In [ ]:
```
#Installing important necessary packages

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

random_state=42
```

In [ ]:
```
df = pd.read_excel("dataloandefault.xlsx")
print(df.shape)
df.head()
```

In [ ]:
```
# Let's get an idea about all the features available in dataset
df.info()
```

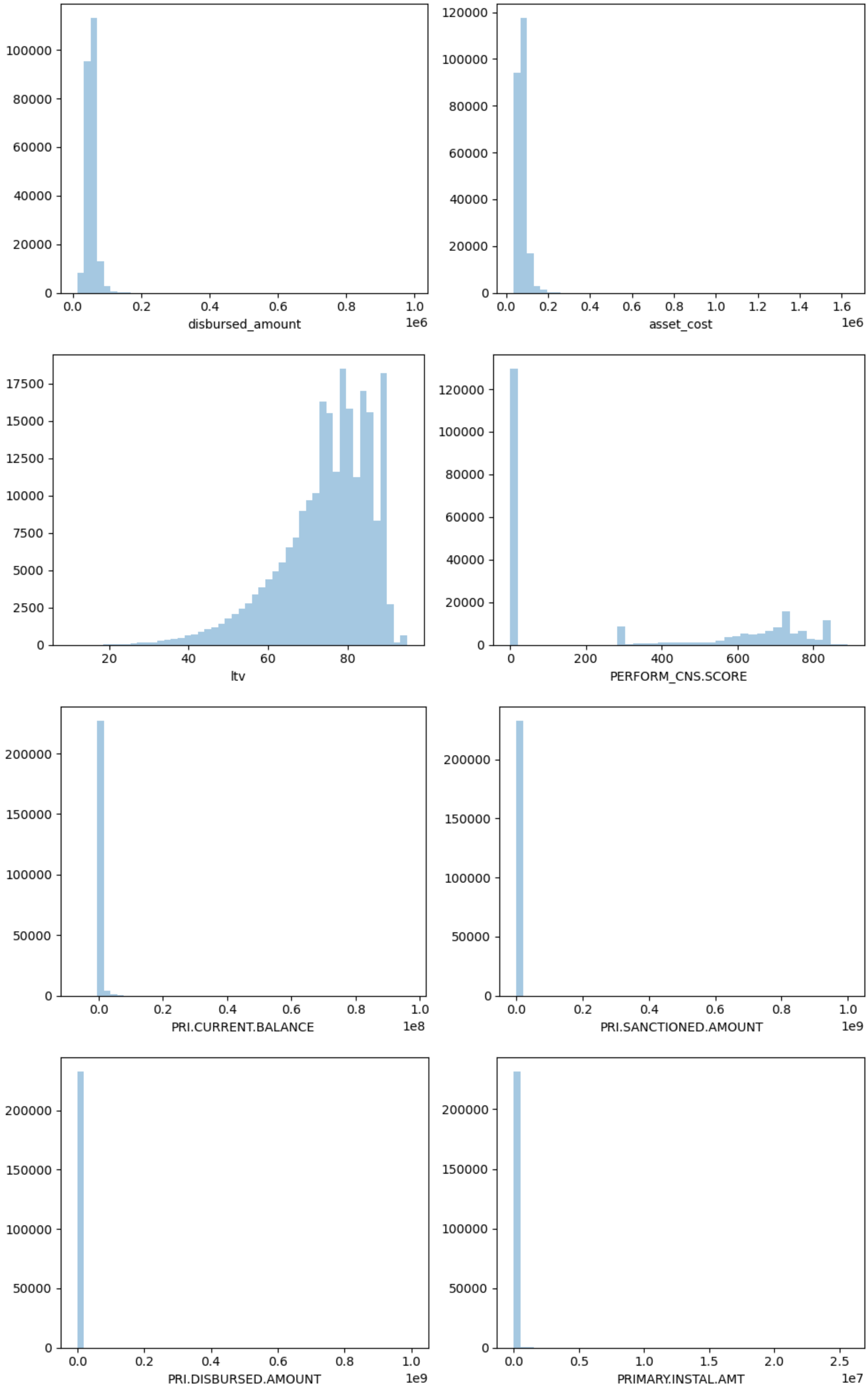
In [ ]:
```
# Inspect the mean and standard deviation to see the scale of each features
df.describe()
```

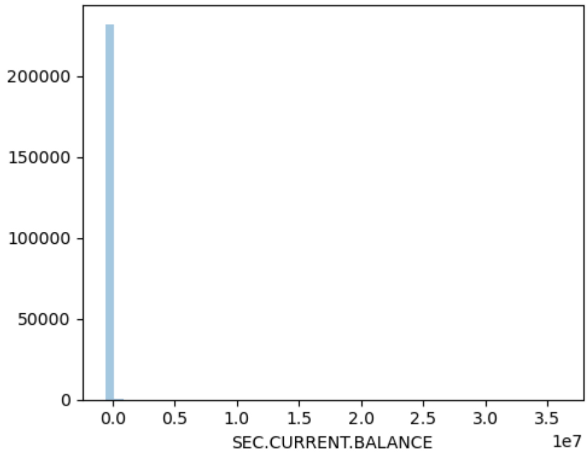
In [ ]:
```
# Now check if there is null value in the data!
df.isnull().sum()
```

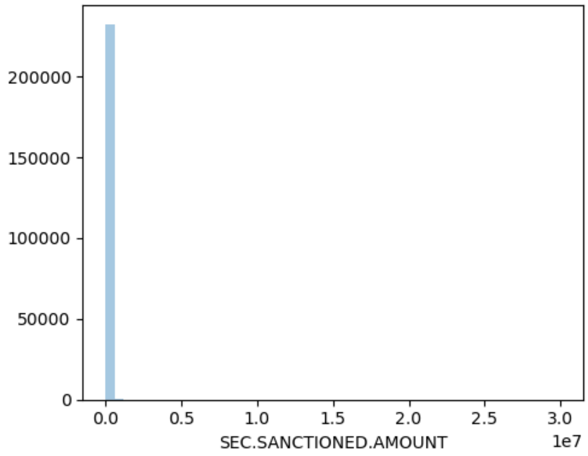
In [ ]:
```
# List of columns with numerical features
numerical_feature_columns = list(df._get_numeric_data().columns)
numerical_feature_columns
```

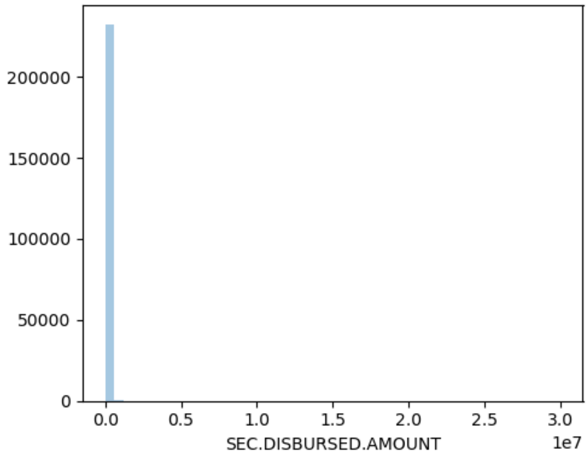
In [ ]:
```
# List of columns with categorical features
categorical_feature_columns = list(set(df.columns) - set(numerical_feature_columns)
categorical_feature_columns
```

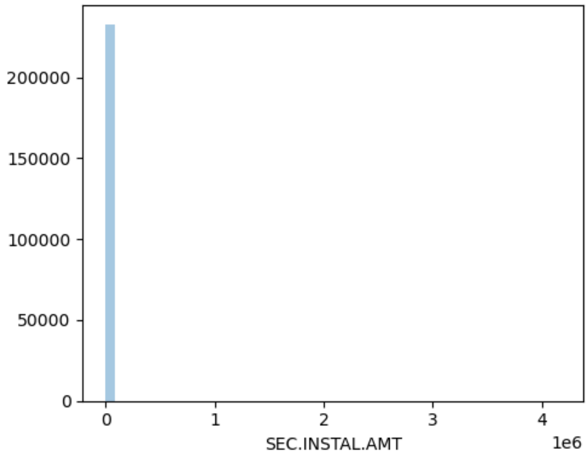
In [8]:
```
num_columns = ['disbursed_amount','asset_cost','ltv','PERFORM_CNS.SCORE','PRI.CURRE
            'PRI.DISBURSED.AMOUNT','PRIMARY.INSTAL.AMT','SEC.CURRENT.BALANCE','SEC.
            'SEC.INSTAL.AMT']

for i in range(0, len(num_columns), 2):
    plt.figure(figsize=(10,4))
    plt.subplot(121)
    sns.distplot(df[num_columns[i]], kde=False)
    plt.subplot(122)
    sns.distplot(df[num_columns[i+1]], kde=False)
    plt.tight_layout()
    plt.show()
```


disbursed_amount
1e6
asset_cost
1e6
ltv
PERFORM_CNS.SCORE
PRI.CURRENT.BALANCE
1e8
PRI.SANCTIONED.AMOUNT
1e9
PRI.DISBURSED.AMOUNT
1e9
PRIMARY.INSTAL.AMT
1e7









In [9]:
```
df[categorical_feature_columns].head()
```

Out[9]:

	Date.of.Birth	Employment.Type	PERFORM_CNS.SCORE.DESCRIPTION	AVERAGE.ACCT.AGE	CREDI
0	1984-01-01	Salaried	No Bureau History Available	0yrs 0mon	
1	1985-08-24	Self employed	No Bureau History Available	0yrs 0mon	
2	1977-12-09	Self employed	No Bureau History Available	0yrs 0mon	
3	1988-06-01	Salaried	No Bureau History Available	0yrs 0mon	
4	1994-07-14	Self employed	No Bureau History Available	0yrs 0mon	

In [10]:
```
#Two features AVERAGE.ACCT.AGE and CREDIT.HISTORY.LENGTH need to convert in terms c

df['AVERAGE.ACCT.AGE'] = df['AVERAGE.ACCT.AGE'].str.replace('yrs ','.',regex=False)
df['AVERAGE.ACCT.AGE'] = df['AVERAGE.ACCT.AGE'].str.replace('mon','',regex=False).a
df['CREDIT.HISTORY.LENGTH'] = df['CREDIT.HISTORY.LENGTH'].str.replace('yrs ','.',re
df['CREDIT.HISTORY.LENGTH'] = df['CREDIT.HISTORY.LENGTH'].str.replace('mon','',rege
df[categorical_feature_columns].head()
```

Out[10]:

	Date.of.Birth	Employment.Type	PERFORM_CNS.SCORE.DESCRIPTION	AVERAGE.ACCT.AGE	CREDI
0	1984-01-01	Salaried	No Bureau History Available	0.0	
1	1985-08-24	Self employed	No Bureau History Available	0.0	
2	1977-12-09	Self employed	No Bureau History Available	0.0	
3	1988-06-01	Salaried	No Bureau History Available	0.0	
4	1994-07-14	Self employed	No Bureau History Available	0.0	

In [11]:

```
#Now let's examine the Employment.Type feature which has missing values.
df['Employment.Type'].isnull().sum()
```

Out[11]:

```
7661
```

In [12]:

```
7661
# Calculate missing percent value from whole dataset
total_null = df.isnull().sum()
percent_null = (total_null/(df.isnull().count())) * 100
missing_data = pd.concat([total_null,percent_null], keys=['Total','Percent'],axis=1
print(missing_data)
```

```
                                     Total   Percent
UniqueID                                 0  0.000000
disbursed_amount                         0  0.000000
asset_cost                               0  0.000000
ltv                                      0  0.000000
branch_id                                0  0.000000
supplier_id                              0  0.000000
manufacturer_id                          0  0.000000
Current_pincode_ID                       0  0.000000
Date.of.Birth                            0  0.000000
Employment.Type                       7661  3.285811
DisbursalDate                            0  0.000000
State_ID                                 0  0.000000
Employee_code_ID                         0  0.000000
MobileNo_Avl_Flag                        0  0.000000
Aadhar_flag                              0  0.000000
PAN_flag                                 0  0.000000
VoterID_flag                             0  0.000000
Driving_flag                             0  0.000000
Passport_flag                            0  0.000000
PERFORM_CNS.SCORE                        0  0.000000
PERFORM_CNS.SCORE.DESCRIPTION            0  0.000000
PRI.NO.OF.ACCTS                          0  0.000000
PRI.ACTIVE.ACCTS                         0  0.000000
PRI.OVERDUE.ACCTS                        0  0.000000
PRI.CURRENT.BALANCE                      0  0.000000
PRI.SANCTIONED.AMOUNT                    0  0.000000
PRI.DISBURSED.AMOUNT                     0  0.000000
SEC.NO.OF.ACCTS                          0  0.000000
SEC.ACTIVE.ACCTS                         0  0.000000
SEC.OVERDUE.ACCTS                        0  0.000000
SEC.CURRENT.BALANCE                      0  0.000000
SEC.SANCTIONED.AMOUNT                    0  0.000000
SEC.DISBURSED.AMOUNT                     0  0.000000
PRIMARY.INSTAL.AMT                       0  0.000000
SEC.INSTAL.AMT                           0  0.000000
NEW.ACCTS.IN.LAST.SIX.MONTHS             0  0.000000
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS      0  0.000000
AVERAGE.ACCT.AGE                         0  0.000000
CREDIT.HISTORY.LENGTH                    0  0.000000
NO.OF_INQUIRIES                          0  0.000000
loan_default                             0  0.000000
```

In [13]:

```
df.dropna(inplace=True)
total_null_1 = df.isnull().sum()
percent_null_1 = (total_null_1/(df.isnull().count())) * 100
missing_data_1 = pd.concat([total_null_1,percent_null_1], keys=['Total','Percent'],
print(missing_data_1)
```

```
                                     Total  Percent
UniqueID                                 0      0.0
disbursed_amount                         0      0.0
asset_cost                               0      0.0
ltv                                      0      0.0
branch_id                                0      0.0
supplier_id                              0      0.0
manufacturer_id                          0      0.0
Current_pincode_ID                       0      0.0
Date.of.Birth                            0      0.0
Employment.Type                          0      0.0
DisbursalDate                            0      0.0
State_ID                                 0      0.0
Employee_code_ID                         0      0.0
MobileNo_Avl_Flag                        0      0.0
Aadhar_flag                              0      0.0
PAN_flag                                 0      0.0
VoterID_flag                             0      0.0
Driving_flag                             0      0.0
Passport_flag                            0      0.0
PERFORM_CNS.SCORE                        0      0.0
PERFORM_CNS.SCORE.DESCRIPTION            0      0.0
PRI.NO.OF.ACCTS                          0      0.0
PRI.ACTIVE.ACCTS                         0      0.0
PRI.OVERDUE.ACCTS                        0      0.0
PRI.CURRENT.BALANCE                      0      0.0
PRI.SANCTIONED.AMOUNT                    0      0.0
PRI.DISBURSED.AMOUNT                     0      0.0
SEC.NO.OF.ACCTS                          0      0.0
SEC.ACTIVE.ACCTS                         0      0.0
SEC.OVERDUE.ACCTS                        0      0.0
SEC.CURRENT.BALANCE                      0      0.0
SEC.SANCTIONED.AMOUNT                    0      0.0
SEC.DISBURSED.AMOUNT                     0      0.0
PRIMARY.INSTAL.AMT                       0      0.0
SEC.INSTAL.AMT                           0      0.0
NEW.ACCTS.IN.LAST.SIX.MONTHS             0      0.0
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS      0      0.0
AVERAGE.ACCT.AGE                         0      0.0
CREDIT.HISTORY.LENGTH                    0      0.0
NO.OF_INQUIRIES                          0      0.0
loan_default                             0      0.0
```

In [14]:
```
# Count the each category values from feature
df['Employment.Type'].value_counts()
```

Out[14]:
```
Self employed    127635
Salaried          97858
Name: Employment.Type, dtype: int64
```

In [15]:
```
# Encode the values in terms of 0 and 1
df['Employment.Type'].replace({'Salaried': 0, 'Self employed': 1}, inplace=True)
```

In [16]:
```
# Dropping unecessary features
df.drop(['Date.of.Birth','DisbursalDate','PERFORM_CNS.SCORE.DESCRIPTION'], axis = 1
```

In [17]:
```
# Now let's check if null values present in data
df.isnull().sum().sum()
```

Out[17]:
```
0
```

In [18]:
```
# Size of the data
df.shape
```

Out[18]:
```
(225493, 38)
```

In [19]:
```
# Identify unique values in each features
df.nunique()
```

Out[19]:
```
UniqueID                               225493
disbursed_amount                        24228
asset_cost                              45415
ltv                                      6541
branch_id                                  82
supplier_id                              2945
manufacturer_id                            11
Current_pincode_ID                       6659
Employment.Type                             2
State_ID                                   22
Employee_code_ID                         3269
MobileNo_Avl_Flag                           1
Aadhar_flag                                 2
PAN_flag                                    2
VoterID_flag                                2
Driving_flag                                2
Passport_flag                               2
PERFORM_CNS.SCORE                         573
PRI.NO.OF.ACCTS                           107
PRI.ACTIVE.ACCTS                           40
PRI.OVERDUE.ACCTS                          22
PRI.CURRENT.BALANCE                     70044
PRI.SANCTIONED.AMOUNT                   43743
PRI.DISBURSED.AMOUNT                    47206
SEC.NO.OF.ACCTS                            37
SEC.ACTIVE.ACCTS                           23
SEC.OVERDUE.ACCTS                           9
SEC.CURRENT.BALANCE                      3197
SEC.SANCTIONED.AMOUNT                    2195
SEC.DISBURSED.AMOUNT                     2519
PRIMARY.INSTAL.AMT                      27608
SEC.INSTAL.AMT                           1890
NEW.ACCTS.IN.LAST.SIX.MONTHS               26
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS        14
AVERAGE.ACCT.AGE                          178
CREDIT.HISTORY.LENGTH                     269
NO.OF_INQUIRIES                            25
loan_default                                2
dtype: int64
```

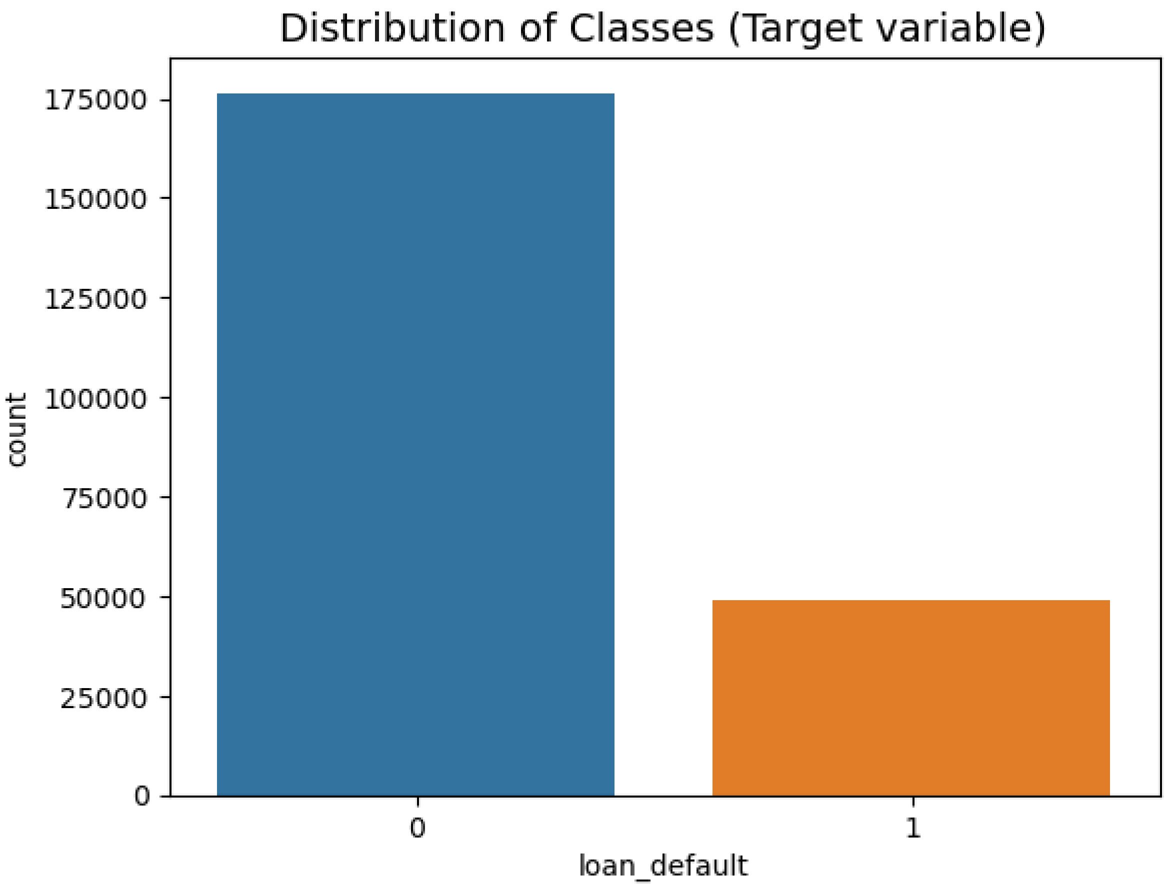
In [20]:
```
corr_max = df.corr()  #create correlation matrix
threshold = 0.5
corr_var_list = []
cols = df.columns.tolist()

for i in range(1, len(cols)):
    for j in range(i):
        if((abs(corr_max.iloc[i,j]) > threshold) & (abs(corr_max.iloc[i,j]) < 1)):
            corr_var_list.append([corr_max.iloc[i,j], i, j])

# Sort the list showing higher ones first
sort_corr_list = sorted(corr_var_list, key=lambda x:abs(x[0]))

#Print correlations and column names
for corr_value, i, j in sort_corr_list:
    print (f"{cols[i]} and {cols[j]} = {round(corr_value, 2)}")
```

```
SEC.OVERDUE.ACCTS and SEC.NO.OF.ACCTS = 0.51
SEC.OVERDUE.ACCTS and SEC.ACTIVE.ACCTS = 0.53
NEW.ACCTS.IN.LAST.SIX.MONTHS and PRI.NO.OF.ACCTS = 0.54
NEW.ACCTS.IN.LAST.SIX.MONTHS and PRI.ACTIVE.ACCTS = 0.7
asset_cost and disbursed_amount = 0.75
PRI.ACTIVE.ACCTS and PRI.NO.OF.ACCTS = 0.75
CREDIT.HISTORY.LENGTH and AVERAGE.ACCT.AGE = 0.82
SEC.ACTIVE.ACCTS and SEC.NO.OF.ACCTS = 0.83
VoterID_flag and Aadhar_flag = -0.87
SEC.SANCTIONED.AMOUNT and SEC.CURRENT.BALANCE = 0.93
SEC.DISBURSED.AMOUNT and SEC.CURRENT.BALANCE = 0.93
PRI.DISBURSED.AMOUNT and PRI.SANCTIONED.AMOUNT = 1.0
SEC.DISBURSED.AMOUNT and SEC.SANCTIONED.AMOUNT = 1.0
```

In [23]:

```
#Plotting distribution of classes of target variable:

print('Distribution of the loan_default in the dataset')
print(df['loan_default'].value_counts()/len(df))

sns.countplot(x = 'loan_default', data=df)
plt.title('Distribution of Classes (Target variable)', fontsize=14)
plt.show()
```

```
Distribution of the loan_default in the dataset
0    0.782845
1    0.217155
Name: loan_default, dtype: float64
```



In [22]:

```
df['loan_default'].value_counts()
```

Out[22]:

```
0    176526
1     48967
Name: loan_default, dtype: int64
```

In [24]:

```
# Over sampling to resolve imbalance
df = df.sample(frac=1)
```

```
loan_default_1 = df.loc[df['loan_default'] == 1]
loan_default_0 = df.loc[df['loan_default'] == 0]

normal_distributed_df = pd.concat([loan_default_1, loan_default_1, loan_default_1,

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)
new_df.head()
```

Out[24]:

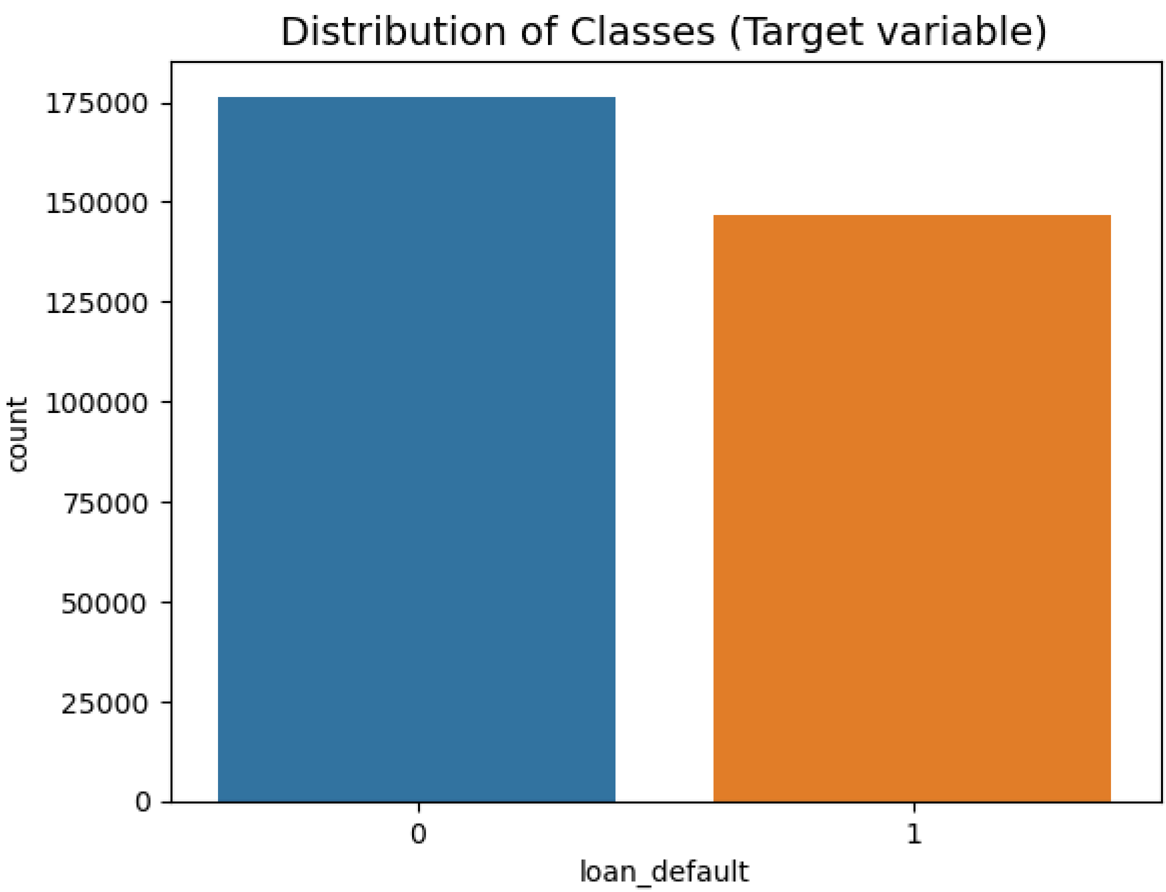
	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id
178585	575628	53079	73765	75.92	5	14145	86
157531	651111	59259	75001	80.00	146	16445	86
143521	452925	48898	59936	88.43	3	14573	45
156715	431930	43394	61430	71.63	136	15705	45
44219	458852	59868	72621	84.99	13	18486	86

5 rows × 38 columns

In [26]:

```
print('Distribution of the loan_default in the dataset')
print(new_df['loan_default'].value_counts()/len(new_df))

sns.countplot(x='loan_default', data=new_df)
plt.title('Distribution of Classes (Target variable)', fontsize=14)
plt.show()
```

```
Distribution of the loan_default in the dataset
0    0.545799
1    0.454201
Name: loan_default, dtype: float64
```



In [27]:
```
# Size of dataset after over sampling
new_df.shape
```

Out[27]:
```
(323427, 38)
```

In [28]:
```
#Seperate features and target variable

X = new_df.drop('loan_default', axis=1)
y = new_df['loan_default'].copy()
```

In [30]:
```
#Split train and test data with 70:30 ratio

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_s
```

In [31]:
```
#Build and evaluate models
#Define evaluation function which calculates following metrics:

#Confusion matrix
#Accuracy score
#Precision
#Recall
#F1 score
#ROC AUC score

def evaluate_model(y_test, y_pred):
    print("Confusion Matrix: \n", metrics.confusion_matrix(y_test, y_pred))
    print("Accuracy: ",metrics.accuracy_score(y_test, y_pred))
    print("Precision: ",metrics.precision_score(y_test, y_pred))
    print("Recall: ",metrics.recall_score(y_test, y_pred))
    print("f1 score: ",metrics.f1_score(y_test, y_pred))
    print("roc_auc_score: ",metrics.roc_auc_score(y_test, y_pred))
```

In [32]:
```
# Scaling training and testing data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [34]:
```
# 1 Logistic Regression

# Find best parameters using grid search
params = {'C':[0.1, 0.5, 1, 5]}

lr = LogisticRegression()
grid = GridSearchCV(estimator=lr, param_grid=params)
grid.fit(X_train, y_train)
y_pred = grid.predict(X_test)
evaluate_model(y_test, y_pred)
```

```
Confusion Matrix:
 [[38047 14960]
 [23994 20028]]
Accuracy:  0.5985323975306351
Precision:  0.5724248313707557
Recall:  0.45495434101131255
f1 score:  0.5069738007847109
roc_auc_score:  0.5863637326578249
```

In [35]:
```
# 2. Decision Trees
params = {'criterion':['gini','entropy'], 'max_depth': [2,3,4,5]}
dt = DecisionTreeClassifier()
dt_clf = GridSearchCV(dt, params)
dt_clf.fit(X_train, y_train)
```

```
y_pred = dt_clf.predict(X_test)
evaluate_model(y_test, y_pred)
```

```
Confusion Matrix:
 [[33728 19279]
 [20348 23674]]
Accuracy:  0.5915963268713478
Precision:  0.5511605708565176
Recall:  0.5377765662623234
f1 score:  0.5443863179074447
roc_auc_score:  0.5870349430062725
```

In [36]:

```
# 3. Random Forest
rf = RandomForestClassifier(n_estimators=250, random_state=random_state)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
evaluate_model(y_test, y_pred)
```

```
Confusion Matrix:
 [[47949  5058]
 [ 3159 40863]]
Accuracy:  0.9153139782951488
Precision:  0.8898543150192723
Recall:  0.9282404252419245
f1 score:  0.908642140022014
roc_auc_score:  0.9164095328994161
```

In [ ]:

```
## Conclusion
# In this classification problem, it is clear the Random Forest Classifier outperfo
#As it has highest accuracy of 91.5% and highest precision of 89%
```