounts vs loan default rate
biv_cat_cat_analysis(train, 'loan_default', 'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS' )
```

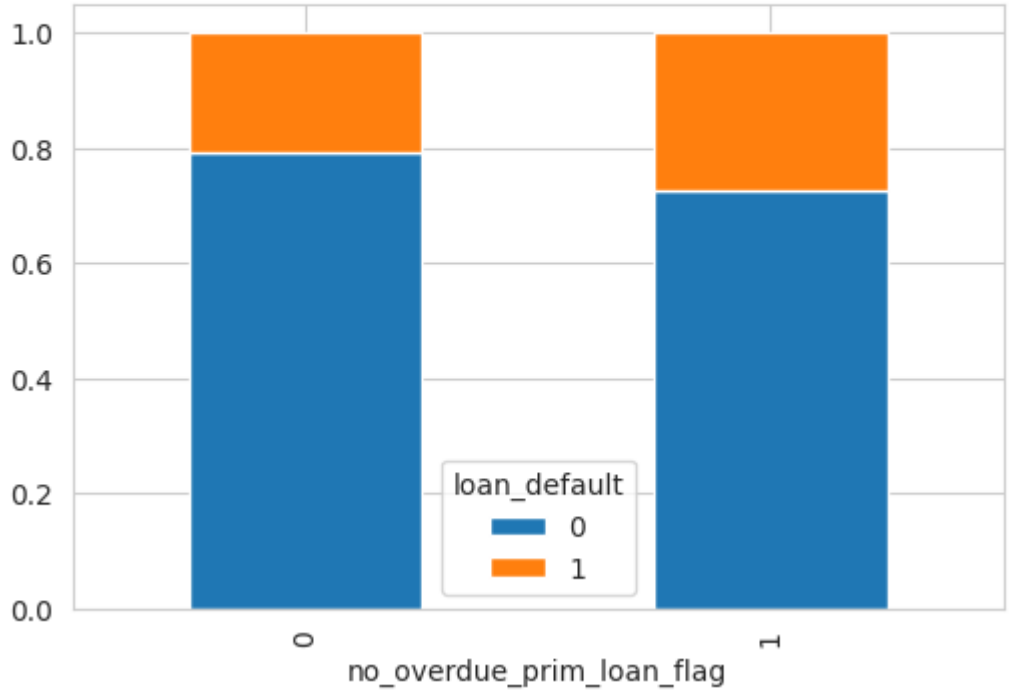


In [119]:

```
# Overdue accounts vs default rate
biv_cat_cat_analysis(train,'loan_default','no_overdue_prim_loan_flag')
```



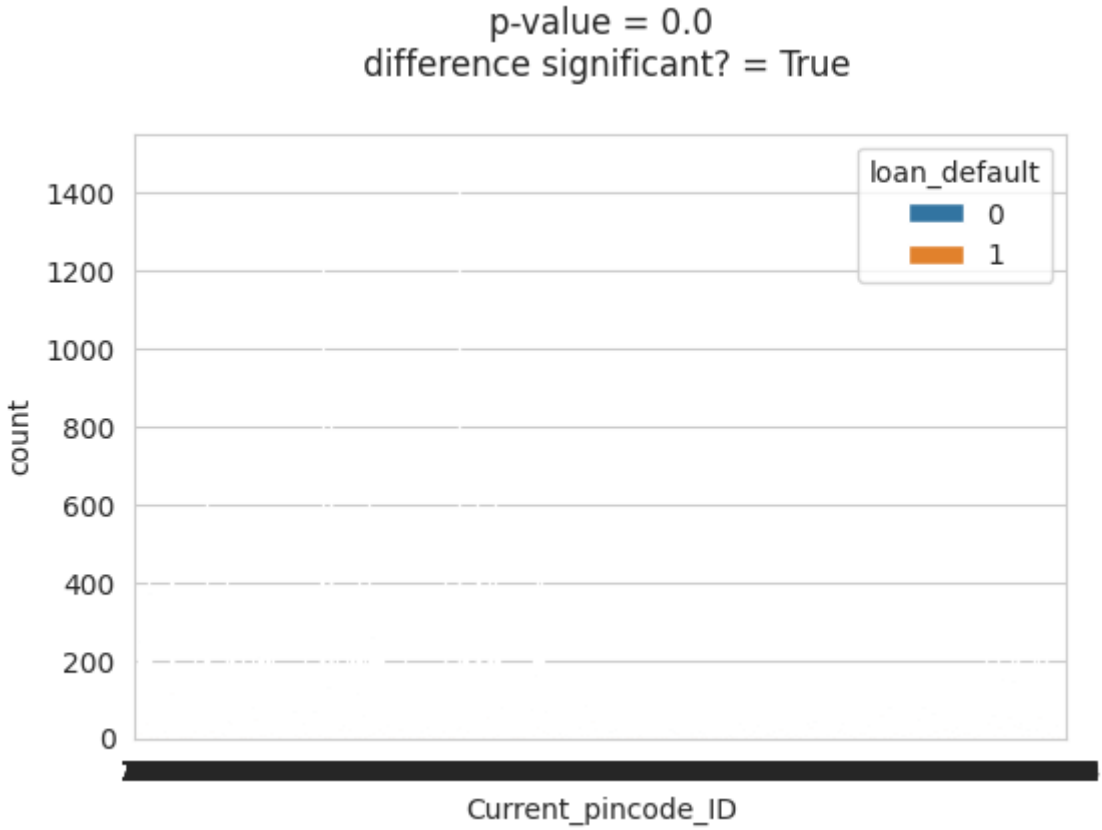
loan_default	0	1
no_overdue_prim_loan_flag		
0	0.790177	0.209823
1	0.725861	0.274139



Default rate[loan_default] vs ID features

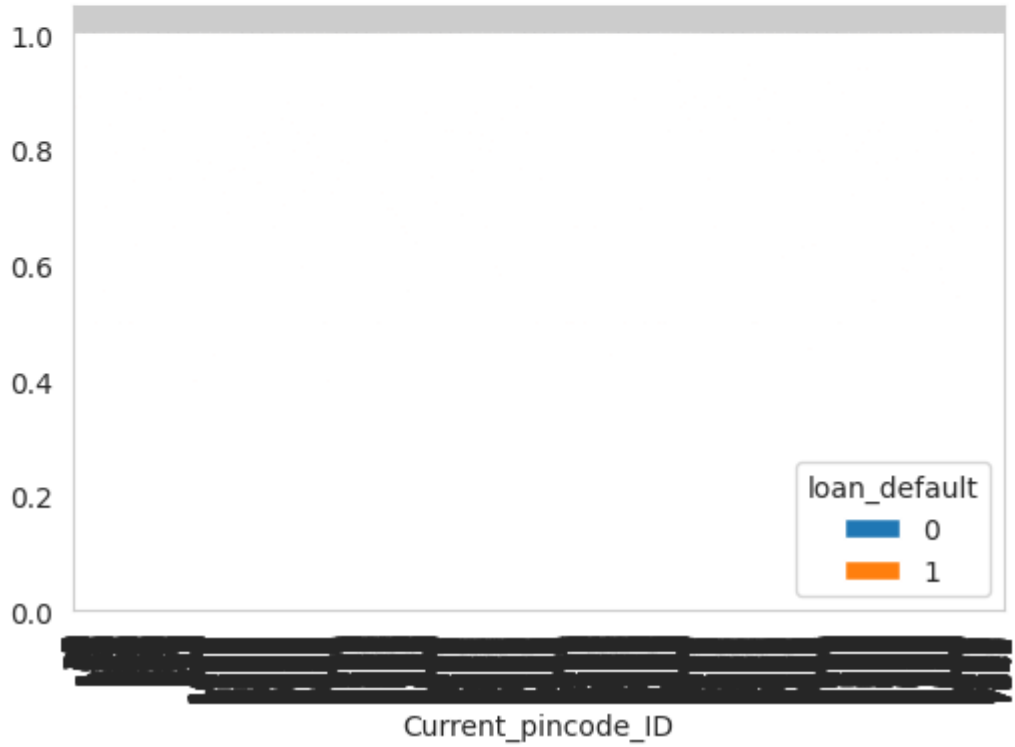
In [120]:

```
#default rate vs binned pin code
biv_cat_cat_analysis(train,'loan_default','Current_pincode_ID')
```



loan_default	0	1
Current_pincode_ID		
1	0.769231	0.230769
2	0.819444	0.180556
3	0.880000	0.120000
4	0.829545	0.170455
5	0.730233	0.269767
...	...	...
7341	0.857143	0.142857
7342	1.000000	NaN
7343	0.500000	0.500000
7344	1.000000	NaN
7345	1.000000	NaN

[6698 rows x 2 columns]



Observation

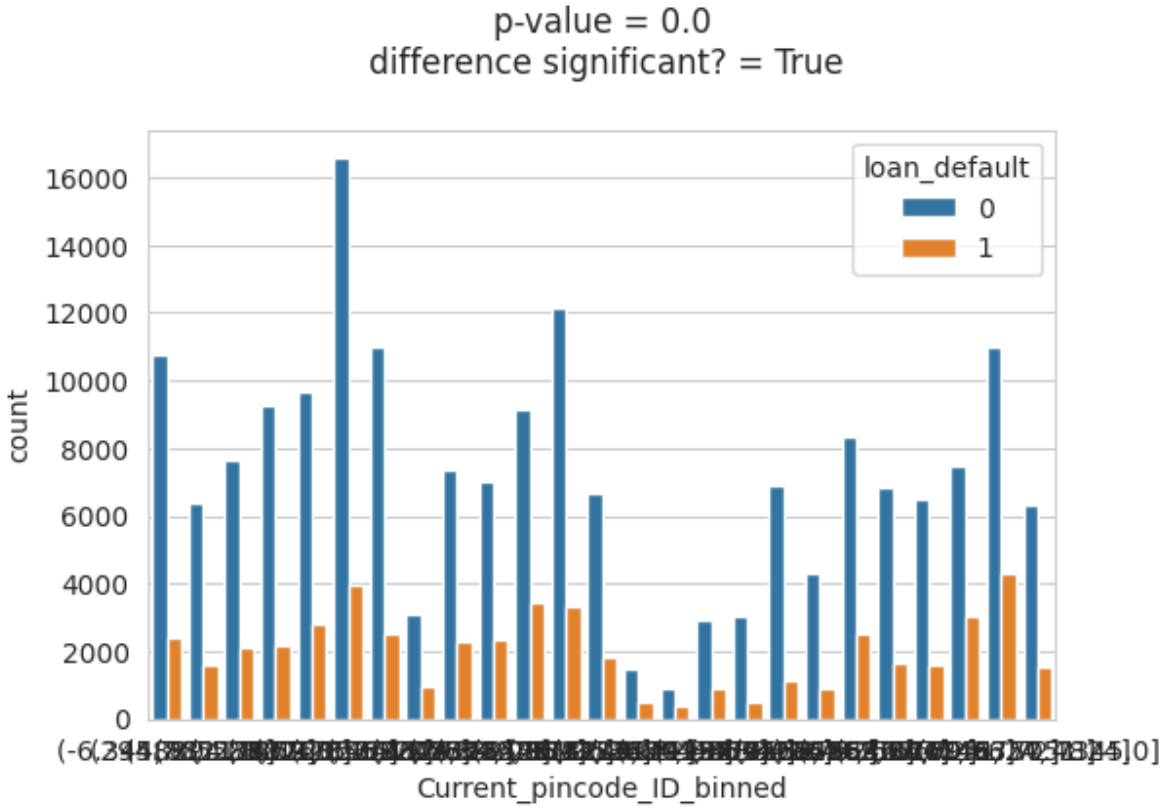
We require bining for clear views lets do that first

In [121]:

```
#binning pin code
train['Current_pincode_ID_binned'] = pd.cut(train['Current_pincode_ID'], bins=25)
```

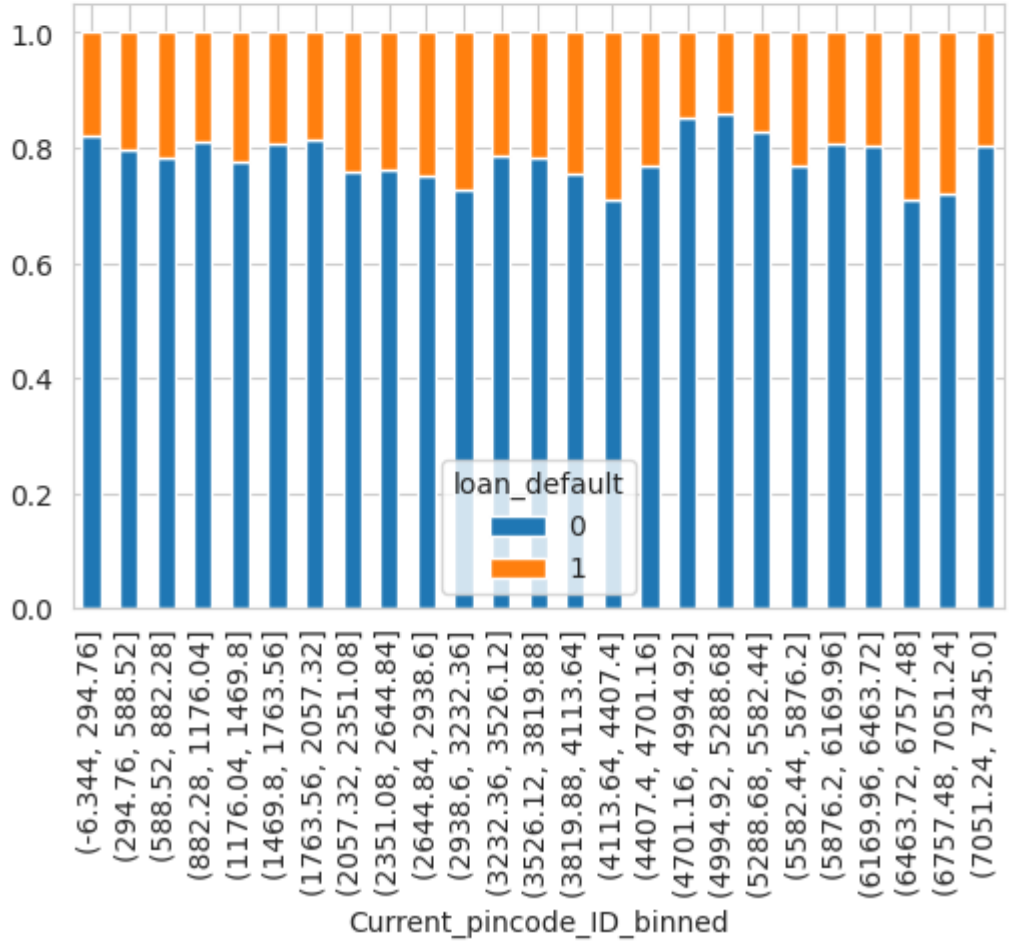
In [122]:

```
#default rate vs binned pin code after binning
biv_cat_cat_analysis(train,'loan_default','Current_pincode_ID_binned')
```



loan_default	0	1
Current_pincode_ID_binned		
(-6.344, 294.76]	0.819125	0.180875
(294.76, 588.52]	0.798196	0.201804
(588.52, 882.28]	0.781606	0.218394
(882.28, 1176.04]	0.810614	0.189386
(1176.04, 1469.8]	0.774046	0.225954
(1469.8, 1763.56]	0.808277	0.191723
(1763.56, 2057.32]	0.814167	0.185833
(2057.32, 2351.08]	0.759861	0.240139
(2351.08, 2644.84]	0.761890	0.238110
(2644.84, 2938.6]	0.750856	0.249144
(2938.6, 3232.36]	0.725924	0.274076
(3232.36, 3526.12]	0.786806	0.213194
(3526.12, 3819.88]	0.783386	0.216614
(3819.88, 4113.64]	0.753460	0.246540
(4113.64, 4407.4]	0.709283	0.290717

(4407.4, 4701.16]	0.768521	0.231479
(4701.16, 4994.92]	0.853356	0.146644
(4994.92, 5288.68]	0.859256	0.140744
(5288.68, 5582.44]	0.826819	0.173181
(5582.44, 5876.2]	0.768000	0.232000
(5876.2, 6169.96]	0.806786	0.193214
(6169.96, 6463.72]	0.802145	0.197855
(6463.72, 6757.48]	0.711162	0.288838
(6757.48, 7051.24]	0.720013	0.279987
(7051.24, 7345.0]	0.802578	0.197422

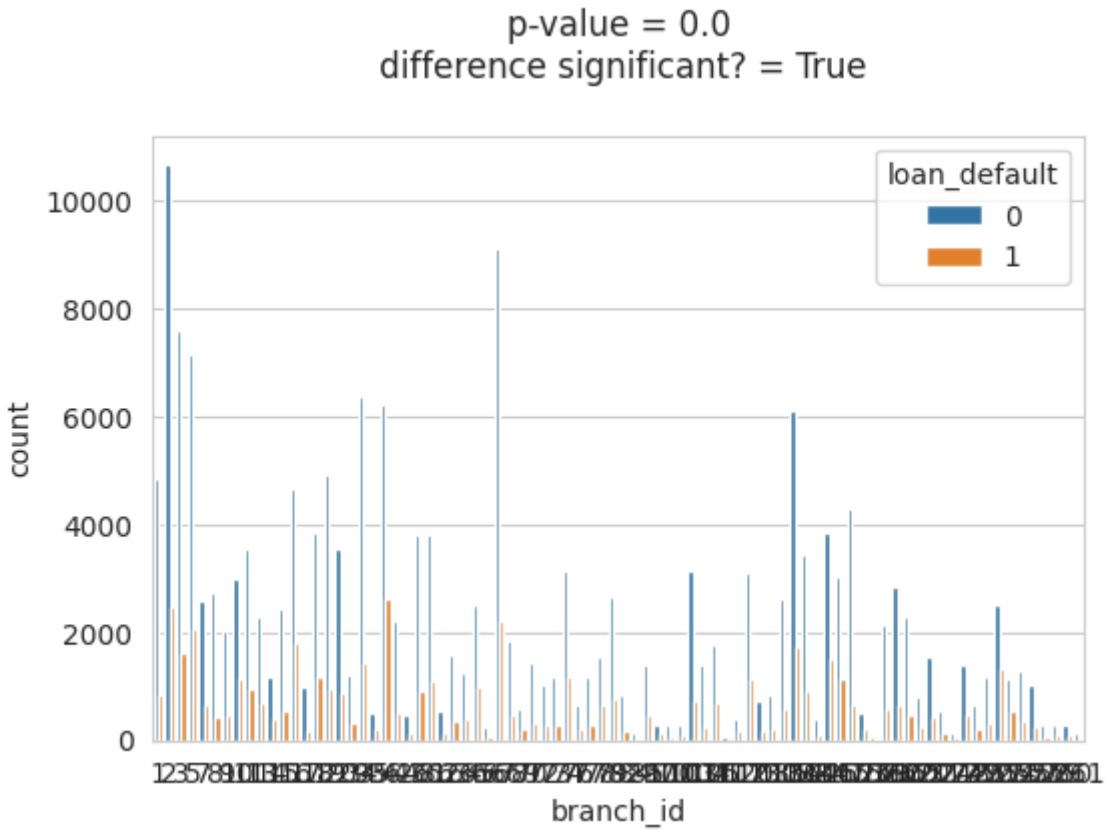


`Observatoin`

No clear patern are seen like more default in some reason kinde of stuff, as we were finding in previous analysis and we came to this conclusion that pin code have some patern in them, it is not the case

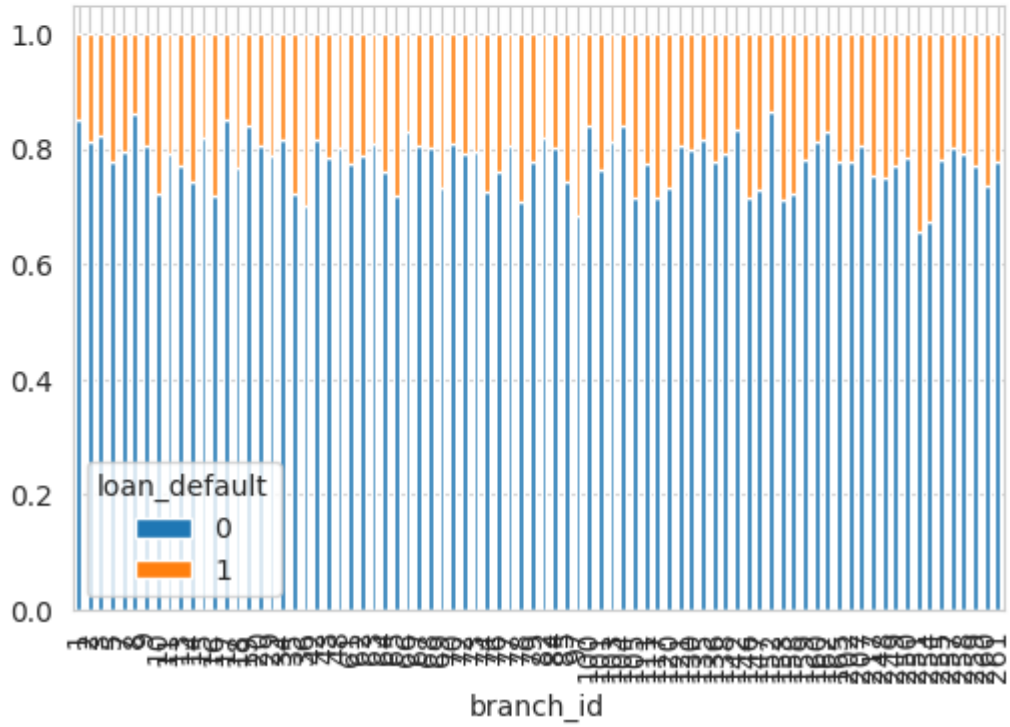
In [123]:

```
#default rate vs branch id
biv_cat_cat_analysis(train,'loan_default','branch_id')
```



loan_default	0	1
branch_id		
1	0.850587	0.149413
2	0.813137	0.186863
3	0.825135	0.174865
5	0.777934	0.222066
7	0.796400	0.203600
...	...	...
257	0.801752	0.198248
258	0.794118	0.205882
259	0.771676	0.228324
260	0.736559	0.263441
261	0.778409	0.221591

[82 rows x 2 columns]



`Observation`

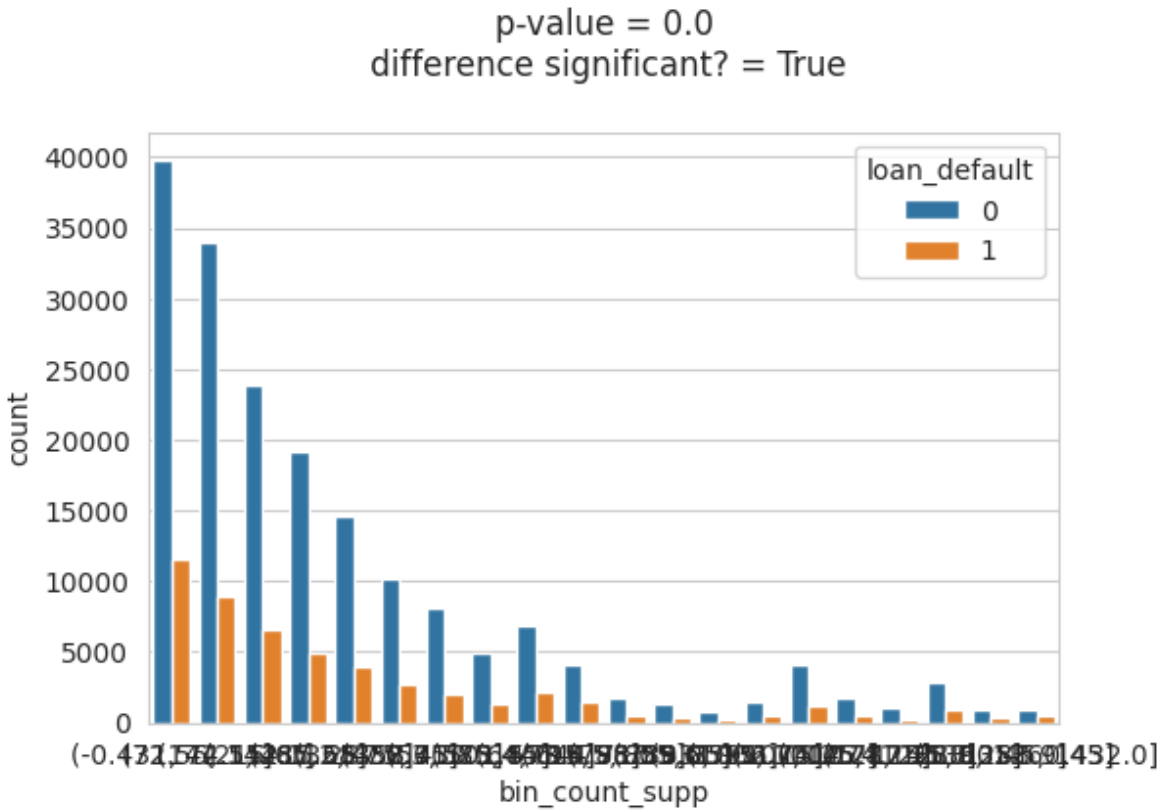
It seams branch_id are all randome values

In [124]:

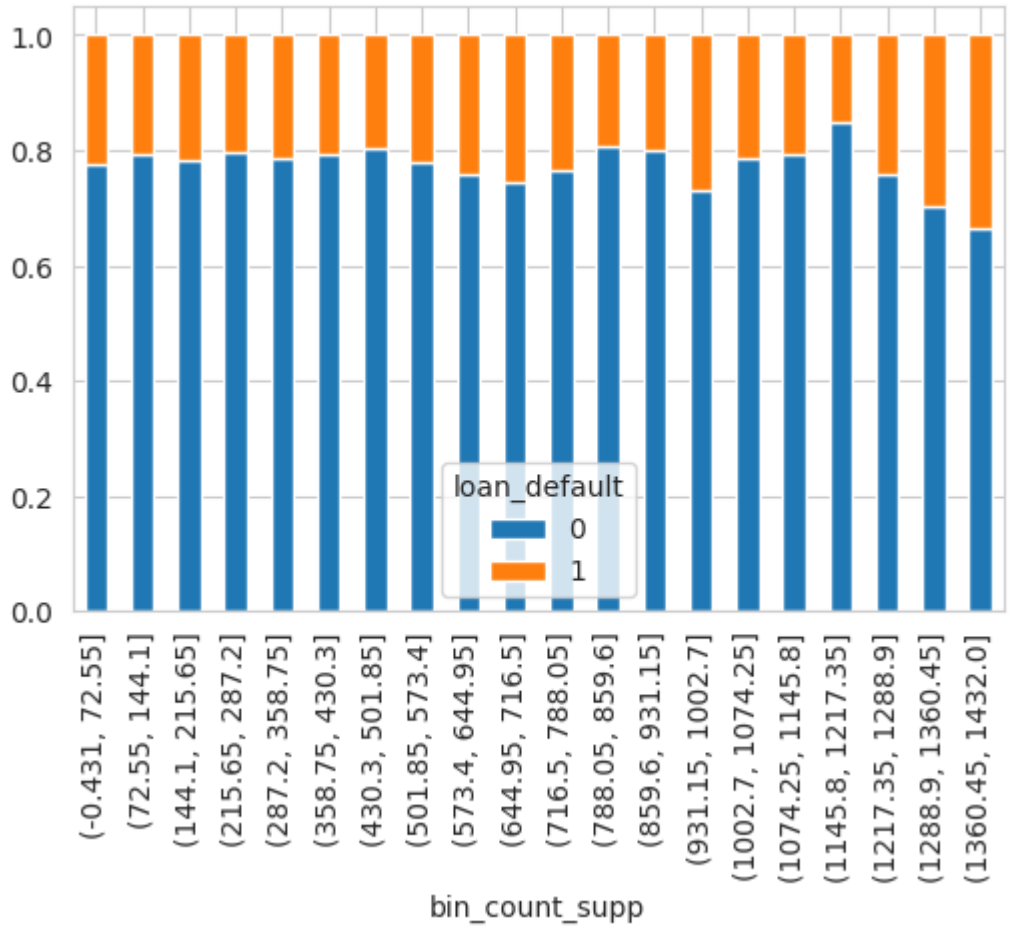
```
#creating bins on basis of [ count of defaulter customers for each supplier ]
df = train[['supplier_id','loan_default']].groupby(['supplier_id'])['loan_default'].
.sort_values(['loan_default_count'],ascending=False)

train = train.merge(df, on='supplier_id', how='left')

#binning
train['bin_count_supp'] = pd.cut(train['loan_default_count'], bins=20)
```

In [125]:

```
#supplier count frequency vs default rate
biv_cat_cat_analysis(train,'loan_default','bin_count_supp')
```



loan_default	0	1
bin_count_supp		
(-0.431, 72.55]	0.775677	0.224323
(72.55, 144.1]	0.792914	0.207086
(144.1, 215.65]	0.782653	0.217347
(215.65, 287.2]	0.797593	0.202407
(287.2, 358.75]	0.786241	0.213759
(358.75, 430.3]	0.791494	0.208506
(430.3, 501.85]	0.803211	0.196789
(501.85, 573.4]	0.780684	0.219316
(573.4, 644.95]	0.759065	0.240935
(644.95, 716.5]	0.744796	0.255204
(716.5, 788.05]	0.766240	0.233760
(788.05, 859.6]	0.806952	0.193048
(859.6, 931.15]	0.799324	0.200676
(931.15, 1002.7]	0.731821	0.268179
(1002.7, 1074.25]	0.784630	0.215370
(1074.25, 1145.8]	0.791889	0.208111
(1145.8, 1217.35]	0.849587	0.150413
(1217.35, 1288.9]	0.759800	0.240200
(1288.9, 1360.45]	0.703077	0.296923
(1360.45, 1432.0]	0.663408	0.336592



`Observation`

Big `supliers` means more customers then these supliers have more defaulting customer

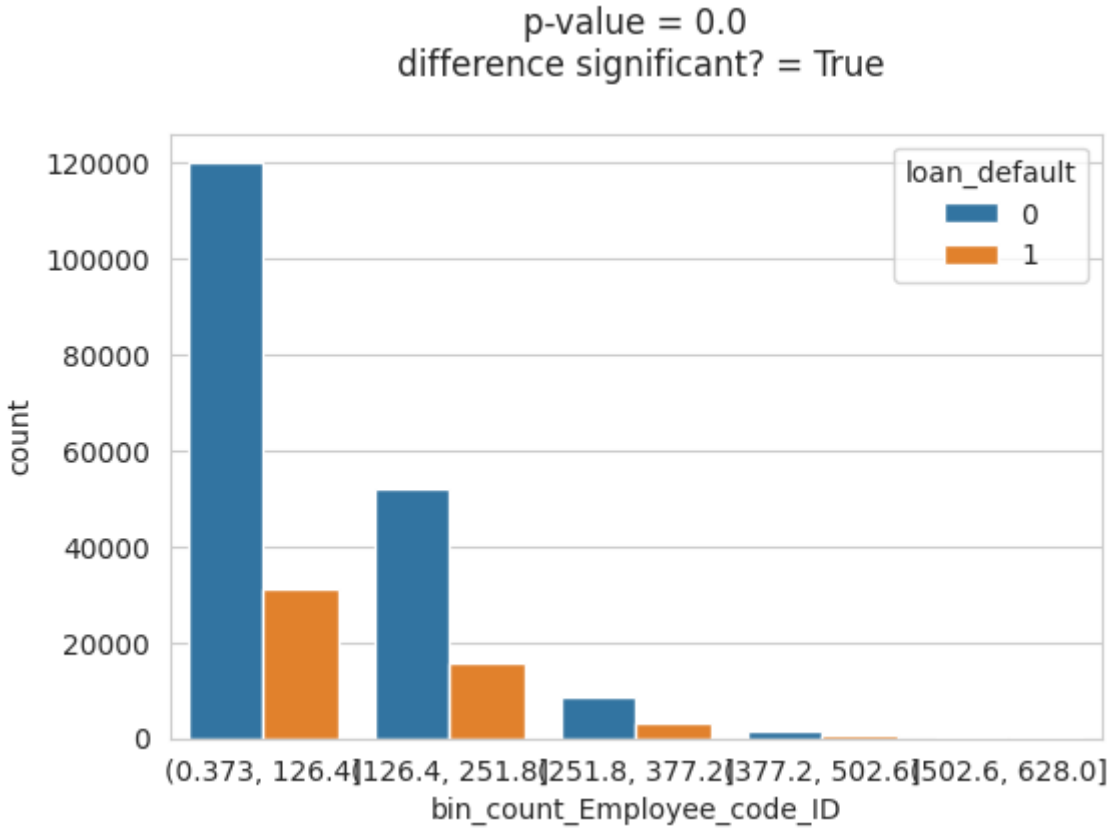
We counted defaulting customer for each supliers and based on ount we have made bins

In [126]:

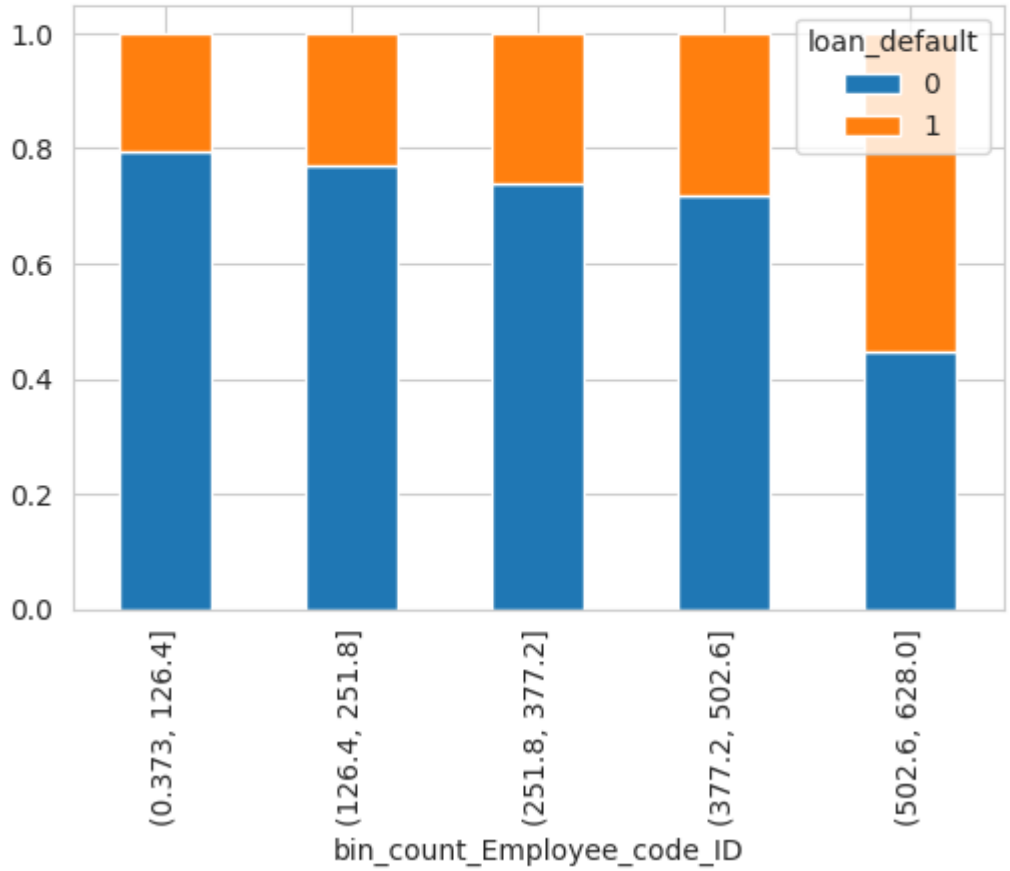
```
df = train[['Employee_code_ID','loan_default']].groupby(['Employee_code_ID'])['loan_

train = train.merge(df, on='Employee_code_ID', how='left')

#binning
train['bin_count_Employee_code_ID'] = pd.cut(train['loan_default_cnt'], bins=5)
```

In [127]:

```
#supplier count frequency vs default rate
biv_cat_cat_analysis(train,'loan_default','bin_count_Employee_code_ID')
```



loan_default	0	1
bin_count_Employee_code_ID		
(0.373, 126.4]	0.794109	0.205891
(126.4, 251.8]	0.770354	0.229646
(251.8, 377.2]	0.739421	0.260579
(377.2, 502.6]	0.716898	0.283102
(502.6, 628.0]	0.445860	0.554140



`Observation`

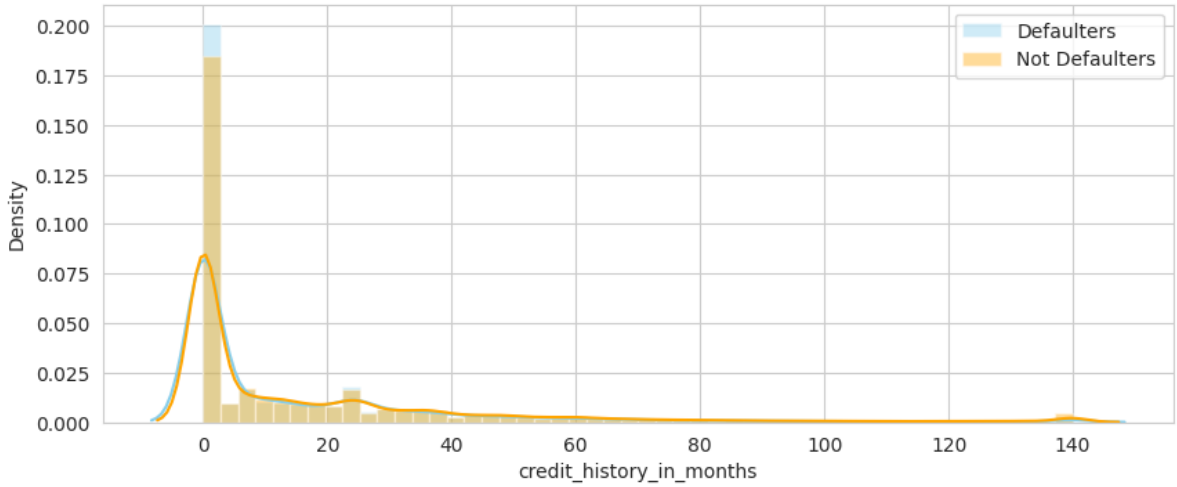
Very few employ handles more then 500 customer but there `default rate` is higher , maybe due to work load as they are handking more customers

In [128]:

```
#patching outliers
ulimit = np.percentile(train.credit_history_in_months.values, 99)
llimit = np.percentile(train.credit_history_in_months.values, 1)
train['credit_history_in_months'][train.credit_history_in_months > ulimit] = ulimit
train['credit_history_in_months'][train.credit_history_in_months < llimit] = llimit
```

In [129]:

```
#alternate way of visualising age_in_years distribution for defaulters & non default
plt.figure(figsize=(10,4),dpi=100)
sns.distplot(train['credit_history_in_months'][train['loan_default'] == 1] , color='
sns.distplot(train['credit_history_in_months'][train['loan_default'] == 0] , color='

plt.legend()
plt.show()
```



Now we can say that least part of data exploration is done by us still there are many factord which can be explored

But now I am convinced to make a model out of this data

In [130]:

```
#loading data
path = '/kaggle/input/bank-loan-data/'
train = pd.read_csv(path + 'train.csv')
test  = pd.read_csv(path + 'test.csv')
```

In [131]:

```
train.head()
```

Out[131]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Curr
0	420825	50578	58400	89.55	67	22807	45	
1	537409	47145	65550	73.23	67	22807	45	
2	417566	53278	61360	89.63	67	22807	45	
3	624493	57513	66113	88.48	67	22807	45	
4	539055	52378	60300	88.39	67	22807	45	

In [132]:

```
test.head()
```

Out[132]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Curr
0	655269	53478	63558	86.54	67	22807	45	
1	723482	55513	63163	89.45	67	22807	45	
2	758529	65282	84320	79.93	78	23135	86	
3	763449	46905	63896	76.58	78	17014	45	
4	708663	51428	63896	86.08	78	17014	45	

In [133]:

```
train.shape,test.shape
```

Out[133]:

```
((233154, 41), (112392, 40))
```

In [134]:

```
#stacking train over test
data = train.append(test,ignore_index=True)
data.shape
```

Out[134]:

```
(345546, 41)
```

In [135]:

```
#null check
data.isnull().sum()
```

Out[135]:

```
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                     11104
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                       112392
dtype: int64
```

In [136]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 345546 entries, 0 to 345545
Data columns (total 41 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   UniqueID                              345546 non-null  int64
 1   disbursed_amount                      345546 non-null  int64
 2   asset_cost                            345546 non-null  int64
 3   ltv                                   345546 non-null  float64
 4   branch_id                             345546 non-null  int64
 5   supplier_id                           345546 non-null  int64
 6   manufacturer_id                       345546 non-null  int64
 7   Current_pincode_ID                    345546 non-null  int64
 8   Date.of.Birth                         345546 non-null  object
 9   Employment.Type                       334442 non-null  object
 10  DisbursalDate                         345546 non-null  object
 11  State_ID                              345546 non-null  int64
 12  Employee_code_ID                      345546 non-null  int64
 13  MobileNo_Avl_Flag                     345546 non-null  int64
 14  Aadhar_flag                           345546 non-null  int64
 15  PAN_flag                              345546 non-null  int64
 16  VoterID_flag                          345546 non-null  int64
 17  Driving_flag                          345546 non-null  int64
 18  Passport_flag                         345546 non-null  int64
 19  PERFORM_CNS.SCORE                     345546 non-null  int64
 20  PERFORM_CNS.SCORE.DESCRIPTION         345546 non-null  object
 21  PRI.NO.OF.ACCTS                       345546 non-null  int64
 22  PRI.ACTIVE.ACCTS                      345546 non-null  int64
 23  PRI.OVERDUE.ACCTS                     345546 non-null  int64
 24  PRI.CURRENT.BALANCE                   345546 non-null  int64
 25  PRI.SANCTIONED.AMOUNT                 345546 non-null  int64
 26  PRI.DISBURSED.AMOUNT                  345546 non-null  int64
 27  SEC.NO.OF.ACCTS                       345546 non-null  int64
 28  SEC.ACTIVE.ACCTS                      345546 non-null  int64
 29  SEC.OVERDUE.ACCTS                     345546 non-null  int64
 30  SEC.CURRENT.BALANCE                   345546 non-null  int64
 31  SEC.SANCTIONED.AMOUNT                 345546 non-null  int64
 32  SEC.DISBURSED.AMOUNT                  345546 non-null  int64
 33  PRIMARY.INSTAL.AMT                    345546 non-null  int64
 34  SEC.INSTAL.AMT                        345546 non-null  int64
 35  NEW.ACCTS.IN.LAST.SIX.MONTHS          345546 non-null  int64
 36  DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS   345546 non-null  int64
 37  AVERAGE.ACCT.AGE                      345546 non-null  object
 38  CREDIT.HISTORY.LENGTH                 345546 non-null  object
 39  NO.OF_INQUIRIES                       345546 non-null  int64
 40  loan_default                          233154 non-null  float64
dtypes: float64(2), int64(33), object(6)
memory usage: 108.1+ MB
```

Credit History & Average Account Age

In [137]:

```
#will apply re on it
print(data['CREDIT.HISTORY.LENGTH'].head(3))
#will apply re on it
print(data['AVERAGE.ACCT.AGE'].head(3))
```

```
0     0yrs 0mon
1    1yrs 11mon
2     0yrs 0mon
Name: CREDIT.HISTORY.LENGTH, dtype: object
0     0yrs 0mon
1    1yrs 11mon
2     0yrs 0mon
Name: AVERAGE.ACCT.AGE, dtype: object
```

In [138]:

```
#function to extract months
def map_to_months(data_rows):
    row_data = list(map(int, re.findall(r'\d+',data_rows)))
    return row_data[0]*12 + row_data[1]

data['credit_history_in_months'] = data['CREDIT.HISTORY.LENGTH'].apply(map_to_months
data['avg_acct_age_in_months']    = data['AVERAGE.ACCT.AGE'].apply(map_to_months)
```

In [139]:

```
#after applying re on it
data['avg_acct_age_in_months'].head(3)
```

Out[139]:

```
0     0
1    23
2     0
Name: avg_acct_age_in_months, dtype: int64
```

AGE

In [140]:

```
#before
print(data['Date.of.Birth'].head(3))
print(data['DisbursalDate'].head(3))
```

```
0    01-01-84
1    31-07-85
2    24-08-85
Name: Date.of.Birth, dtype: object
0    03-08-18
1    26-09-18
2    01-08-18
Name: DisbursalDate, dtype: object
```

In [141]:

```
data['Date.of.Birth'] = pd.to_datetime(data['Date.of.Birth'], format='%d-%m-%y')
data['DisbursalDate'] = pd.to_datetime(data['DisbursalDate'], format='%d-%m-%y')
```

In [142]:

```
future = data['Date.of.Birth'].dt.date > date(year=2019,month=1,day=1)
data.loc[future, 'Date.of.Birth'] -= timedelta(days=365.25*100)

data['age_in_years'] = ((data['DisbursalDate'] - data['Date.of.Birth']).dt.days)/365

data['dob_months'] = data['Date.of.Birth'].dt.month
data['dob_days']   = data['Date.of.Birth'].dt.day
data['dob_weeks']  = data['Date.of.Birth'].dt.week
```

Disbursal Date

In [143]:

```
data['disb_months'] = data['DisbursalDate'].dt.month
data['disb_days']   = data['DisbursalDate'].dt.day
data['disb_weeks']  = data['DisbursalDate'].dt.week
```

In [144]:

```
#after
print(data['Date.of.Birth'].head(3))
print(data['DisbursalDate'].head(3))
```

```
0   1984-01-01
1   1985-07-31
2   1985-08-24
Name: Date.of.Birth, dtype: datetime64[ns]
0   2018-08-03
1   2018-09-26
2   2018-08-01
Name: DisbursalDate, dtype: datetime64[ns]
```

Primary & Secondary Accounts Info

for secondary accounts we have very less information so we are adding primary and secondary accounts

In [145]:

```
data['ACTIVE.ACCTS'] = data['PRI.ACTIVE.ACCTS'] + data['SEC.ACTIVE.ACCTS']
data['CURRENT.BALANCE']  = data['PRI.CURRENT.BALANCE'] + data['SEC.CURRENT.BALANCE']
data['DISBURSED.AMOUNT'] = data['PRI.DISBURSED.AMOUNT'] + data['SEC.DISBURSED.AMOUNT
data['NO.OF.ACCTS']   = data['SEC.NO.OF.ACCTS'] + data['PRI.NO.OF.ACCTS']
data['OVERDUE.ACCTS'] = data['PRI.OVERDUE.ACCTS'] + data['SEC.OVERDUE.ACCTS']
data['SANCTIONED.AMOUNT' ] = data['PRI.SANCTIONED.AMOUNT'] + data['SEC.SANCTIONED.AM
data['INSTAL.AMT'] = data['PRIMARY.INSTAL.AMT'] + data['SEC.INSTAL.AMT']

data['SANCTION_DISBURSED']   = data['SANCTIONED.AMOUNT'] - data['DISBURSED.AMOUNT']
data['NO_DEACTIVE_ACCOUNTS'] = data['NO.OF.ACCTS'] - data['ACTIVE.ACCTS']
```

Loan Information

disbursed_amount ==>

$$da = \frac{ltv}{100} * asset_cost$$

==> Commission involved

$$= [\frac{ltv}{100} * asset_cost] - da$$

In [146]:

```
#Commission involved
data['extra_finance'] = data['asset_cost'] * (data['ltv']/100) - data['disbursed_amc
```

Score Description

In [147]:

```
data['PERFORM_CNS.SCORE.DESCRIPTION'].unique()
```

Out[147]:

```
array(['No Bureau History Available', 'I-Medium Risk', 'L-Very High Ri
sk',
       'A-Very Low Risk',
       'Not Scored: Not Enough Info available on the customer',
       'D-Very Low Risk', 'M-Very High Risk', 'B-Very Low Risk',
       'C-Very Low Risk', 'E-Low Risk', 'H-Medium Risk', 'F-Low Risk',
       'K-High Risk',
       'Not Scored: No Activity seen on the customer (Inactive)',
       'Not Scored: Sufficient History Not Available',
       'Not Scored: No Updates available in last 36 months', 'G-Low Ri
sk',
       'J-High Risk', 'Not Scored: Only a Guarantor',
       'Not Scored: More than 50 active Accounts found'], dtype=objec
t)
```

In [148]:

```
data['PERFORM_CNS.SCORE.DESCRIPTION'].replace({'C-Very Low Risk':'Very Low Risk','A-
                                               'D-Very Low Risk':'Very Low Risk','B-Ve
                                               'M-Very High Risk':'Very High Risk','L-
                                               'F-Low Risk':'Low Risk','E-Low Risk':'I
                                               'I-Medium Risk':'Medium Risk','J-High F
                                               'K-High Risk':'High Risk'},inplace=True

data['Not_Scored'] = np.where(data['PERFORM_CNS.SCORE.DESCRIPTION'].str.contains('No
data['Very_Low']   = np.where(data['PERFORM_CNS.SCORE.DESCRIPTION'].str.contains('Ve
data['Very_High']  = np.where(data['PERFORM_CNS.SCORE.DESCRIPTION'].str.contains('Ve
data['No_History'] = np.where(data['PERFORM_CNS.SCORE.DESCRIPTION'].str.contains('No
```

employment type

In [149]:

```
data['Employment.Type'].unique()
```

Out[149]:

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

In [150]:

```
data['Employment.Type'] = data['Employment.Type'].apply(lambda x: 0 if x == "Salarie
```

In [151]:

```
data['Employment.Type'].unique()
```

Out[151]:

```
array([0, 1])
```

Identity Proof Flags

I have added all of them collectvly they will make more sense

In [152]:

```
for i in ['Aadhar_flag','Driving_flag','PAN_flag','Passport_flag','VoterID_flag']:
    data[i] = data[i].astype(np.object)

data['Total_Flag'] = data['Driving_flag'] + data['Aadhar_flag'] + data['PAN_flag'] +
```

NOTE ==> for tree based model we can keep all these id's but for Logistic Regression it is hard as loat of features will effect it's performance

In [153]:

```
#droping some of id's and some of other features which have be enginered
drop_cols = ['Date.of.Birth','UniqueID','MobileNo_Avl_Flag',
             'PERFORM_CNS.SCORE.DESCRIPTION','AVERAGE.ACCT.AGE','CREDIT.HISTORY.LENG
             'Aadhar_flag', 'Driving_flag', 'PAN_flag', 'Passport_flag', 'VoterID_fl
             'Employee_code_ID','branch_id','State_ID','manufacturer_id','supplier_i

data.drop(drop_cols,axis=1,inplace=True)
```

Re-Split

No again splitting train and test data from main data
Logick is where we dont have loan_default information that is test data

In [154]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 345546 entries, 0 to 345545
Data columns (total 49 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   disbursed_amount                      345546 non-null  int64
 1   asset_cost                            345546 non-null  int64
 2   ltv                                   345546 non-null  float64
 3   Current_pincode_ID                    345546 non-null  int64
 4   Employment.Type                       345546 non-null  int64
 5   DisbursalDate                         345546 non-null  datetime64
[ns]
 6   PERFORM_CNS.SCORE                     345546 non-null  int64
 7   PRI.NO.OF.ACCTS                       345546 non-null  int64
 8   PRI.ACTIVE.ACCTS                      345546 non-null  int64
 9   PRI.OVERDUE.ACCTS                     345546 non-null  int64
 10  PRI.CURRENT.BALANCE                   345546 non-null  int64
 11  PRI.SANCTIONED.AMOUNT                 345546 non-null  int64
 12  PRI.DISBURSED.AMOUNT                  345546 non-null  int64
 13  SEC.NO.OF.ACCTS                       345546 non-null  int64
 14  SEC.ACTIVE.ACCTS                      345546 non-null  int64
 15  SEC.OVERDUE.ACCTS                     345546 non-null  int64
 16  SEC.CURRENT.BALANCE                   345546 non-null  int64
 17  SEC.SANCTIONED.AMOUNT                 345546 non-null  int64
 18  SEC.DISBURSED.AMOUNT                  345546 non-null  int64
 19  PRIMARY.INSTAL.AMT                    345546 non-null  int64
 20  SEC.INSTAL.AMT                        345546 non-null  int64
 21  NEW.ACCTS.IN.LAST.SIX.MONTHS          345546 non-null  int64
 22  DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS   345546 non-null  int64
 23  NO.OF_INQUIRIES                       345546 non-null  int64
 24  loan_default                          233154 non-null  float64
 25  credit_history_in_months              345546 non-null  int64
 26  avg_acct_age_in_months                345546 non-null  int64
 27  age_in_years                          345546 non-null  float64
 28  dob_months                            345546 non-null  int64
 29  dob_days                              345546 non-null  int64
 30  dob_weeks                             345546 non-null  int64
 31  disb_months                           345546 non-null  int64
 32  disb_days                             345546 non-null  int64
 33  disb_weeks                            345546 non-null  int64
 34  ACTIVE.ACCTS                          345546 non-null  int64
 35  CURRENT.BALANCE                       345546 non-null  int64
 36  DISBURSED.AMOUNT                      345546 non-null  int64
 37  NO.OF.ACCTS                           345546 non-null  int64
 38  OVERDUE.ACCTS                         345546 non-null  int64
 39  SANCTIONED.AMOUNT                     345546 non-null  int64
 40  INSTAL.AMT                            345546 non-null  int64
 41  SANCTION_DISBURSED                    345546 non-null  int64
 42  NO_DEACTIVE_ACCOUNTS                  345546 non-null  int64
 43  extra_finance                         345546 non-null  float64
 44  Not_Scored                            345546 non-null  int64
 45  Very_Low                              345546 non-null  int64
 46  Very_High                             345546 non-null  int64
 47  No_History                            345546 non-null  int64
 48  Total_Flag                            345546 non-null  object
dtypes: datetime64[ns](1), float64(4), int64(43), object(1)
memory usage: 129.2+ MB
```

In [155]:

```
data['Employment.Type']
```

Out[155]:

```
0          0
1          1
2          1
3          1
4          1
          ..
345541     1
345542     1
345543     1
345544     1
345545     1
Name: Employment.Type, Length: 345546, dtype: int64
```

In [156]:

```
train = data[data['loan_default'].isnull() != True]
test  =  data[data['loan_default'].isnull() == True]
```

In [157]:

```
train.DisbursalDate.describe()
```

Out[157]:

```
count                  233154
unique                     84
top       2018-10-31 00:00:00
freq                     8826
first     2018-08-01 00:00:00
last      2018-10-31 00:00:00
Name: DisbursalDate, dtype: object
```

Observation

All dates are in past for train data

In [158]:

```
test.DisbursalDate.describe()
```

Out[158]:

```
count                  112392
unique                     27
top       2018-11-15 00:00:00
freq                     7803
first     2018-11-03 00:00:00
last      2018-11-30 00:00:00
Name: DisbursalDate, dtype: object
```

Observation

All dates are in future for test data in compare to train data

We have to `make 1 validation set` to check our model performance, but question is:

- As data for test is in future in respect to train data, **`we can use randome split or not???`**
  - We cannot use `randome_split` as that will not `preserve structure` of data

One way to overcome to it is

In [159]:

```
import datetime
```

In [160]:

```
x_train_not_final = train[train.DisbursalDate < datetime.datetime(2018,10,1)]
x_valid_not_final = train[train.DisbursalDate >= datetime.datetime(2018,10,1)] #last
```

based on `DisbursalDate` we splited out data set into train and validation, now droping it as we have extracted features out of it before

In [161]:

```
x_train = x_train_not_final.drop(['loan_default','DisbursalDate'],axis=1)
y_train = x_train_not_final.loan_default

x_valid = x_valid_not_final.drop(['loan_default','DisbursalDate'],axis=1)
y_valid = x_valid_not_final.loan_default
```

In [162]:

```
x_train.shape,y_train.shape,x_valid.shape,y_valid.shape
```

Out[162]:

```
((134790, 47), (134790,), (98364, 47), (98364,))
```

In [163]:

```
from sklearn.linear_model import LogisticRegression as lr
from sklearn.tree import DecisionTreeClassifier as dtc

from sklearn.metrics import confusion_matrix,accuracy_score,recall_score,roc_auc_scc
```

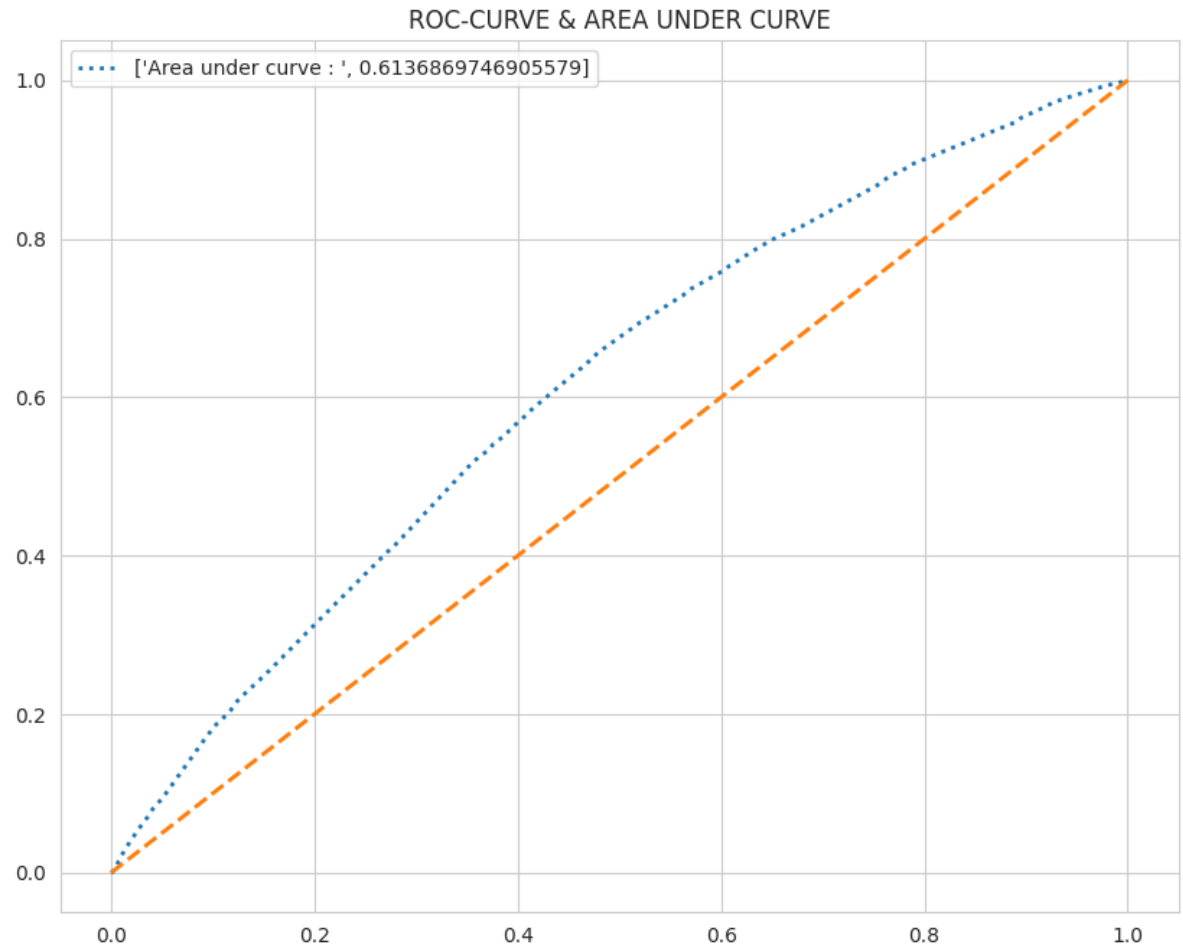
In [164]:

```
#modeling function
def model(algorithm, dtrain_X, dtrain_Y, dtest_X, dtest_Y, cols=None):
    algorithm.fit(dtrain_X[cols],dtrain_Y)
    predictions = algorithm.predict(dtest_X[cols])
    print(algorithm)

    print(f"Accuracy score : {accuracy_score(predictions,dtest_Y)}")
    print(f"Recall score   : {recall_score(predictions,dtest_Y)}")
    print(f"Classification report :\n{classification_report(predictions,dtest_Y)}")

    fig = plt.figure(figsize=(10,8))
    ax  = fig.add_subplot(111)
    prediction_probabilities = algorithm.predict_proba(dtest_X[cols])[:,1]
    fpr , tpr , thresholds   = roc_curve(dtest_Y,prediction_probabilities)
    ax.plot(fpr,tpr,label    = ["Area under curve : ",auc(fpr,tpr)],linewidth=2,line
    ax.plot([0,1],[0,1],linewidth=2,linestyle="dashed")
    plt.legend(loc="best")
    plt.title("ROC-CURVE & AREA UNDER CURVE")
```

In [165]:

```
dtc = dtc(max_depth=7)
#using modelling function
model(dtc,x_train,y_train,x_valid,y_valid,x_train.columns)
```

```
DecisionTreeClassifier(max_depth=7)
Accuracy score : 0.7639380261071124
Recall score   : 0.20394736842105263
Classification report :
              precision    recall  f1-score   support

         0.0       1.00      0.76      0.87     98212
         1.0       0.00      0.20      0.00       152

    accuracy                           0.76     98364
   macro avg       0.50      0.48      0.43     98364
weighted avg       1.00      0.76      0.86     98364
```



In [166]:

```
pred = dtc.predict_proba(test.drop(['DisbursalDate','loan_default'], axis =1))
pred
```

Out[166]:

```
array([[0.75066013, 0.24933987],
       [0.7805326 , 0.2194674 ],
       [0.71208142, 0.28791858],
       ...,
       [0.87126237, 0.12873763],
       [0.72938689, 0.27061311],
       [0.88752247, 0.11247753]])
```

- First column have Probability of 0
- Second column have Probability of 1 ==> we want this

In [167]:

```
pred[:,-1]
```

Out[167]:

```
array([0.24933987, 0.2194674 , 0.28791858, ..., 0.12873763, 0.2706131
1,
       0.11247753])
```

In [168]:

```
#reading sample sub files which have randome loan_default values
sample_submission = pd.read_csv(path + 'sample.csv')

#replacing loan_default with predictions
sample_submission['loan_default'] = pred[:,-1]

#writing prediciton in new file
sample_submission.to_csv('pred_dtc_new.csv', index=False)
```

In [169]:

```
sample_submission.head()
```

Out[169]:

	UniqueID	loan_default
0	655269	0.249340
1	723482	0.219467
2	758529	0.287919
3	763449	0.194112
4	708663	0.287919

Now we can plot this dt ans see if it alined with out hypothesis