```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing all the libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

To read/write with excel files, we need to install Python library called openpyxl

```
# !pip install openpyxl
```

Data Exploration

Loading the excel file i.e data

```
df = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/Loan_Prediction_Dataset/data.xlsx')
```

```
# get an overview of the data
df.head()
```

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

5 rows × 41 columns

```
df.tail()
```

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufa
233149	561031	57759	76350	77.28	5	22289	
233150	649600	55009	71200	78.72	138	17408	
233151	603445	58513	68000	88.24	135	23313	
233152	442948	22824	40458	61.79	160	16212	
233153	545300	35299	72698	52.27	3	14573	

5 rows × 41 columns

To check the no of rows and columns in the dataframe we use shape method

```
df.shape
```

```
(233154, 41)
```

To check the summary or information of dataframe such as index, columns, non-null counts, memory usage

```
df.info()
```

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

To find the statistical summary data, we use describe() method

```
df.describe()
```

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	
count	233154.000000	233154.000000	2.331540e+05	233154.000000	233154.000000	23
mean	535917.573376	54356.993528	7.586507e+04	74.746530	72.936094	1
std	68315.693711	12971.314171	1.894478e+04	11.456636	69.834995	
min	417428.000000	13320.000000	3.700000e+04	10.030000	1.000000	1
25%	476786.250000	47145.000000	6.571700e+04	68.880000	14.000000	1
75%	595039.750000	60413.000000	7.920175e+04	83.670000	130.000000	2

To check null values if any and sum it

```
df.isnull().sum()
```

```
UniqueID                                 0
disbursed_amount                         0
asset_cost                               0
ltv                                      0
branch_id                                0
supplier_id                              0
manufacturer_id                          0
Current_pincode_ID                       0
Date.of.Birth                            0
Employment.Type                       7661
DisbursalDate                            0
State_ID                                 0
Employee_code_ID                         0
MobileNo_Avl_Flag                        0
Aadhar_flag                              0
PAN_flag                                 0
VoterID_flag                             0
Driving_flag                             0
Passport_flag                            0
PERFORM_CNS.SCORE                        0
PERFORM_CNS.SCORE.DESCRIPTION            0
PRI.NO.OF.ACCTS                          0
PRI.ACTIVE.ACCTS                         0
PRI.OVERDUE.ACCTS                        0
PRI.CURRENT.BALANCE                      0
PRI.SANCTIONED.AMOUNT                    0
PRI.DISBURSED.AMOUNT                     0
SEC.NO.OF.ACCTS                          0
SEC.ACTIVE.ACCTS                         0
SEC.OVERDUE.ACCTS                        0
SEC.CURRENT.BALANCE                      0
SEC.SANCTIONED.AMOUNT                    0
SEC.DISBURSED.AMOUNT                     0
PRIMARY.INSTAL.AMT                       0
SEC.INSTAL.AMT                           0
NEW.ACCTS.IN.LAST.SIX.MONTHS             0
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS      0
AVERAGE.ACCT.AGE                         0
CREDIT.HISTORY.LENGTH                    0
NO.OF_INQUIRIES                          0
loan_default                             0
dtype: int64
```

We can see that Employement.type has 7661 null values

Data Cleaning

Observations for data cleaning

1. Lot of column names have '.' in the name. Need to replace it with '_'
2. Most of the categorical variables have data type of int insted of object/category
3. Date of birth, disbursal date etc. does not have date data type. Need to change the datatype.
4. Account age and Credit history columns have dates, need to convert it to number of days
5. Need to calculate customer age from Date of birth
6. Looking at df.info(), only Employment Type has null values

```
# Replacing '.' with '_' in the column names

df.columns = df.columns.str.replace('.', '_')
df.columns
```

```
Index(['UniqueID', 'disbursed_amount', 'asset_cost', 'ltv', 'branch_id',
       'supplier_id', 'manufacturer_id', 'Current_pincode_ID', 'Date_of_Birth',
       'Employment_Type', 'DisbursalDate', 'State_ID', 'Employee_code_ID',
       'MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag', 'VoterID_flag',
       'Driving_flag', 'Passport_flag', 'PERFORM_CNS_SCORE',
       'PERFORM_CNS_SCORE_DESCRIPTION', 'PRI_NO_OF_ACCTS', 'PRI_ACTIVE_ACCTS',
       'PRI_OVERDUE_ACCTS', 'PRI_CURRENT_BALANCE', 'PRI_SANCTIONED_AMOUNT',
       'PRI_DISBURSED_AMOUNT', 'SEC_NO_OF_ACCTS', 'SEC_ACTIVE_ACCTS',
       'SEC_OVERDUE_ACCTS', 'SEC_CURRENT_BALANCE', 'SEC_SANCTIONED_AMOUNT',
       'SEC_DISBURSED_AMOUNT', 'PRIMARY_INSTAL_AMT', 'SEC_INSTAL_AMT',
       'NEW_ACCTS_IN_LAST_SIX_MONTHS', 'DELINQUENT_ACCTS_IN_LAST_SIX_MONTHS',
       'AVERAGE_ACCT_AGE', 'CREDIT_HISTORY_LENGTH', 'NO_OF_INQUIRIES',
       'loan_default'],
      dtype='object')
```

```
#Dropping unnecessary columns

df.drop(columns= ['UniqueID','supplier_id','Current_pincode_ID','Employee_code_ID',
                  'MobileNo_Avl_Flag', 'PERFORM_CNS_SCORE_DESCRIPTION'],axis = 1, inplace = True)
```

```
df.shape
```

```
(233154, 35)
```

```
#Datatype of branchID,manufacturerID, StateID etc. are given as Int,

#need to convert them to object datatype

Categorical_data = ['loan_default','branch_id','manufacturer_id','Employment_Type' ,'State_ID']
df[Categorical_data] = df[Categorical_data].astype(object)

# or
# df['branch_id'] = df['branch_id'].astype('category')
```

```
#Converting Object data type to appropriate date format

df['Date_of_Birth'] = pd.to_datetime(df['Date_of_Birth'], format='%d-%m-%y')
df['DisbursalDate']= pd.to_datetime(df['DisbursalDate'], format='%d-%m-%y')
```

```
# Extracting Age from Date of Birth column

def from_dob_to_age(born):
    today = datetime.date.today()
    return today.year - born.year - ((today.month, today.day) < (born.month, born.day))

df['age'] = df['Date_of_Birth'].apply(lambda x: from_dob_to_age(x))
df.drop(columns=['Date_of_Birth'],axis=1,inplace= True)
df.drop(columns=['DisbursalDate'],axis=1,inplace= True)
```

```
# Two features **AVERAGE.ACCT.AGE** and **CREDIT.HISTORY.LENGTH** in total months values


years1 = df.AVERAGE_ACCT_AGE.str.split(" ").str[0].str.split("y").str[0]

months1 = df.AVERAGE_ACCT_AGE.str.split(" ").str[1].str.split("m").str[0]

years1 = np.array(years1).astype(int)
months1 = np.array(months1).astype(int)

df.AVERAGE_ACCT_AGE = years1*12 + months1   # to convert years to months, *12



#Changing CREDIT_HISTORY_LENGTH in total months values

years2 = df.CREDIT_HISTORY_LENGTH.str.split(" ").str[0].str.split("y").str[0]
months2 = df.CREDIT_HISTORY_LENGTH.str.split(" ").str[1].str.split("m").str[0]
years2 = np.array(years2).astype(int)
months2 = np.array(months2).astype(int)
df.CREDIT_HISTORY_LENGTH = years2*12 + months2  # to convert years to months, *12
```

```
df.head()
```

	disbursed_amount	asset_cost	ltv	branch_id	manufacturer_id	Employment_Type
0	50578	58400	89.55	67	45	Salaried
1	53278	61360	89.63	67	45	Self employed
2	52378	60300	88.39	67	45	Self employed
3	46349	61500	76.42	67	45	Salaried
4	43594	78256	57.50	67	86	Self employed

5 rows × 34 columns

To return counts of unique values we use value_counts() method

```
#Handling null values in Employment type

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

```
Self employed    127635
Salaried          97858
Name: Employment_Type, dtype: int64
```

```
df['Employment_Type'].describe()
```

```
count            225493
unique                2
top       Self employed
freq             127635
Name: Employment_Type, dtype: object
```

```
df["Employment_Type"].isnull().sum()
```

```
7661
```

As we can see, Employment type has 7661 null values

Since Employment Type is a categorical value, we can replace null values of it using mode() function

```
df['Employment_Type'].fillna(df['Employment_Type'].mode()[0], inplace= True)

# or
# df['Employment_Type'].fillna(df['Employment_Type'].value_counts().index[0], inplace=True)


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

Exploratory Data Analysis

How is the target variable distributed overall?

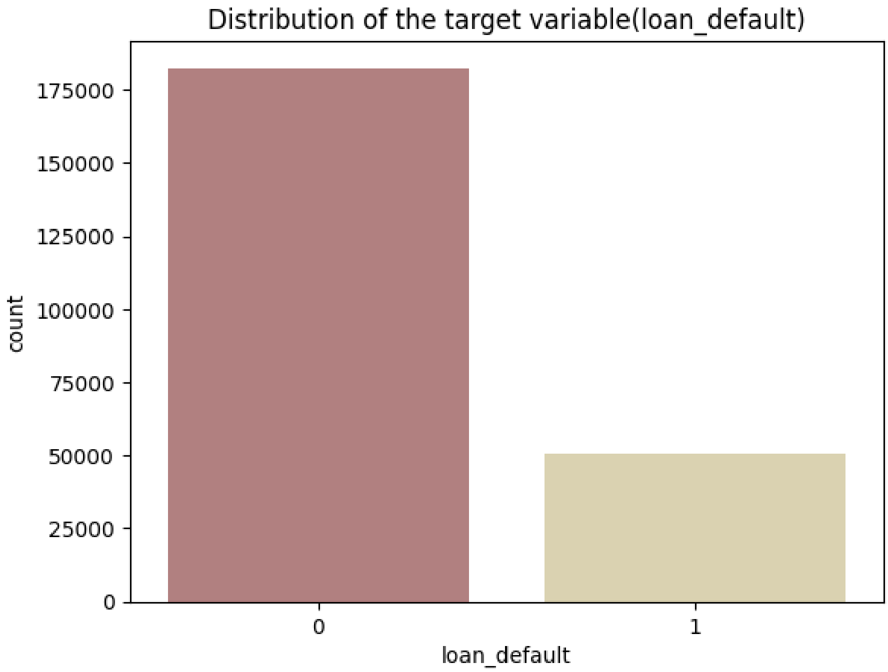
```
loan_default = df[df["loan_default"]==1]
loan_normal = df[df["loan_default"]==0]
```

```
(loan_default.shape,loan_normal.shape)
```

```
((50611, 34), (182543, 34))
```

```
sns.countplot(x='loan_default', data = df, palette = 'pink')
plt.title('Distribution of the target variable(loan_default)')
```

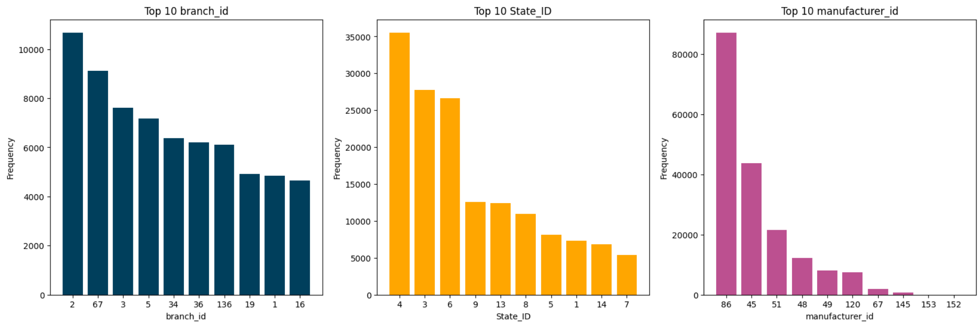
```
Text(0.5, 1.0, 'Distribution of the target variable(loan_default)')
```



We clearly observe that, this is an Imbalanced dataset. We need to resolve class imbalance before model building.

Distribution of the target variable across the various categories such as branch, state, manufacturer, etc.

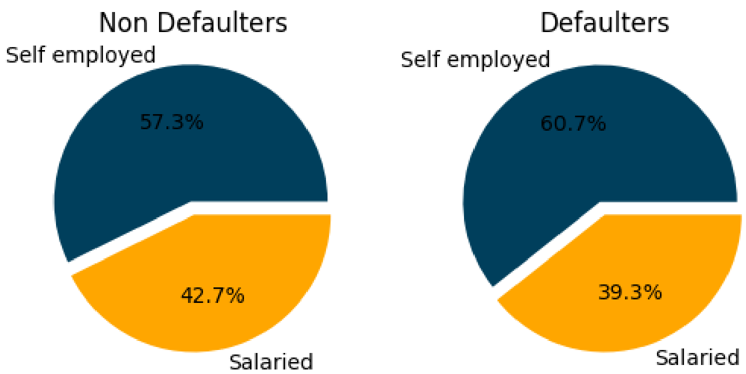
```
cate = ['branch_id','State_ID','manufacturer_id']
colors= ['#003f5c','#ffa600','#bc5090']
plt.figure(figsize= (20,6))


for i, column in enumerate(cate, 1):
    plt.subplot(1, 3, i)
    plt.bar(df[df['loan_default'] == 0][column].value_counts().head(10).index.astype(str),
            df[df['loan_default'] == 0][column].value_counts().head(10),color= colors[i-1])
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.title('Top 10 '+column)
```



Employment type wise distribution of defaulters

Use pie charts to express how different types of employment defines defaulter and non-defaulters

```
labels = [ 'Self employed','Salaried']

plt.subplot(1,2,1);
explode = (0.1,0);
plt.pie(df[df["loan_default"] ==0]['Employment_Type'].value_counts(), explode=explode, labels = labels, colors= ['#003
plt.title('Non Defaulters');

plt.subplot(1,2,2);
explode = (0.1,0);
plt.pie(df[df["loan_default"] ==1]['Employment_Type'].value_counts(), explode=explode, labels = labels,colors= ['#003f
plt.title('Defaulters');

plt.show()
```



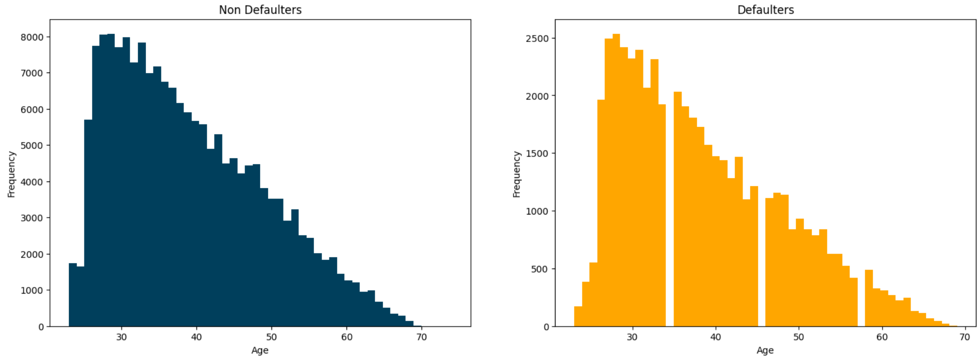
For both defaulter and non defaulter, the distribution looks the same. This does not provide any meaningful insight.

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

```
df["age"].describe()
```

```
count    233154.000000
mean         39.078892
std           9.808210
min          23.000000
25%          31.000000
50%          37.000000
75%          46.000000
max          74.000000
Name: age, dtype: float64
```

```
plt.figure(figsize= (18,6))
plt.subplot(1,2,1);
plt.hist(df[df['loan_default']==0]['age'], bins= 50, color='#003f5c');
plt.title('Non Defaulters');
plt.xlabel('Age');
plt.ylabel('Frequency');
plt.subplot(1,2,2);
plt.hist(df[df['loan_default']==1]['age'], bins= 50,color= '#ffa600');
plt.title('Defaulters');
plt.xlabel('Age');
plt.ylabel('Frequency');
```



In both cases the distribution looks the same. Age group 25-35 looks the most borrower

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

```
flags = ['Aadhar_flag','VoterID_flag','PAN_flag']

print('''Types of IDs given in defaulted cases:
            Defaulters''')

for i in range(len(flags)):
    Value1 = sum(loan_default[flags[i]])

    print(f"{flags[i]}       {Value1}")
```

```
Types of IDs given in defaulted cases:
            Defaulters
Aadhar_flag       41065
VoterID_flag       8816
PAN_flag       3877
```

To calculate a correlation matrix using the corr() function:

This will return a DataFrame where row i and column j contains the correlation coefficient.

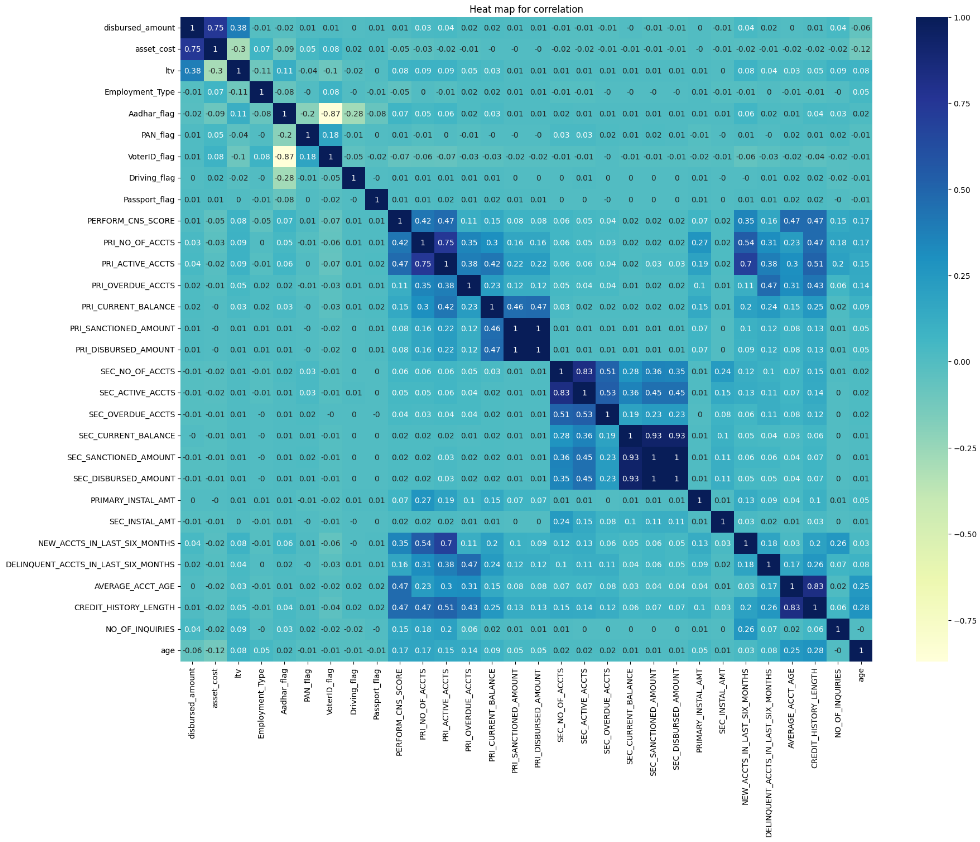
The correlation coefficient is a value between -1 and 1

A value close to 1 indicates a strong positive relationship, a value close to -1 indicates a strong negative relationship, and a value close to 0 indicates no relationship.

Heatmap for Correlation

```
corr_matrix = df.corr().round(2)
plt.subplots(figsize=(20,15))
sns.heatmap(corr_matrix, annot=True, cmap="YlGnBu")
plt.title('Heat map for correlation')
```

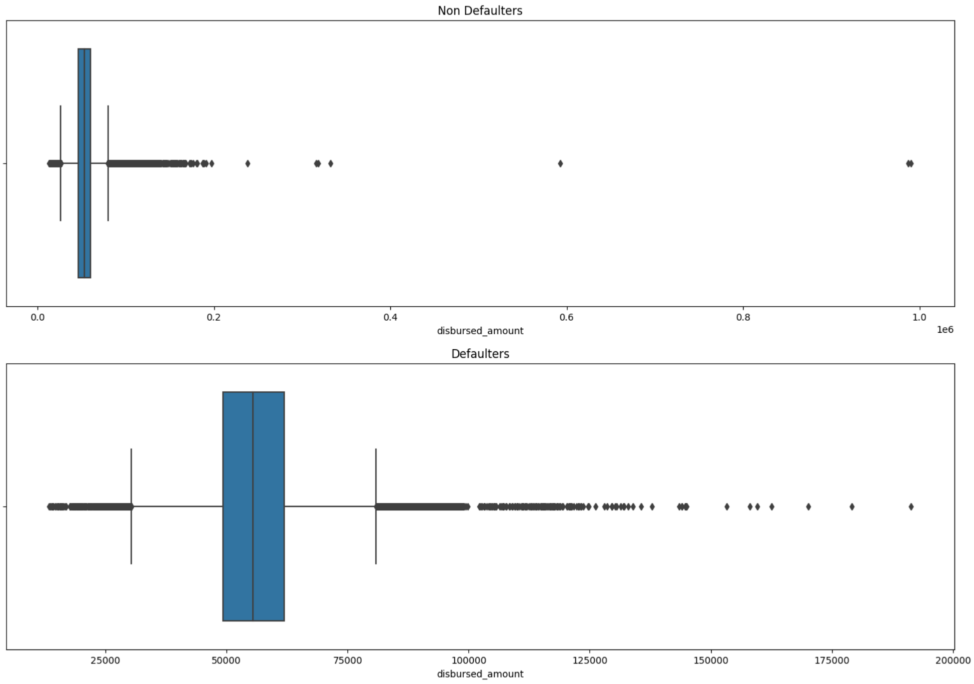
```
Text(0.5, 1.0, 'Heat map for correlation')
```



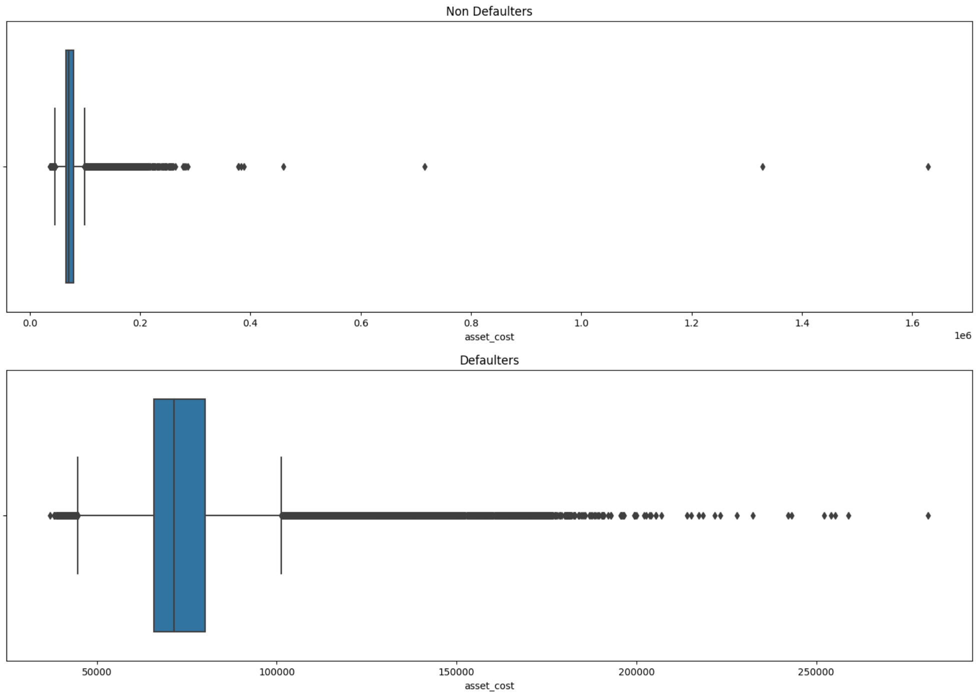
```
quantative_data = ['disbursed_amount', 'asset_cost', 'ltv','PERFORM_CNS_SCORE','age']
df[quantative_data].describe()
```

	disbursed_amount	asset_cost	ltv	PERFORM_CNS_SCORE	ag
count	233154.000000	2.331540e+05	233154.000000	233154.000000	233154.00000

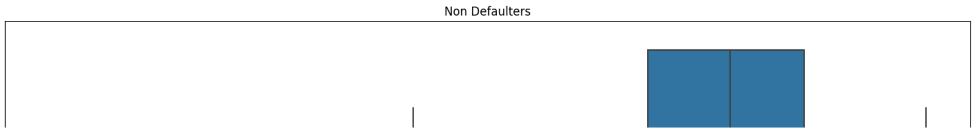
```
#Disbursed amount

plt.figure(figsize= (18,12))
plt.subplot(2,1,1)
plt.title('Non Defaulters')
sns.boxplot(x='disbursed_amount', data=df[df['loan_default'] == 0])

plt.subplot(2,1,2)
plt.title('Defaulters')
sns.boxplot(x='disbursed_amount', data=df[df['loan_default'] == 1])
plt.show()
```



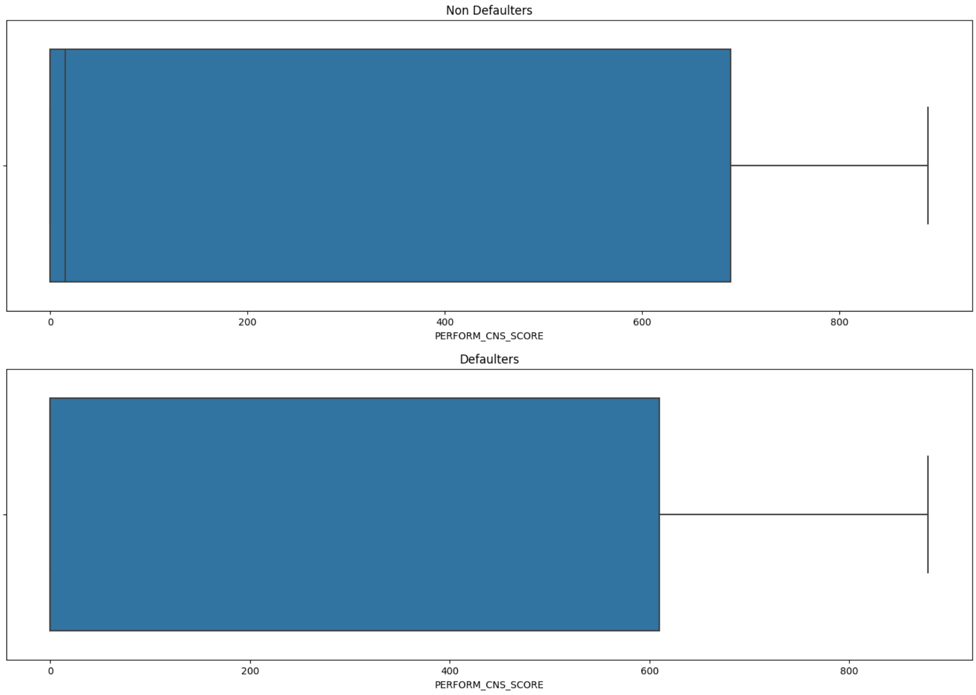

```
#Asset cost

plt.figure(figsize= (18,12))
plt.subplot(2,1,1)
sns.boxplot(x='asset_cost', data=df[df['loan_default'] == 0])
plt.title('Non Defaulters')

plt.subplot(2,1,2)
sns.boxplot(x='asset_cost', data=df[df['loan_default'] == 1])
plt.title('Defaulters')
plt.show()
```



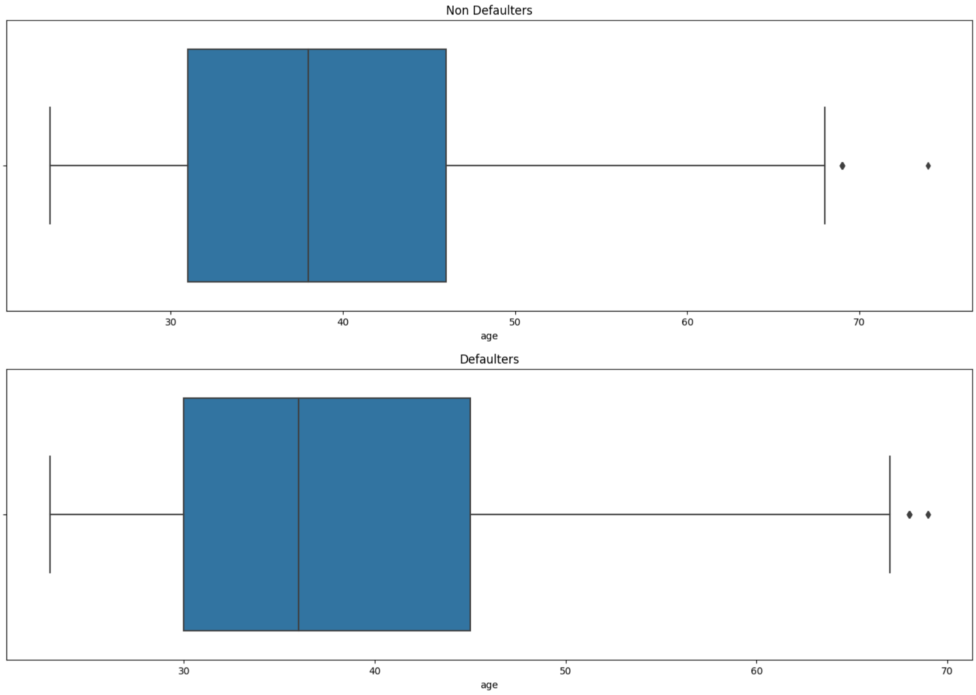

```
#ltv

plt.figure(figsize= (18,12))
plt.subplot(2,1,1)
sns.boxplot(x='ltv', data=df[df['loan_default'] == 0])
plt.title('Non Defaulters')

plt.subplot(2,1,2)
sns.boxplot(x='ltv', data=df[df['loan_default'] == 1])
plt.title('Defaulters')
plt.show()
```




```
#PERFORM CNS SCORE

plt.figure(figsize= (18,12))
plt.subplot(2,1,1)
sns.boxplot(x='PERFORM_CNS_SCORE', data=df[df['loan_default'] == 0])
plt.title('Non Defaulters')

plt.subplot(2,1,2)
sns.boxplot(x='PERFORM_CNS_SCORE', data=df[df['loan_default'] == 1])
plt.title('Defaulters')
plt.show()
```



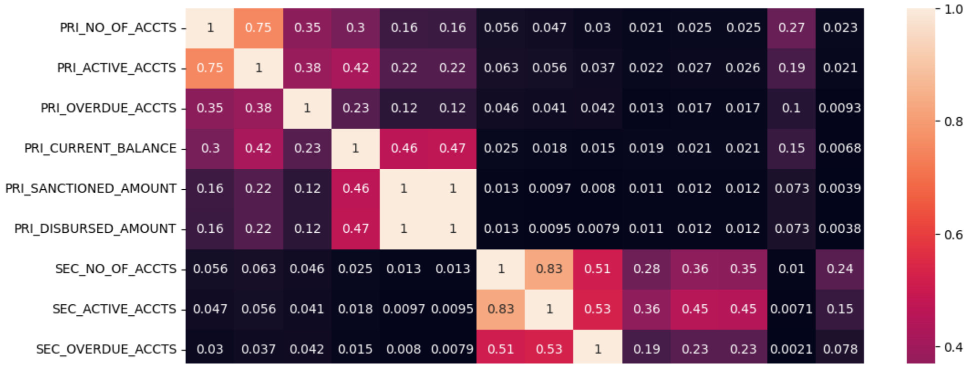

```
#Age

plt.figure(figsize= (18,12))
plt.subplot(2,1,1)
sns.boxplot(x='age', data=df[df['loan_default'] == 0])
plt.title('Non Defaulters')

plt.subplot(2,1,2)
sns.boxplot(x='age', data=df[df['loan_default'] == 1])
plt.title('Defaulters')
plt.show()
```



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

Looking at account details

```
# Checking the correlation between primary and secondary accounts

plt.figure(figsize=(12,8))
sns.heatmap(df[['PRI_NO_OF_ACCTS', 'PRI_ACTIVE_ACCTS', 'PRI_OVERDUE_ACCTS','PRI_CURRENT_BALANCE',
                'PRI_SANCTIONED_AMOUNT','PRI_DISBURSED_AMOUNT','SEC_NO_OF_ACCTS', 'SEC_ACTIVE_ACCTS',
                'SEC_OVERDUE_ACCTS','SEC_CURRENT_BALANCE','SEC_SANCTIONED_AMOUNT',
                'SEC_DISBURSED_AMOUNT','PRIMARY_INSTAL_AMT', 'SEC_INSTAL_AMT']].corr(),annot=True)
plt.show()
```




```
#Combining Primary and Sec accounts

df['NO_OF_ACCTS'] = df['PRI_NO_OF_ACCTS'] + df['SEC_NO_OF_ACCTS']
df['ACTIVE_ACCTS'] = df['PRI_ACTIVE_ACCTS'] + df['SEC_ACTIVE_ACCTS']
df['OVERDUE_ACCTS'] = df['PRI_OVERDUE_ACCTS'] + df['SEC_OVERDUE_ACCTS']
df['CURRENT_BALANCE'] = df['PRI_CURRENT_BALANCE'] + df['SEC_CURRENT_BALANCE']
df['SANCTIONED_AMOUNT'] = df['PRI_SANCTIONED_AMOUNT'] + df['SEC_SANCTIONED_AMOUNT']
df['DISBURSED_AMOUNT'] = df['PRI_DISBURSED_AMOUNT'] + df['SEC_DISBURSED_AMOUNT']
df['INSTAL_AMT'] = df['PRIMARY_INSTAL_AMT'] + df['SEC_INSTAL_AMT']

df.drop(columns = ['PRI_NO_OF_ACCTS', 'PRI_ACTIVE_ACCTS', 'PRI_OVERDUE_ACCTS',
                   'PRI_CURRENT_BALANCE', 'PRI_SANCTIONED_AMOUNT','PRI_DISBURSED_AMOUNT',
                   'SEC_NO_OF_ACCTS', 'SEC_ACTIVE_ACCTS', 'SEC_OVERDUE_ACCTS','SEC_CURRENT_BALANCE',
                'SEC_SANCTIONED_AMOUNT', 'SEC_DISBURSED_AMOUNT','PRIMARY_INSTAL_AMT', 'SEC_INSTAL_AMT'],
        inplace= True, axis = 1)
```

Is there a difference between the sanctioned and disbursed amount of primary and secondary loans?

```
acc_details = ['NO_OF_ACCTS','ACTIVE_ACCTS','OVERDUE_ACCTS','CURRENT_BALANCE',
      'SANCTIONED_AMOUNT','DISBURSED_AMOUNT','INSTAL_AMT']
```

Non defaulters

```
df[df['loan_default'] == 0][acc_details].describe()
```

	NO_OF_ACCTS	ACTIVE_ACCTS	OVERDUE_ACCTS	CURRENT_BALANCE	SANCTIONED_AMOUN
count	182543.000000	182543.000000	182543.000000	1.825430e+05	1.825430e+0
mean	2.599886	1.110971	0.152063	1.854112e+05	2.405482e+0
std	5.338380	2.051149	0.547825	1.014777e+06	1.253845e+0
min	0.000000	0.000000	0.000000	-6.678296e+06	0.000000e+0
25%	0.000000	0.000000	0.000000	0.000000e+00	0.000000e+0
50%	1.000000	0.000000	0.000000	0.000000e+00	0.000000e+0
75%	3.000000	1.000000	0.000000	4.052250e+04	7.270350e+0
max	354.000000	144.000000	25.000000	9.652492e+07	1.058657e+0

Defaulters

```
df[df['loan_default'] == 1][acc_details].describe()
```

	NO_OF_ACCTS	ACTIVE_ACCTS	OVERDUE_ACCTS	CURRENT_BALANCE	SANCTIONED_AMOUNT
count	50611.000000	50611.000000	50611.000000	5.061100e+04	5.061100e+04
mean	2.138428	0.911166	0.206101	1.205323e+05	1.726052e+05
std	5.094236	1.712724	0.618799	7.314120e+05	4.528697e+06
min	0.000000	0.000000	0.000000	-2.013721e+06	0.000000e+00
25%	0.000000	0.000000	0.000000	0.000000e+00	0.000000e+00
max	453.000000	35.000000	18.000000	4.505116e+07	1.000000e+09

Model building

Seperate features and target variable

```
X = df.drop(columns = 'loan_default', axis=1)
y = df['loan_default']
# y.info()
y = y.astype('int')
```

Separate the dataframes into x_train, x_test, y_train, and y_test

```
from sklearn.model_selection import train_test_split
```

```
random_state=42

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

Scaling data before model training and testing

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train.shape[0], X_test.shape[0]
```

```
(163207, 69947)
```

```
# !pip install imbalanced-learn
```

Performing Under Sampling using SMOTE

```
from imblearn.under_sampling import RandomUnderSampler
```

```
us = RandomUnderSampler(sampling_strategy='auto', random_state=None)
```

```
X_train
```

```
array([[-0.92875044, -1.01053819,  0.36965373, ..., -0.18895388,
        -0.18813582, -0.0889625 ],
       [-0.52173899, -0.62015584,  0.38625038, ..., -0.18895388,
        -0.18813582, -0.0889625 ],
       [ 3.42431954,  3.49791107, -0.36584004, ..., -0.18895388,
        -0.18813582, -0.0889625 ],
       ...,
       [-0.49983276, -0.5679384 ,  0.11022184, ...,  0.47794942,
         0.47801129, -0.02247264],
       [ 1.31568593,  0.7079149 ,  0.6928517 , ..., -0.18092938,
        -0.18012042, -0.06472238],
```

```
       [-0.69825878, -0.71579734, -0.00508122, ..., -0.18895388,
        -0.18813582, -0.08072647]])
```

```
y_train
```

```
81759     0
115977    0
51043     0
18380     0
192570    0
         ..
119879    1
103694    0
131932    0
146867    0
121958    0
Name: loan_default, Length: 163207, dtype: int64
```

```
X_train,y_train = us.fit_resample(X_train,y_train)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn import metrics
```

```
def score_output(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("==================================== Train Result:=============================================\n")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________________________________________________________________
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________________________________________________________________
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("===================================== Test Result:=============================================\n")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________________________________________________________________
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________________________________________________________________
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

Model -1 Logistic Regession

```
lr_clf = LogisticRegression(solver='liblinear')
lr_clf.fit(X_train, y_train)

score_output(lr_clf, X_train, y_train, X_test, y_test, train=True)
score_output(lr_clf, X_train, y_train, X_test, y_test, train=False)
```

```
==================================== Train Result:=============================================

Accuracy Score: 59.57%
_______________________________________________________________________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.604867      0.587958  0.595671      0.596413      0.596413
recall         0.551826      0.639517  0.595671      0.595671      0.595671
f1-score       0.577130      0.612655  0.595671      0.594893      0.594893
support    35716.000000  35716.000000  0.595671  71432.000000  71432.000000
_______________________________________________________________________________________________________
Confusion Matrix:
 [[19709 16007]
 [12875 22841]]
```

```
==================================== Test Result:=============================================

Accuracy Score: 56.83%
_______________________________________________________________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.847647      0.276278  0.568273      0.561963      0.725976
recall         0.550389      0.634374  0.568273      0.592381      0.568273
f1-score       0.667416      0.384919  0.568273      0.526167      0.607259
support    55052.000000  14895.000000  0.568273  69947.000000  69947.000000
_______________________________________________________________________________________________
Confusion Matrix:
 [[30300 24752]
 [ 5446  9449]]
```

Model -2 Random Forest

```
rf_clf = RandomForestClassifier(n_estimators=15, max_depth=50, max_features=12, min_samples_leaf=100, random_state=42)
rf_clf.fit(X_train,y_train)
score_output(rf_clf, X_train, y_train, X_test, y_test, train=True)
score_output(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
===================================== Train Result:=============================================

Accuracy Score: 63.37%
_______________________________________________________________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.647380      0.622263  0.633652      0.634821      0.634821
recall         0.587076      0.680227  0.633652      0.633652      0.633652
f1-score       0.615755      0.649955  0.633652      0.632855      0.632855
support    35716.000000  35716.000000  0.633652  71432.000000  71432.000000
_______________________________________________________________________________________________
Confusion Matrix:
 [[20968 14748]
 [11421 24295]]

==================================== Test Result:=============================================

Accuracy Score: 57.73%
_______________________________________________________________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.853623      0.283692  0.577323      0.568657      0.732258
recall         0.558781      0.645854  0.577323      0.602318      0.577323
f1-score       0.675427      0.394222  0.577323      0.534825      0.615545
support    55052.000000  14895.000000  0.577323  69947.000000  69947.000000
_______________________________________________________________________________________________
Confusion Matrix:
 [[30762 24290]
 [ 5275  9620]]
```

Model -3 KNN

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)

score_output(knn_clf, X_train, y_train, X_test, y_test, train=True)
score_output(knn_clf, X_train, y_train, X_test, y_test, train=False)
```

```
===================================== Train Result:=============================================

Accuracy Score: 71.23%
_______________________________________________________________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.714942      0.709640  0.712258      0.712291      0.712291
recall         0.706014      0.718502  0.712258      0.712258      0.712258
```

```
f1-score       0.710450      0.714043  0.712258      0.712247      0.712247
support    35716.000000  35716.000000  0.712258  71432.000000  71432.000000
_______________________________________________
Confusion Matrix:
 [[25216 10500]
 [10054 25662]]

===================================== Test Result:=============================================

Accuracy Score: 55.15%
_______________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.821822      0.251523  0.551475      0.536672      0.700378
recall         0.549190      0.559919  0.551475      0.554555      0.551475
f1-score       0.658399      0.347117  0.551475      0.502758      0.592112
support    55052.000000  14895.000000  0.551475  69947.000000  69947.000000
_______________________________________________
Confusion Matrix:
 [[30234 24818]
 [ 6555  8340]]
```