Credit-risk-assignment-naren-hazare

March 25, 2024

1 BFSI Credit Risk Assignment

ECL method is used for provisioning the capital buffer to protect banks against possible default of the customers. The ECL provisioning is a mandatory accounting principle set by the Basel III norms.

The Basel norms, also known as the Basel Accords or Basel Regulations, are a set of international regulatory standards for the banking industry. These norms were developed by the Basel Committee on Banking Supervision, which is an international committee of banking supervisors from around the world. The committee was formed in 1974 by the central bank governors of the Group of Ten (G-10) countries. The history of the Basel norms can be traced back to the late 1970s and early 1980s when the banking industry was facing a series of crises and failures. These crises were caused by a combination of factors, including insufficient capital and liquidity, inadequate risk management and weak supervisory oversight. In response to these crises, the Basel Committee began to develop a set of international standards for bank capital and risk management to strengthen the resilience of the global banking system and reduce the risk of bank failures.

The first Basel Accord, known as Basel I, was issued in 1988, which introduced the first set of minimum capital requirements for banks. Basel I was revised in 2004 with the introduction of Basel II, which aimed to improve the risk sensitivity of the capital requirements and provide a more sophisticated approach to calculating capital ratios. Subsequently, Basel III was introduced in 2009, with stricter rules and regulations, largely in response to the financial crisis of 2007–2008 and the ensuing economic recession. It aimed to strengthen the resilience of the banking system against financial stress and improve the ability of banks to absorb losses.

The Basel norms are used to ensure that banks maintain sufficient levels of capital and liquidity to withstand financial shocks and reduce the risk of bank failures. The norms cover a range of areas, including minimum capital requirements, risk-weighted assets and the calculation of capital ratios. Banks are required to comply with the Basel norms to ensure the stability and resilience of the global financial system. Non-compliance with these norms can result in regulatory penalties and other consequences for banks.

To comply with the regulatory norms, a bank needs to provision funds. Provisioning refers to the process of setting aside funds to cover potential losses from defaulted loans. Therefore, provisioning is an important part of a bank's risk management strategy. The provisioning by banks is also an important macroeconomic metric to gauge the economic conditions of a country. Banks may use several methods to calculate the amount of provisioning required, such as lifetime expected loss (LEL), stressed loss analysis (SLA), current expected credit loss (CECL) and through-the-cycle (TTC) and expected credit loss (ECL).

For this assignment, we will focus on the expected credit loss (ECL) calculation method.

Expected credit loss (ECL) computation is a method used in credit risk management to determine the amount of loss a bank is expected to incur in the event a borrower defaults on their loan. Different banks may use different methodologies for calculating the expected credit loss (ECL) and provisioning. rk of a bank. Banks are allowed to use their own methodologies and incorporate factors relevant to their specific business operations. Some banks may choose to use historical data and statistical models to estimate the components of ECL calculation, while others may rely on expert judgement. The choice of the method can vary depending on factors such as the bank's risk appetite, the nature of the loans and the available data. Additionally, some banks may include certain external factors, such as macroeconomic conditions, in their calculations, while others may not.

The formula for ECL typically used in practice is as follows:

ECL = EAD x PD x LGD

Expected credit loss = Exposure at default x Probability of Default x Loss given default]

ECLs are calculated based on the exposure at default (EAD), probability of default (PD) and the loss given default (LGD) for each borrower. Banks can calculate the ECL for different points in time based on their risk management strategy and regulatory requirements.

For this assignment, we will consider the latest date from which the data is available as the point in time. This means we will estimate the expected credit loss (ECL) for the borrower assuming that the borrower has defaulted at the present point in time.

[1]:
```python
#importing important libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```python
#importing the datasets
main = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\main_loan_base.csv")
monthly = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\monthly_balance_base.csv")
repayment = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\repayment_base.csv")
```

[3]:
```python
#importing the test datasets
test_main = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\test_main_loan_base.csv")
test_monthly = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\test_monthly_balance_base.csv")
test_repayment = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\test_repayment_base.csv")
```

[4]:
```python
main.sample(5)
```

[4]:

```
          loan_acc_num          customer_name  \
26115   LN53044141  Stuvan Bhattacharyya
46374   LN78872768         Gokul Upadhyay
35588   LN38410221             Amira Walla
3211    LN35354345               Zaina Roy
49893   LN47179047            Mannat Badal

                               customer_address loan_type  loan_amount  \
26115    65/39\nDasgupta Street\nMadurai 547606       Car       412562
46374  H.No. 62, Sehgal Nagar\nTadipatri-851958       Car      1613603
35588       H.No. 05, Toor Zila, Sasaram-844919  Personal       232548
3211          15/39\nDivan Path\nGwalior-035353  Personal       237990
49893     H.No. 008, Walla Nagar, Ranchi 880700       Car       538696

       collateral_value  cheque_bounces  number_of_loans  missed_repayments  \
26115           3069.25               2                3                 24
46374         151235.59               1                2                 15
35588          63545.29               0                0                  5
3211           18832.69               5                2                 20
49893          43223.73               4                2                 38

       vintage_in_months  tenure_years  interest  monthly_emi disbursal_date  \
26115                 70             5       8.4      8444.47     2016-04-03
46374                 74             3      11.9     53517.67     2018-08-15
35588                 54             2      14.4     11209.29     2013-10-09
3211                  90             4      10.1      6047.48     2017-02-23
49893                 15             5      11.6     11874.39     2016-07-28

      default_date
26115   2017-12-02
46374   2019-10-10
35588   2014-09-22
3211    2017-12-06
49893   2020-07-04
```

[5]:

```
main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   loan_acc_num        50000 non-null  object
 1   customer_name       50000 non-null  object
 2   customer_address    50000 non-null  object
 3   loan_type           50000 non-null  object
 4   loan_amount         50000 non-null  int64
```

```
 5   collateral_value   50000 non-null  float64
 6   cheque_bounces     50000 non-null  int64
 7   number_of_loans    50000 non-null  int64
 8   missed_repayments  50000 non-null  int64
 9   vintage_in_months  50000 non-null  int64
 10  tenure_years       50000 non-null  int64
 11  interest           50000 non-null  float64
 12  monthly_emi        50000 non-null  float64
 13  disbursal_date     50000 non-null  object
 14  default_date       50000 non-null  object
dtypes: float64(3), int64(6), object(6)
memory usage: 5.7+ MB
```

[6]:
```
main['disbursal_date'] = pd.to_datetime(main['disbursal_date'],␣
 ↪infer_datetime_format=True)
main['default_date'] = pd.to_datetime(main['default_date'],␣
 ↪infer_datetime_format=True)
#data['repayment_date'] = pd.to_datetime(data['repayment_date'],␣
 ↪infer_datetime_format=True)
```

[7]:
```
test_main['disbursal_date'] = pd.to_datetime(test_main['disbursal_date'],␣
 ↪infer_datetime_format=True)
test_main['default_date'] = pd.to_datetime(test_main['default_date'],␣
 ↪infer_datetime_format=True)
```

[8]:
```
#Creating a new data set with zero duplicates.
main = main[~main['loan_acc_num'].duplicated()]
print(main.shape)
```

```
(49985, 15)
```

[9]:
```
#Creating a new data set with zero duplicates.
test_main = test_main[~test_main['loan_acc_num'].duplicated()]
print(test_main.shape)
```

```
(9997, 15)
```

[10]:
```
repayment.sample(5)
```

[10]:

	loan_acc_num	repayment_amount	repayment_date
461523	LN79484850	376.31	2016-06-17
607027	LN24199017	266.76	2021-09-25
511840	LN24852729	533.80	2022-02-28
613263	LN64304296	46083.68	2020-04-16
318579	LN77598978	4663.22	2022-06-20

[11]:
```
repayment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 626601 entries, 0 to 626600
Data columns (total 3 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   loan_acc_num      626601 non-null  object
 1   repayment_amount  626601 non-null  float64
 2   repayment_date    626601 non-null  object
dtypes: float64(1), object(2)
memory usage: 14.3+ MB
```

[12]:
```
repayment['loan_acc_num'].nunique()
```

[12]: 46008

[13]:
```
repayment = round(repayment.groupby('loan_acc_num')['repayment_amount'].sum(),2)
repayment = pd.DataFrame({"loan_acc_num":repayment.index, "repayment_amount":
  ↪repayment.values})
repayment.head()
```

[13]:

	loan_acc_num	repayment_amount
0	LN10000701	40020.99
1	LN10001077	112218.47
2	LN10004116	290634.94
3	LN10007976	337321.72
4	LN10010204	61290.49

[14]:
```
test_repayment = round(test_repayment.
  ↪groupby('loan_acc_num')['repayment_amount'].sum(),2)
test_repayment = pd.DataFrame({"loan_acc_num":test_repayment.index,␣
  ↪"repayment_amount":test_repayment.values})
test_repayment.head()
```

[14]:

	loan_acc_num	repayment_amount
0	LN10011015	1725.31
1	LN10028091	3560.31
2	LN10033713	11582.17
3	LN10045654	66181.74
4	LN10051605	87664.41

[15]:
```
main.shape
```

[15]: (49985, 15)

[16]:
```
repayment.shape
```

[16]: (46008, 2)

[17]:
```python
df = pd.merge(
    left=main,
    right=repayment,
    left_on='loan_acc_num',
    right_on='loan_acc_num',
    how='left'
)
```

[18]:
```python
test_df = pd.merge(
    left=test_main,
    right=test_repayment,
    left_on='loan_acc_num',
    right_on='loan_acc_num',
    how='left'
)
```

[19]:
```python
df.shape
```

[19]:
```
(49985, 16)
```

[20]:
```python
#deriving the target
df['target'] =␣
↪(df['loan_amount']-(df['collateral_value']+df['repayment_amount']))/
↪df['loan_amount']
```

[21]:
```python
df.sample(5)
```

[21]:
```
      loan_acc_num  customer_name                                customer_address  \
32509  LN81149619     Lagan Kale  H.No. 63, Bumb Circle\nDharmavaram 542104
1198   LN75032804  Manikya Kumer               15/031, Gulati, Gwalior 284732
11207  LN89734434     Siya Balan        35, Shankar Ganj\nBihar Sharif 540743
5065   LN37978442   Bhavin Yadav                  55/06\nDhar\nMadurai-565249
10319  LN33032272     Sara Koshy        809, Chandran Path\nBangalore 301072

      loan_type  loan_amount  collateral_value  cheque_bounces  \
32509  Personal        35436            344.39               0
1198   Personal        39527           2093.46               4
11207  Personal        81504           9862.72               1
5065   Personal       441347          99431.46               0
10319       Car      1202157         318746.51               1

       number_of_loans  missed_repayments  vintage_in_months  tenure_years  \
32509                0                  5                 68             5
1198                 2                 14                 39             2
11207                2                  5                 15             1
5065                 0                  2                147             4
10319                5                 20                 67             4
```

```
       interest  monthly_emi disbursal_date default_date  repayment_amount  \
32509      14.1       826.37     2019-02-28   2021-01-17          28532.60
1198        9.8      1820.32     2014-02-13   2015-04-27           6525.79
11207      11.8      7233.91     2017-06-07   2017-12-20          11511.17
5065       10.0     11193.70     2015-04-09   2018-11-17         290422.75
10319      11.2     31187.27     2014-08-28   2016-10-19               NaN

         target
32509  0.185095
1198   0.781940
11207  0.737757
5065   0.116672
10319       NaN
```

[22]:
```
df.isnull().sum()
```

[22]:
```
loan_acc_num             0
customer_name            0
customer_address         0
loan_type                0
loan_amount              0
collateral_value         0
cheque_bounces           0
number_of_loans          0
missed_repayments        0
vintage_in_months        0
tenure_years             0
interest                 0
monthly_emi              0
disbursal_date           0
default_date             0
repayment_amount      3977
target                3977
dtype: int64
```

[23]:
```
test_df.isnull().sum()
```

[23]:
```
loan_acc_num            0
customer_name           0
customer_address        0
loan_type               0
loan_amount             0
collateral_value        0
cheque_bounces          0
number_of_loans         0
missed_repayments       0
```

```
vintage_in_months      0
tenure_years           0
interest               0
monthly_emi            0
disbursal_date         0
default_date           0
repayment_amount     768
dtype: int64
```

[24]:
```
#df['repayment_date'] = df['repayment_date'].fillna(df['repayment_date'].
↪mode()[0])
```

[25]:
```
#null value imputation
df['repayment_amount'] = df['repayment_amount'].fillna(0)
```

[26]:
```
#null value imputation
test_df['repayment_amount'] = test_df['repayment_amount'].fillna(0)
```

[27]:
```
#null value imputation
df['target'] = df['target'].fillna(df['target'].mean())
```

[28]:
```
#creating new variable: feature engineering
df['due'] = df['loan_amount'] - df['repayment_amount']
```

[29]:
```
#creating new variable: feature engineering
test_df['due'] = test_df['loan_amount'] - test_df['repayment_amount']
```

[30]:
```
monthly.sample(5)
```

[30]:

	loan_acc_num	date	balance_amount
2826607	LN49230151	2014-01-25	9084.638007
3369105	LN75119807	2011-01-09	10018.130067
3271932	LN51738513	2008-09-15	26219.728290
2674919	LN95981566	2013-09-17	156307.498623
3773599	LN85040703	2013-06-28	7935.863790

[31]:
```
monthly.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4002490 entries, 0 to 4002489
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   loan_acc_num    object
 1   date            object
 2   balance_amount  float64
dtypes: float64(1), object(2)
```

```
memory usage: 91.6+ MB
```

[32]:
```python
monthly['loan_acc_num'].nunique()
```

[32]:
```
49671
```

[33]:
```python
monthly = round(monthly.groupby('loan_acc_num')['balance_amount'].mean(),2)
monthly = pd.DataFrame({"loan_acc_num":monthly.index, "average_monthly_balance":
 ↪monthly.values})
monthly.head()
```

[33]:

	loan_acc_num	average_monthly_balance
0	LN10000701	2301.88
1	LN10001077	2296.28
2	LN10004116	8887.38
3	LN10007976	9420.56
4	LN10010204	6446.21

[34]:
```python
test_monthly = round(test_monthly.groupby('loan_acc_num')['balance_amount'].
 ↪mean(),2)
test_monthly = pd.DataFrame({"loan_acc_num":test_monthly.index,␣
 ↪"average_monthly_balance":test_monthly.values})
test_monthly.head()
```

[34]:

	loan_acc_num	average_monthly_balance
0	LN10011015	25.09
1	LN10028091	62.53
2	LN10033713	182.41
3	LN10045654	1838.35
4	LN10051605	3374.17

[35]:
```python
#Merge the datasets
data = pd.merge(
    left=df,
    right=monthly,
    left_on='loan_acc_num',
    right_on='loan_acc_num',
    how='left'
)
```

[36]:
```python
#Merge the datasets
test_data = pd.merge(
    left=test_df,
    right=test_monthly,
    left_on='loan_acc_num',
    right_on='loan_acc_num',
    how='left'
```

```
)
```

[37]:
```
data.shape
```

[37]:
```
(49985, 19)
```

[38]:
```
test_data.shape
```

[38]:
```
(9997, 18)
```

[39]:
```
data.isnull().sum()
```

[39]:
```
loan_acc_num                 0
customer_name                0
customer_address             0
loan_type                    0
loan_amount                  0
collateral_value             0
cheque_bounces               0
number_of_loans              0
missed_repayments            0
vintage_in_months            0
tenure_years                 0
interest                     0
monthly_emi                  0
disbursal_date               0
default_date                 0
repayment_amount             0
target                       0
due                          0
average_monthly_balance    314
dtype: int64
```

[40]:
```
#null value imputation
data['average_monthly_balance'] = data['average_monthly_balance'].
  ↪fillna(data['average_monthly_balance'].mean())
```

[41]:
```
#null value imputation
test_data['average_monthly_balance'] = test_data['average_monthly_balance'].
  ↪fillna(test_data['average_monthly_balance'].mean())
```

[42]:
```
data.isnull().sum()
```

[42]:
```
loan_acc_num               0
customer_name              0
customer_address           0
loan_type                  0
```

```
loan_amount                0
collateral_value           0
cheque_bounces             0
number_of_loans            0
missed_repayments          0
vintage_in_months          0
tenure_years               0
interest                   0
monthly_emi                0
disbursal_date             0
default_date               0
repayment_amount           0
target                     0
due                        0
average_monthly_balance    0
dtype: int64
```

[43]:
```
test_data.isnull().sum()
```

[43]:
```
loan_acc_num               0
customer_name              0
customer_address           0
loan_type                  0
loan_amount                0
collateral_value           0
cheque_bounces             0
number_of_loans            0
missed_repayments          0
vintage_in_months          0
tenure_years               0
interest                   0
monthly_emi                0
disbursal_date             0
default_date               0
repayment_amount           0
due                        0
average_monthly_balance    0
dtype: int64
```

[44]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49985 entries, 0 to 49984
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   loan_acc_num             49985 non-null  object
```

```
 1   customer_name           49985 non-null  object
 2   customer_address        49985 non-null  object
 3   loan_type               49985 non-null  object
 4   loan_amount             49985 non-null  int64
 5   collateral_value        49985 non-null  float64
 6   cheque_bounces          49985 non-null  int64
 7   number_of_loans         49985 non-null  int64
 8   missed_repayments       49985 non-null  int64
 9   vintage_in_months       49985 non-null  int64
 10  tenure_years            49985 non-null  int64
 11  interest                49985 non-null  float64
 12  monthly_emi             49985 non-null  float64
 13  disbursal_date          49985 non-null  datetime64[ns]
 14  default_date            49985 non-null  datetime64[ns]
 15  repayment_amount        49985 non-null  float64
 16  target                  49985 non-null  float64
 17  due                     49985 non-null  float64
 18  average_monthly_balance 49985 non-null  float64
dtypes: datetime64[ns](2), float64(7), int64(6), object(4)
memory usage: 7.6+ MB
```

[45]:
```
data.describe()
```

[45]:
```
        loan_amount  collateral_value  cheque_bounces  number_of_loans  \
count  4.998500e+04      49985.000000    49985.000000     49985.000000
mean   3.817142e+05      57195.113444        1.764769         1.509573
std    5.037769e+05      93412.679667        1.760305         1.259326
min    2.000000e+03          0.070000        0.000000         0.000000
25%    2.393400e+04       3329.430000        0.000000         0.000000
50%    1.926920e+05      19866.280000        1.000000         1.000000
75%    4.334780e+05      62323.370000        3.000000         2.000000
max    1.999992e+06     592545.710000       11.000000         6.000000

       missed_repayments  vintage_in_months  tenure_years      interest  \
count       49985.000000       49985.000000  49985.000000  49985.000000
mean            9.807482          80.016705      2.994578     11.484611
std             7.787036          44.141987      1.415455      2.019790
min             0.000000          15.000000      1.000000      8.000000
25%             4.000000          44.000000      2.000000      9.700000
50%             8.000000          78.000000      3.000000     11.500000
75%            15.000000         113.000000      4.000000     13.200000
max            38.000000         258.000000      5.000000     15.000000

         monthly_emi  repayment_amount        target           due  \
count   49985.000000      4.998500e+04  49985.000000  4.998500e+04
mean    16593.115676      1.514337e+05      0.423265  2.302806e+05
std     26696.292090      2.554824e+05      0.225701  3.437854e+05
```

```
min        42.520000       0.000000e+00     -5.708000 -8.573322e+04
25%      1158.280000       6.761950e+03      0.242430  1.580488e+04
50%      6541.020000       4.849626e+04      0.423265  9.599627e+04
75%     19438.430000       1.668996e+05      0.600358  2.602902e+05
max    179521.680000       1.852111e+06      0.898372  1.997948e+06

       average_monthly_balance
count             49985.000000
mean               7679.277191
std               16071.151167
min                   0.100000
25%                 417.980000
50%                2186.470000
75%                7557.930000
max              261799.900000
```

[46]:
```
#dropping the records with negative LGD
data = data.drop(data[data['target']<0].index)
```

[47]:
```
test = test_data.copy()
```

EDA

[48]:
```
#separating numeric and categorical features
numeric_data = data.select_dtypes(include=[np.number])
categorical_data = data.select_dtypes(exclude=[np.number])
```

[49]:
```
#top 10 correlated features
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=10):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations")
print(get_top_abs_correlations(numeric_data, 10))
```

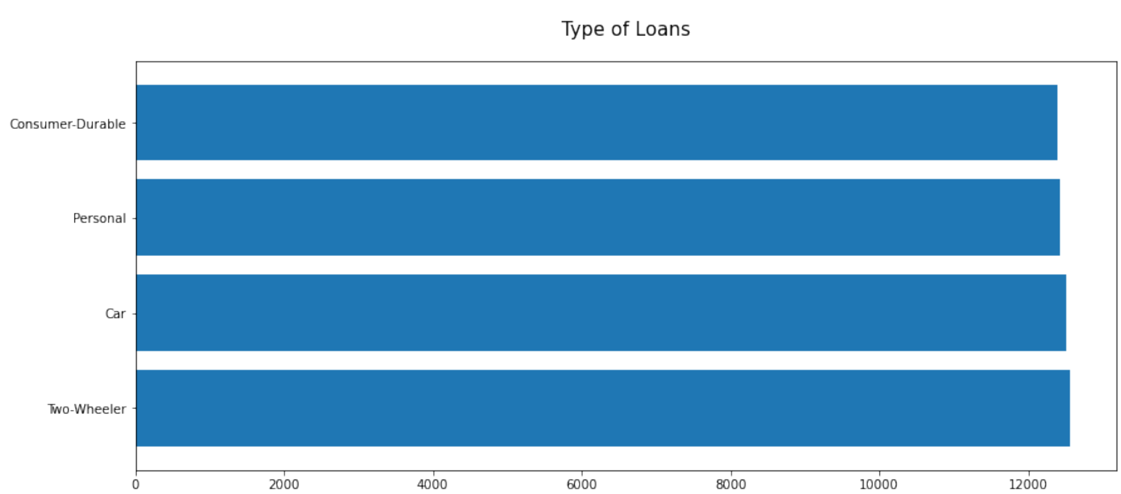
```
Top Absolute Correlations
loan_amount        due                         0.886538
```

```
                    monthly_emi                0.819133
                    collateral_value           0.809763
collateral_value    due                        0.803826
monthly_emi         average_monthly_balance    0.798077
loan_amount         repayment_amount           0.780483
monthly_emi         due                        0.718615
repayment_amount    average_monthly_balance    0.714145
vintage_in_months   target                     0.705141
collateral_value    monthly_emi                0.668101
dtype: float64
```

[50]:
```python
from statistics import mean
print("Average Monthly EMI: ",round(mean(data['monthly_emi']),2))
print("Average Repayment Amount: ",round(mean(data['repayment_amount']),2))
print("Average Loan Amount: ",round(mean(data['loan_amount']),2))
```

```
Average Monthly EMI:  16553.64
Average Repayment Amount:  150993.26
Average Loan Amount:  381630.43
```

[51]:
```python
#Univariate Analysis
plt.figure(figsize = [14,6])
data["loan_type"].value_counts().plot.barh(width = .8)
plt.title("Type of Loans", fontdict={"fontsize":15}, pad =20)
plt.show()
```

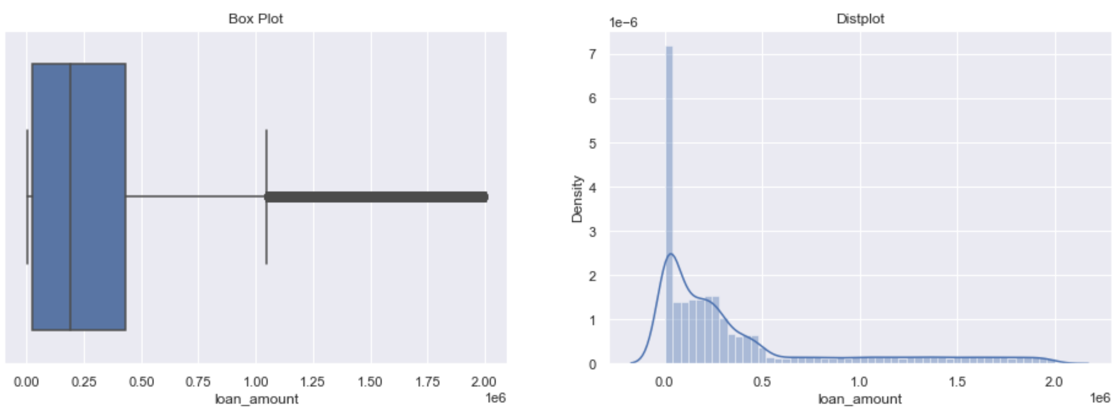


[52]:
```python
def Uni_Analysis_Numarical(dataframe, column):
    sns.set(style='darkgrid')
    plt.figure(figsize=(25, 5))

    plt.subplot(1, 3, 1)
```

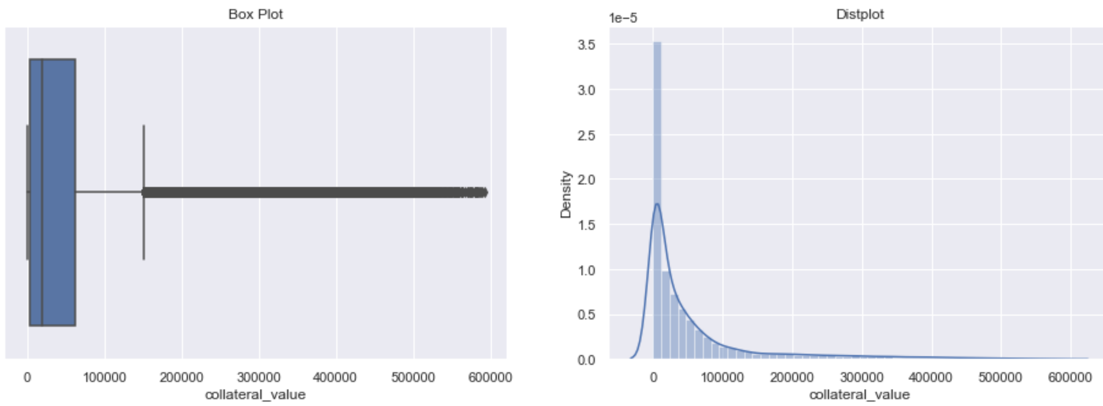
```
    sns.boxplot(data=dataframe, x=column, orient='v').set(title='Box Plot')

    plt.subplot(1, 3, 2)
    sns.distplot(dataframe[column].dropna()).set(title='Distplot')
    plt.show()
```

[53]:
```
#Distribution of the numerical features
for i in numeric_data:
    Uni_Analysis_Numarical(data,i)
```

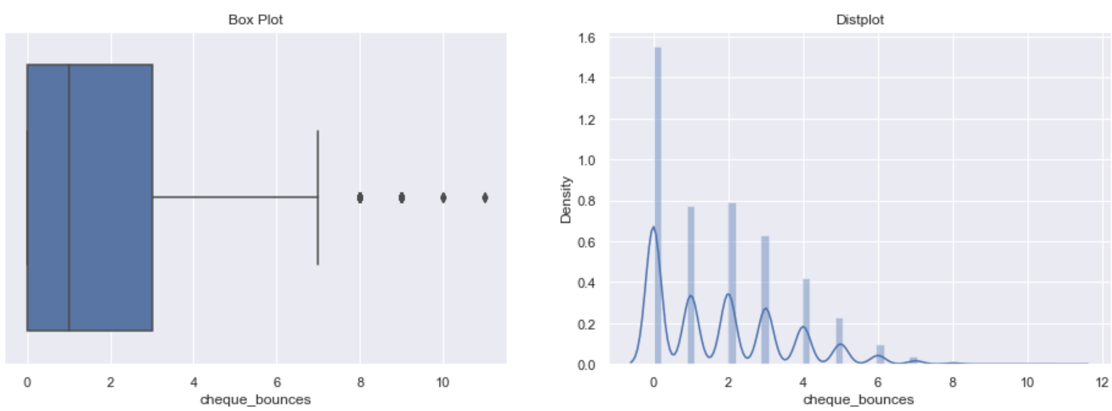
```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



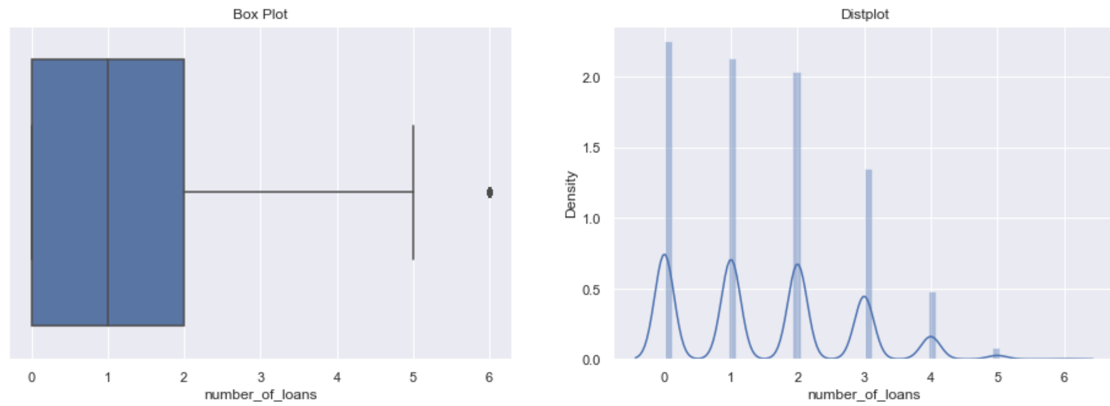

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



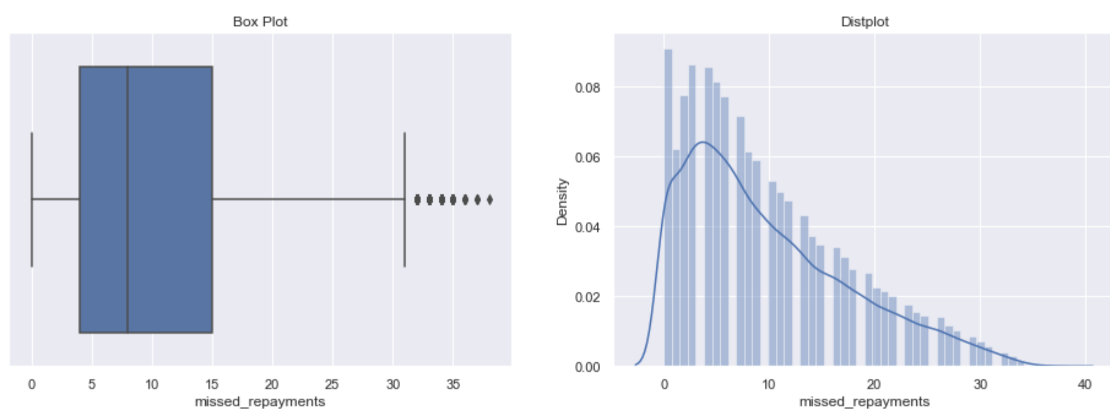

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



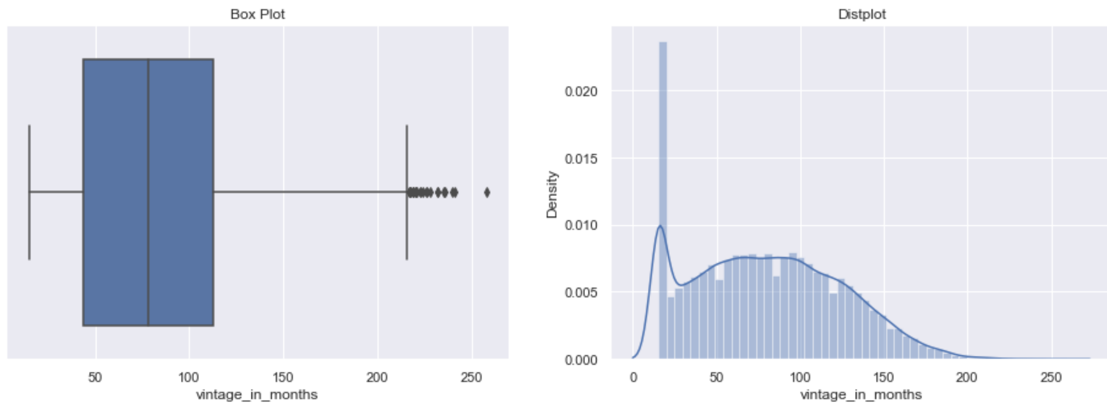

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



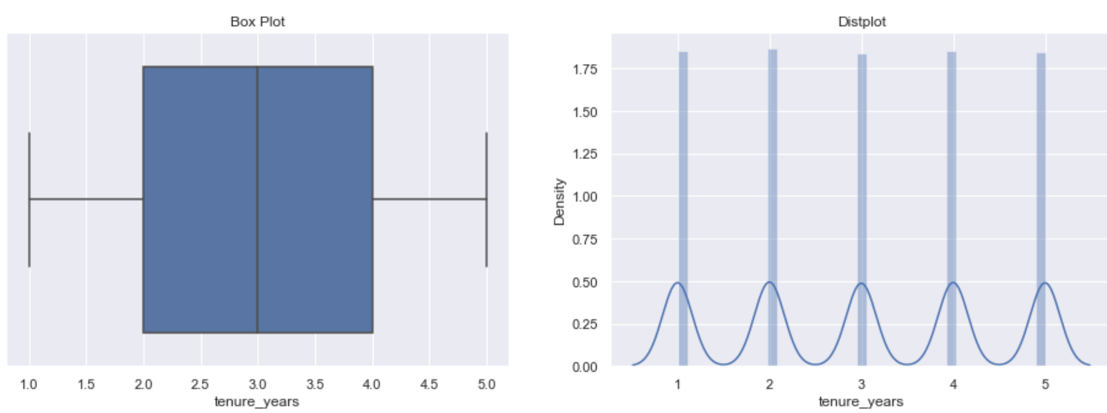

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



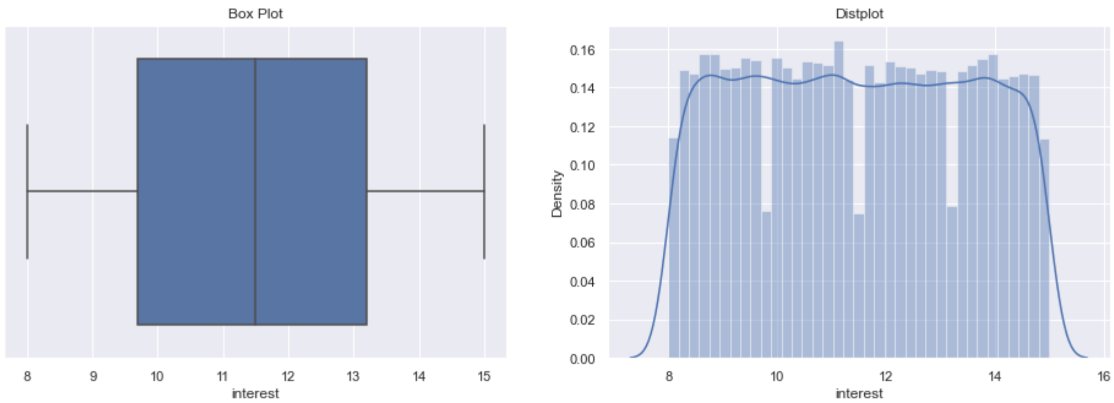

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



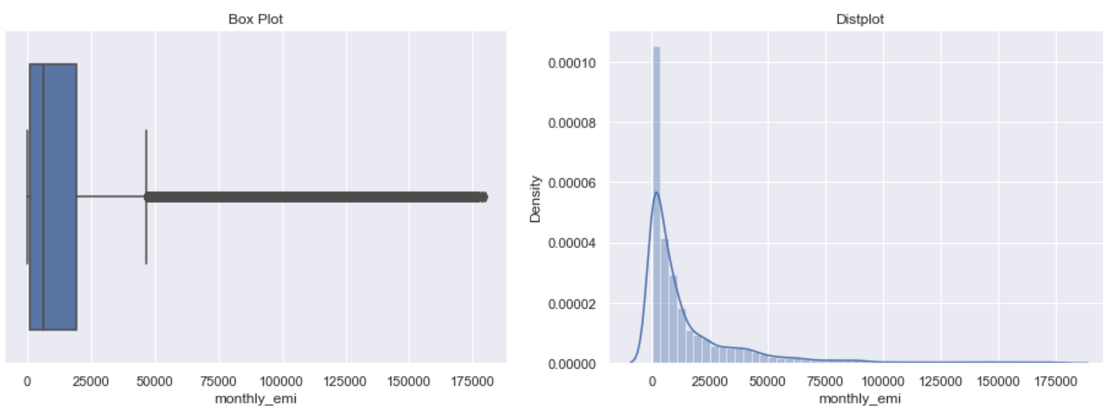

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



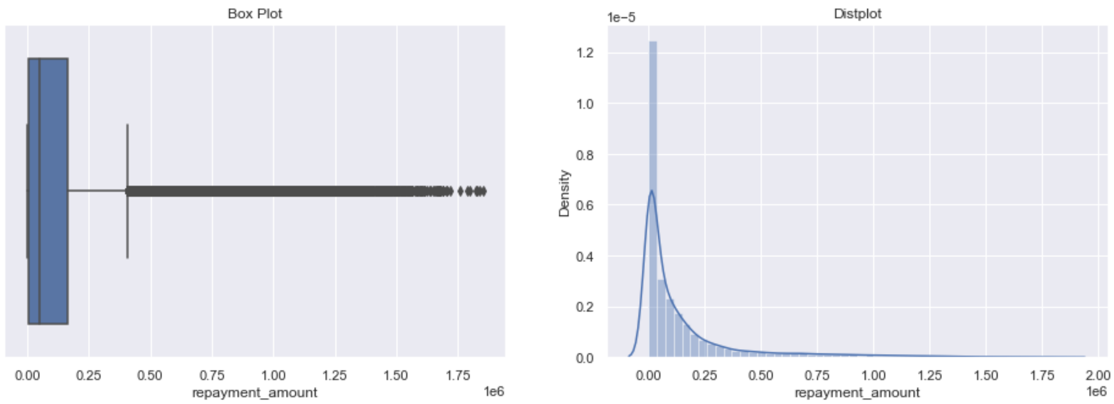

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



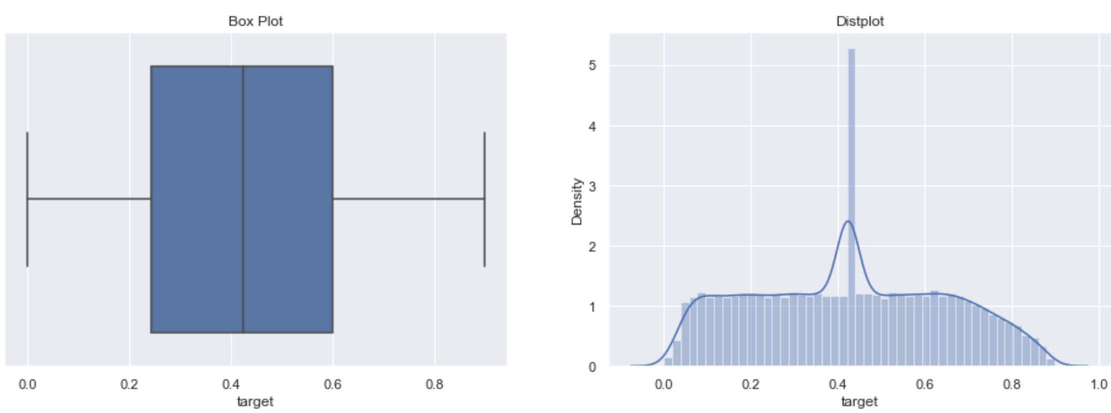

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



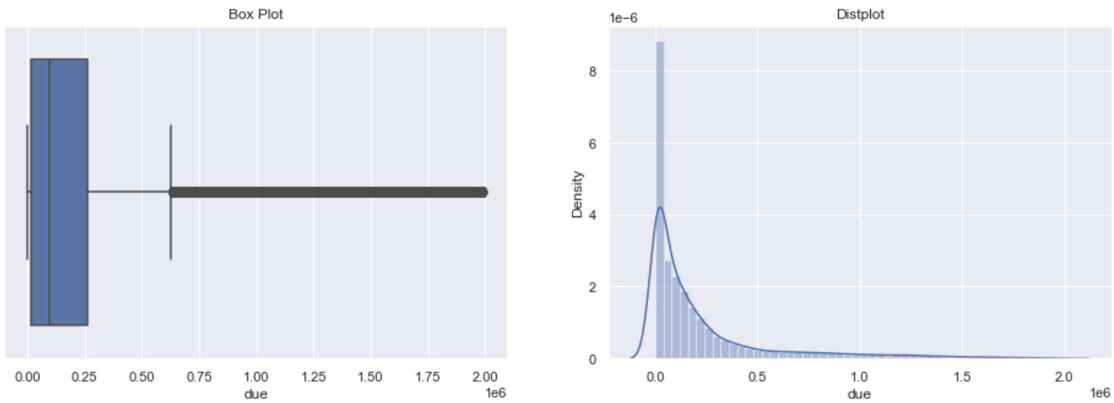

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



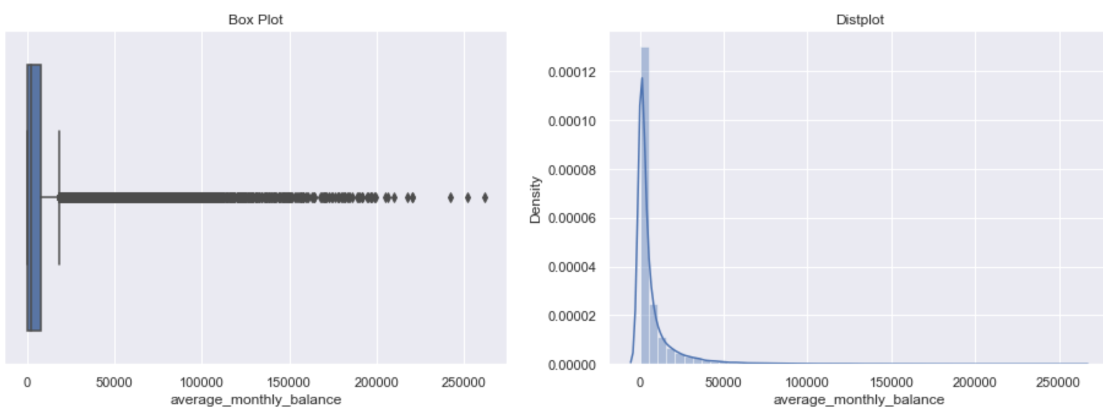

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

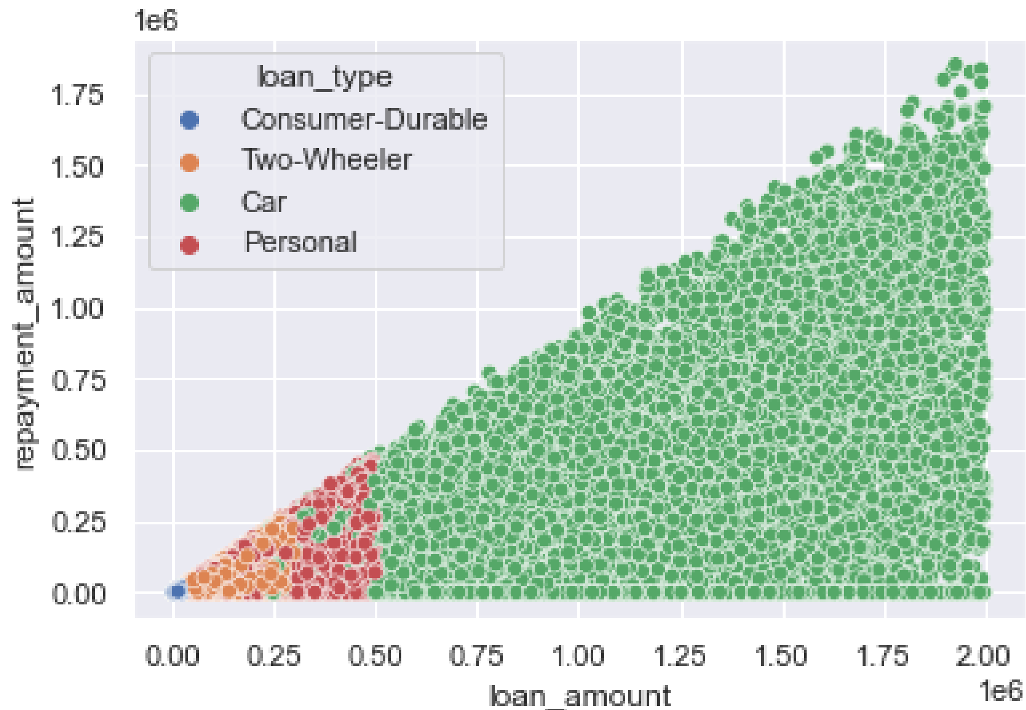



```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_core.py:1319: UserWarning:
Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\naren\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



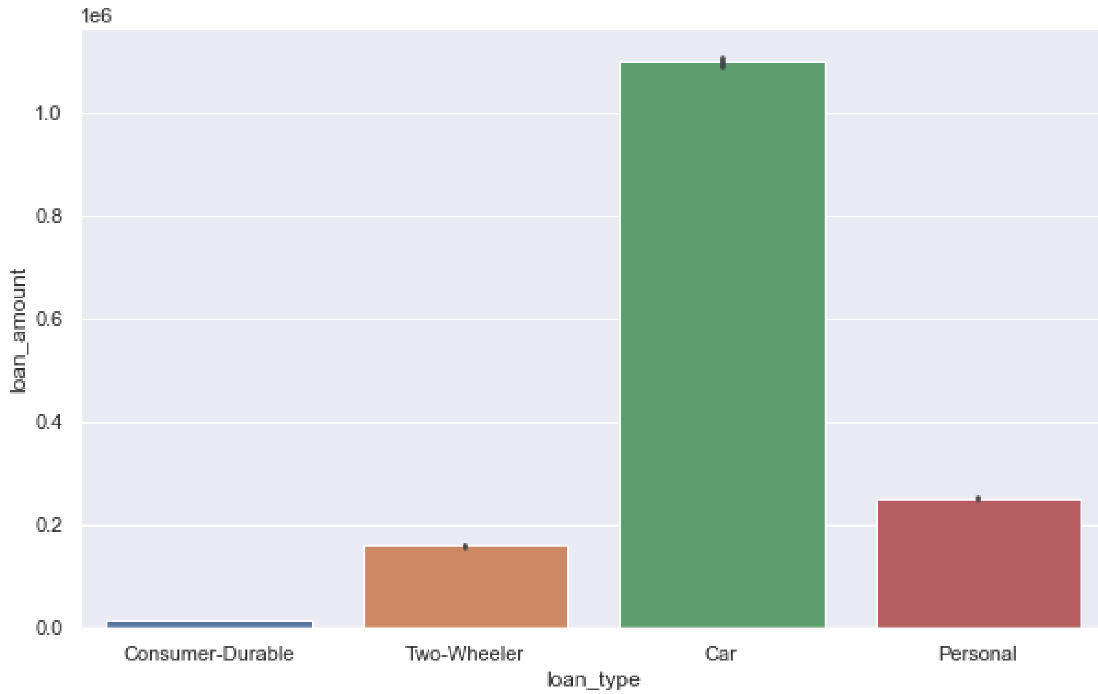
[54]:
```python
sns.scatterplot(x=data.loan_amount,
                y = data.repayment_amount,
                data=data,hue = 'loan_type')
```

[54]: `<AxesSubplot:xlabel='loan_amount', ylabel='repayment_amount'>`

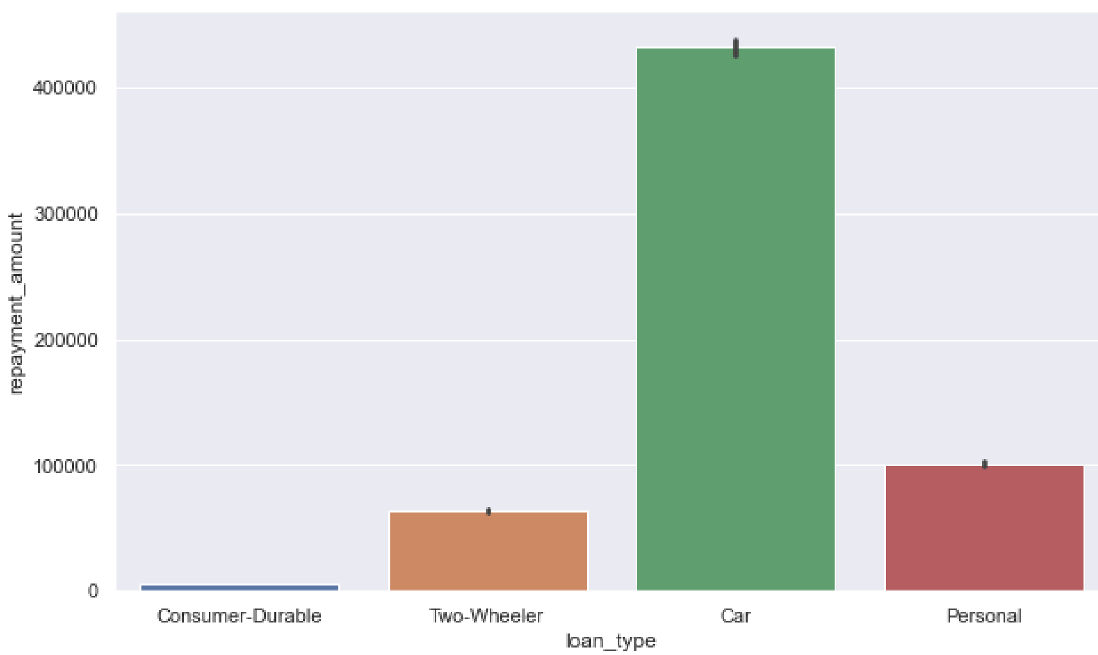


Observation: Though the number of Two-Wheeler loan is greater than others, Car loan comprised the maximum loan amount

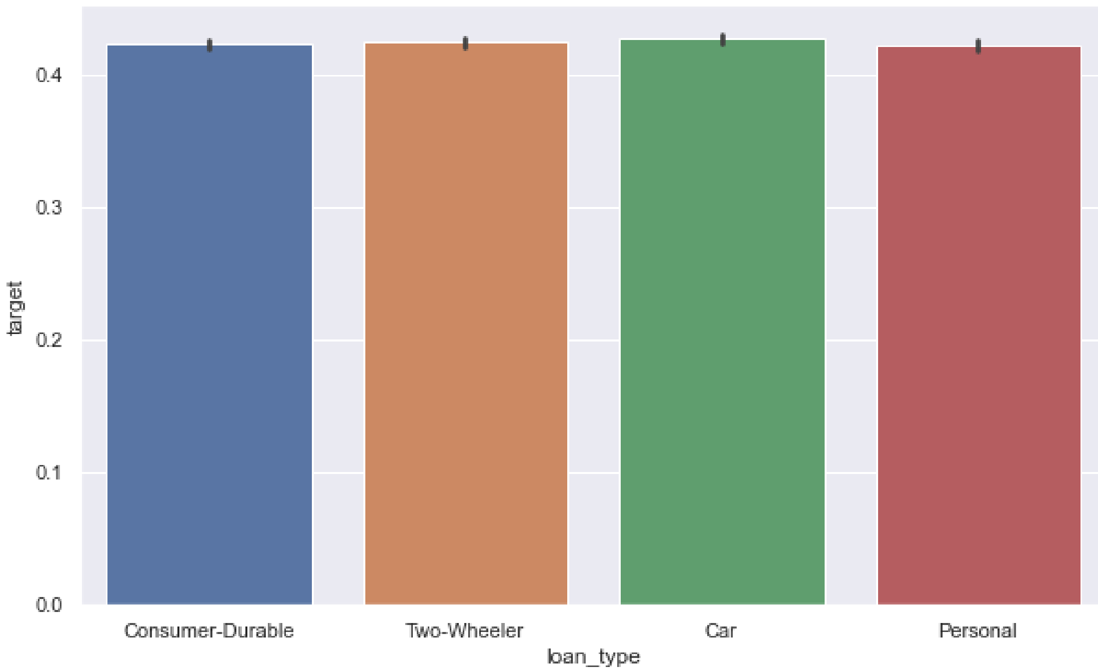
[55]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.loan_amount)
plt.show()
```



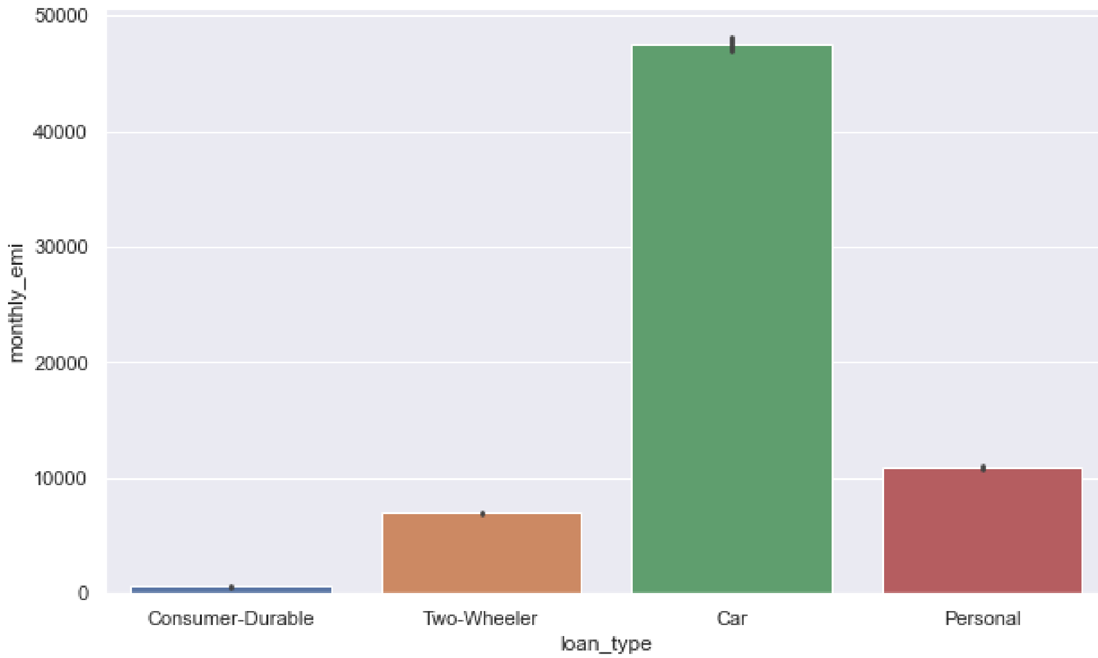
[56]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.repayment_amount)
plt.show()
```



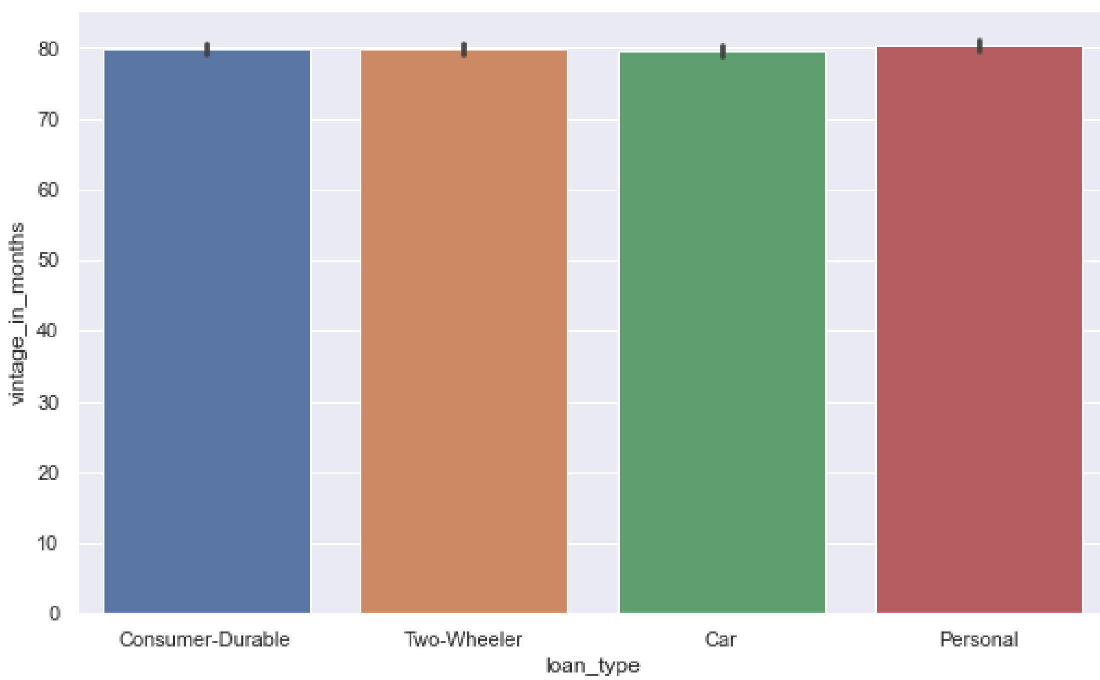
[57]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.target)
plt.show()
```



[58]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.monthly_emi)
plt.show()
```

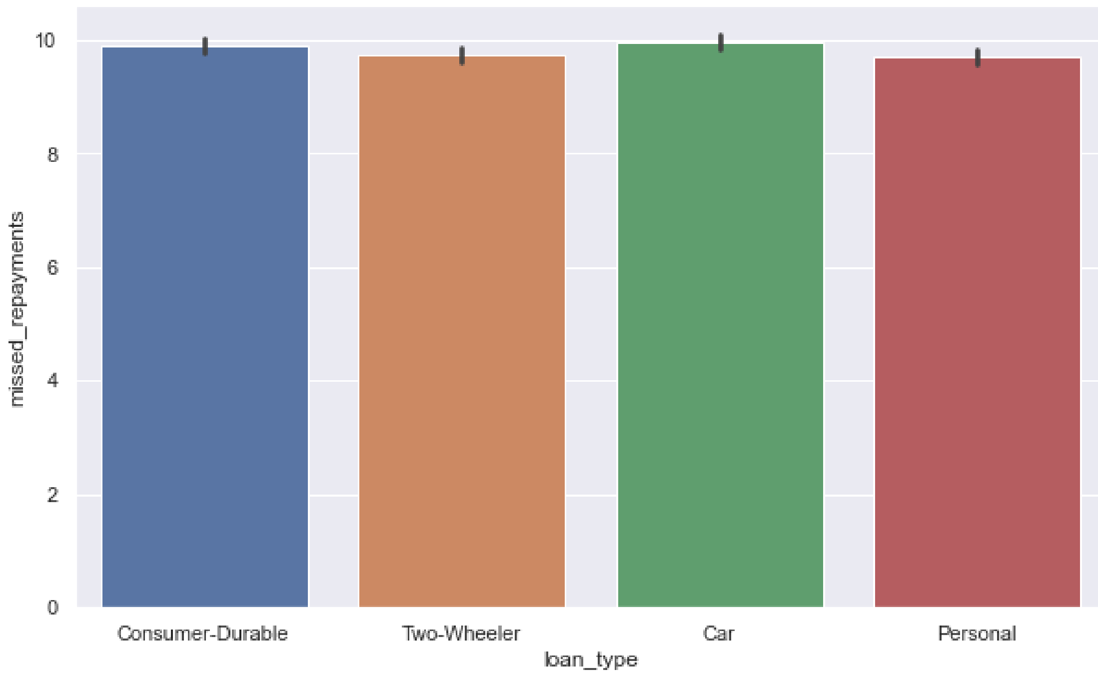


[59]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.vintage_in_months)
plt.show()
```



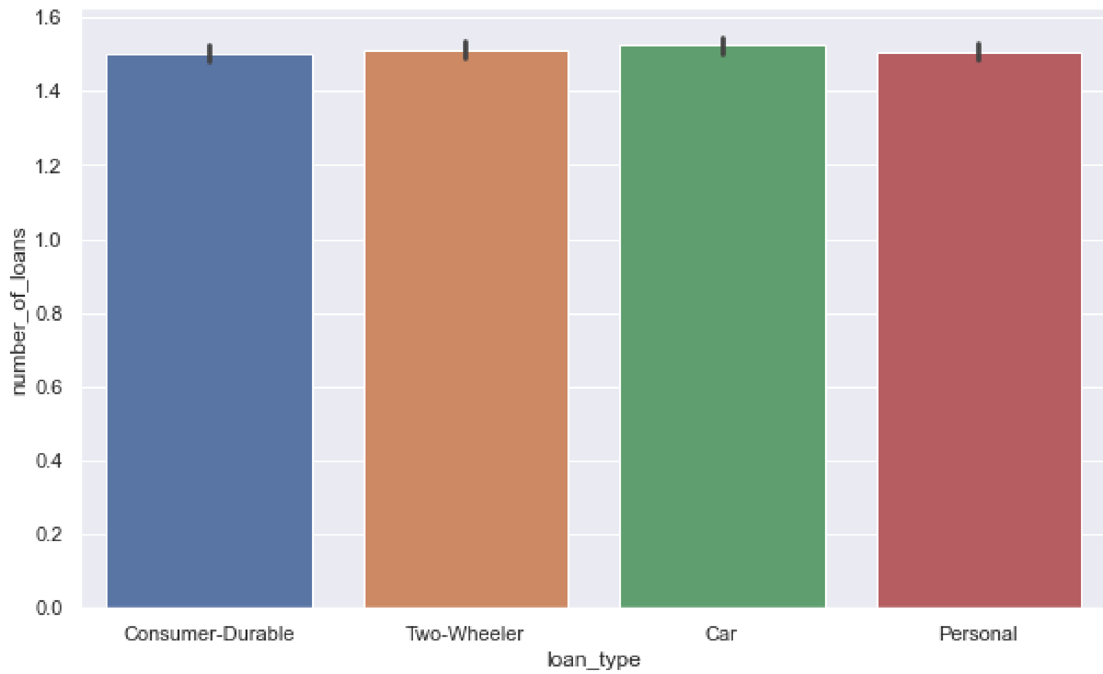
[60]:

```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.missed_repayments)
plt.show()
```

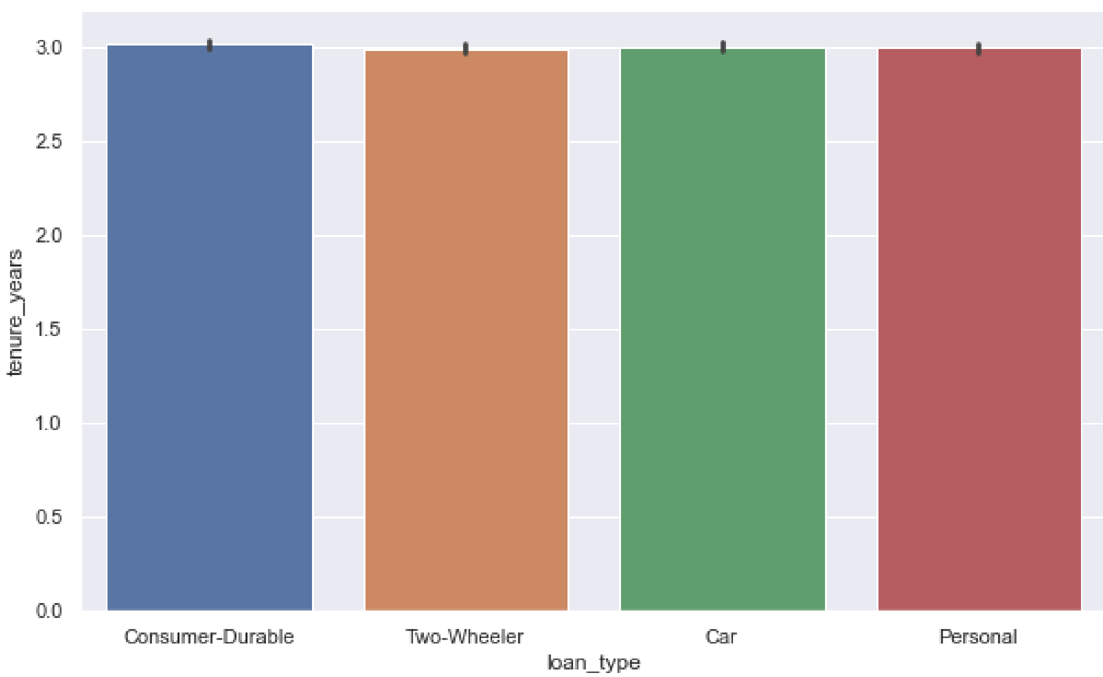


[61]:

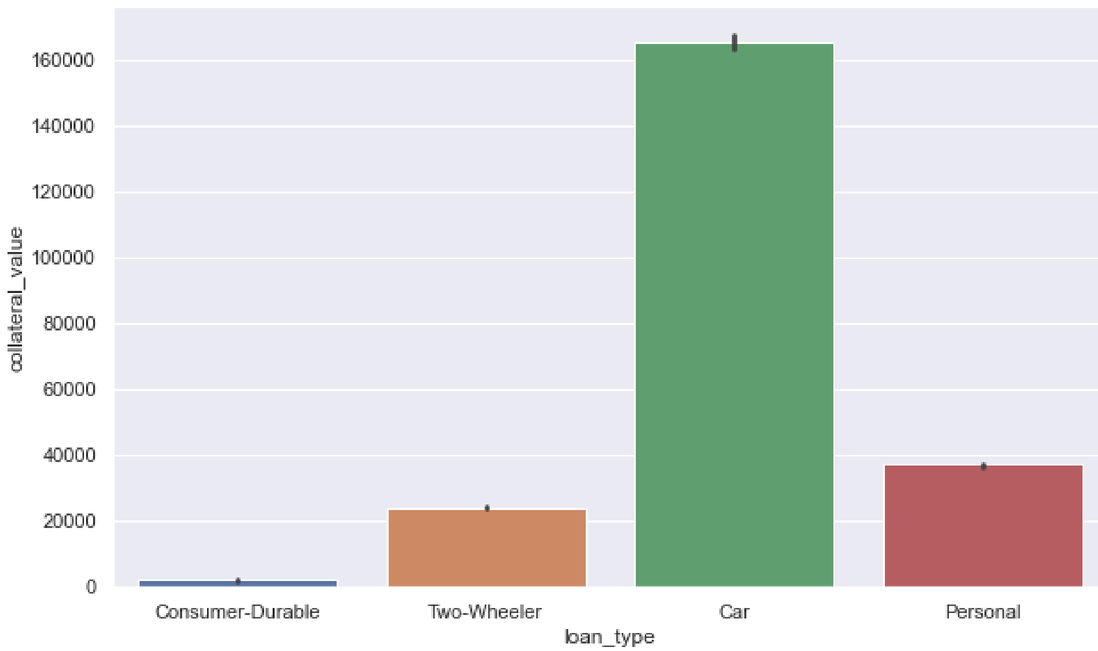
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.number_of_loans)
plt.show()
```



[62]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.tenure_years)
plt.show()
```



[63]:
```
plt.figure(figsize = [10,6])
sns.set(style='darkgrid')
sns.barplot(x = data.loan_type,y = data.collateral_value)
plt.show()
```

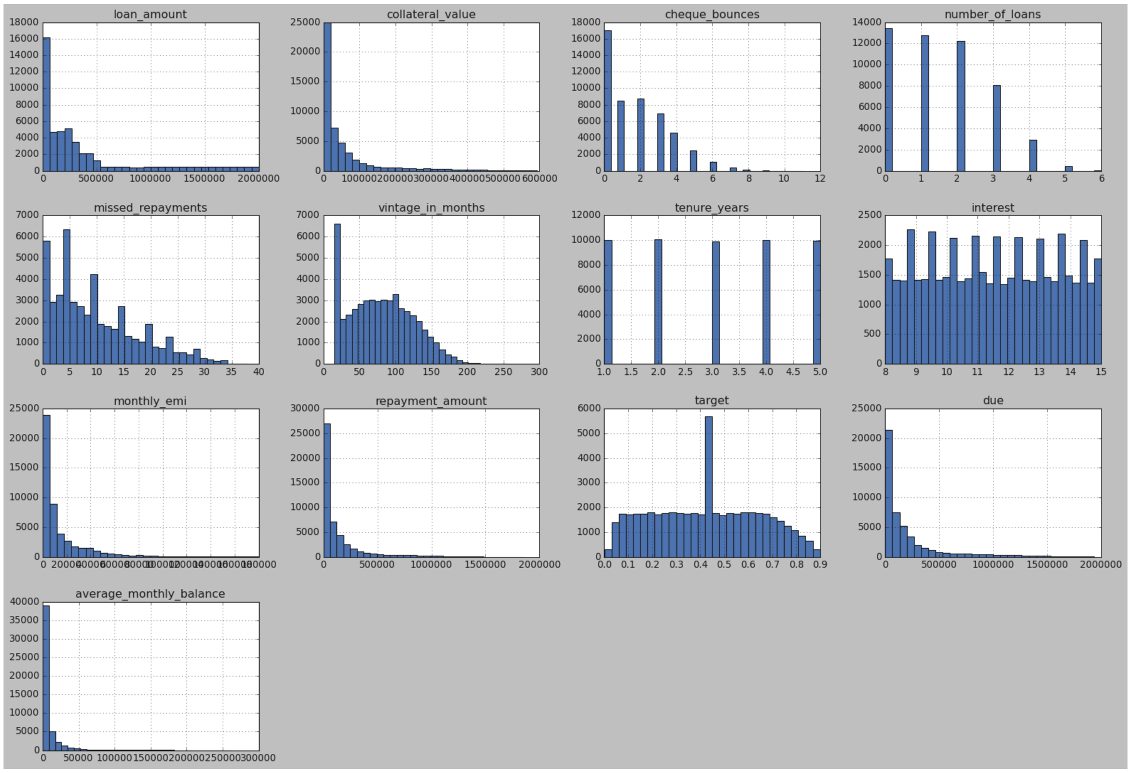


Observation: Repayment Amount or EMI amount of Car loan are way higher than other loan types

Data Preparation: variable transformation, feature engineering

[64]:
```
#separating numeric and categorical features
numeric_data = data.select_dtypes(include=[np.number])
categorical_data = data.select_dtypes(exclude=[np.number])
```

[65]:
```
#plt.figure(figsize = (24,24))
plt.style.use('classic')
data[numeric_data.columns].hist(bins=30, figsize = (24,16))
plt.show()
```

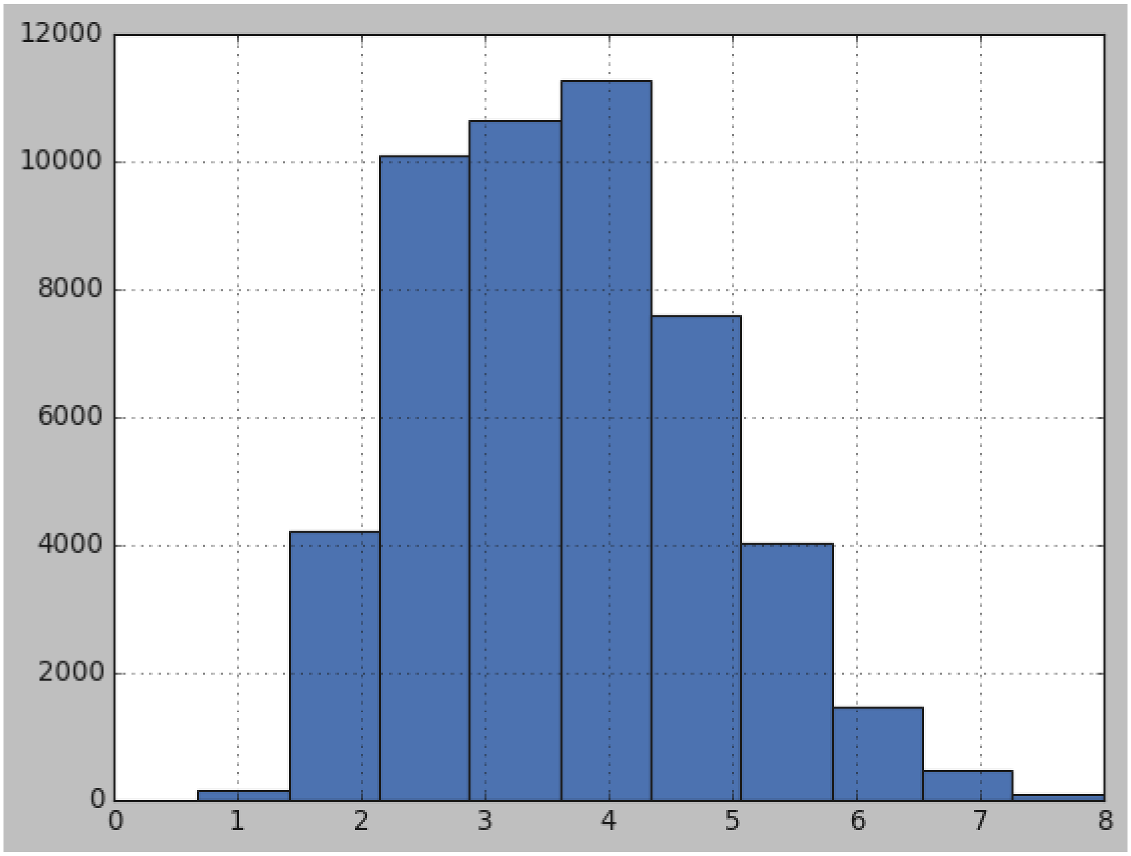


[66]:
```
numeric_data.columns
```

[66]:
```
Index(['loan_amount', 'collateral_value', 'cheque_bounces', 'number_of_loans',
       'missed_repayments', 'vintage_in_months', 'tenure_years', 'interest',
       'monthly_emi', 'repayment_amount', 'target', 'due',
       'average_monthly_balance'],
      dtype='object')
```

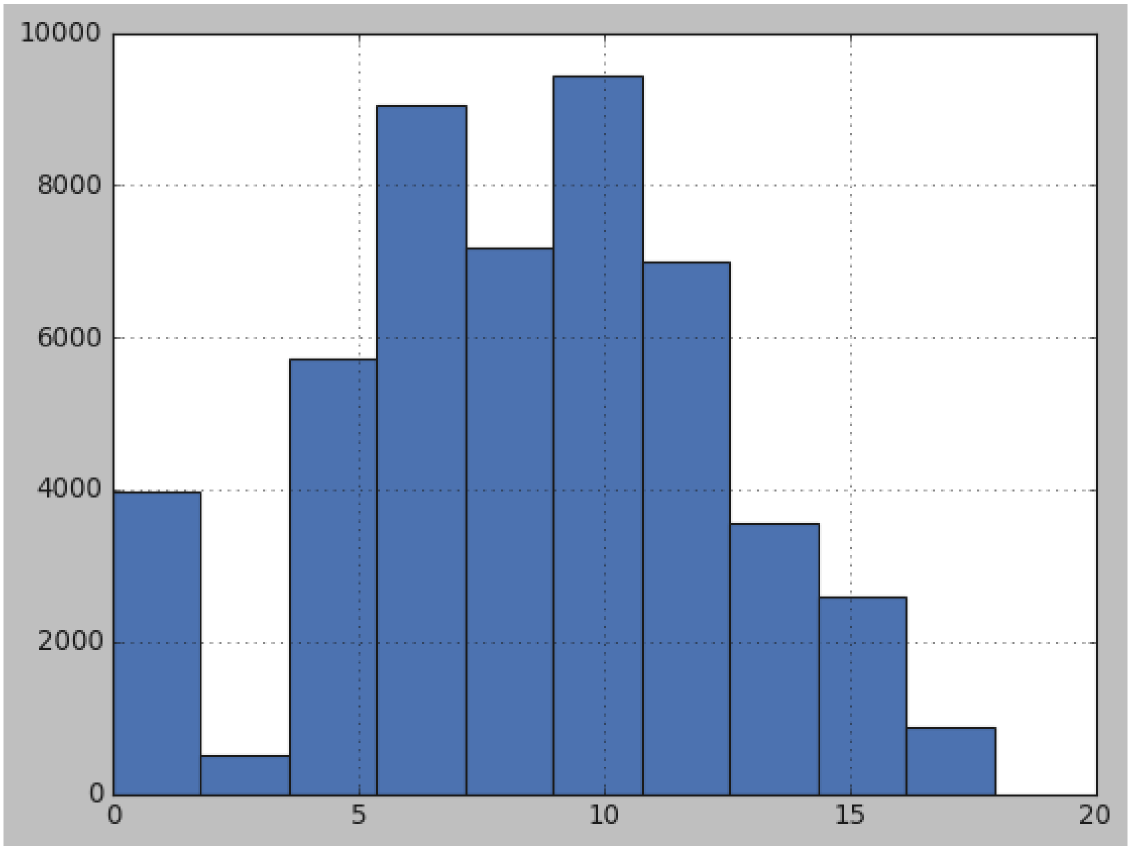
[67]:
```
(data['average_monthly_balance']**(1/6)).hist()
```

[67]:
```
<AxesSubplot:>
```




```
[68]: (data['repayment_amount']**(1/5)).hist()
      #fig = plt.figure()
      #(data['cheque_bounces']).hist()
```

```
[68]: <AxesSubplot:>
```



I used Power Transformation here with respect to the linear regression assumption that all the independent features should have normal distribution

[69]:
```
data['loan_amount'] = data['loan_amount']**(1/5)
```

[70]:
```
data['collateral_value'] = data['collateral_value']**(1/5)
```

[71]:
```
data['cheque_bounces'] = data['cheque_bounces']**(1/2)
```

[72]:
```
data['missed_repayments'] = data['missed_repayments']**(1/2)
```

[73]:
```
data['vintage_in_months'] = data['vintage_in_months']**(1/2)
```

[74]:
```
data['monthly_emi'] = data['monthly_emi']**(1/6)
```

[75]:
```
data['repayment_amount'] = data['repayment_amount']**(1/5)
```

[76]:
```
data['average_monthly_balance'] = data['average_monthly_balance']**(1/6)
```

[77]:
```
test_data['collateral_value'] = test_data['collateral_value']**(1/5)
test_data['cheque_bounces'] = test_data['cheque_bounces']**(1/2)
```

```
test_data['missed_repayments'] = test_data['missed_repayments']**(1/2)
test_data['vintage_in_months'] = test_data['vintage_in_months']**(1/2)
test_data['monthly_emi'] = test_data['monthly_emi']**(1/6)
test_data['repayment_amount'] = test_data['repayment_amount']**(1/5)
test_data['average_monthly_balance'] = test_data['average_monthly_balance']**(1/
 ↪8)
```

[ ]:

```

```

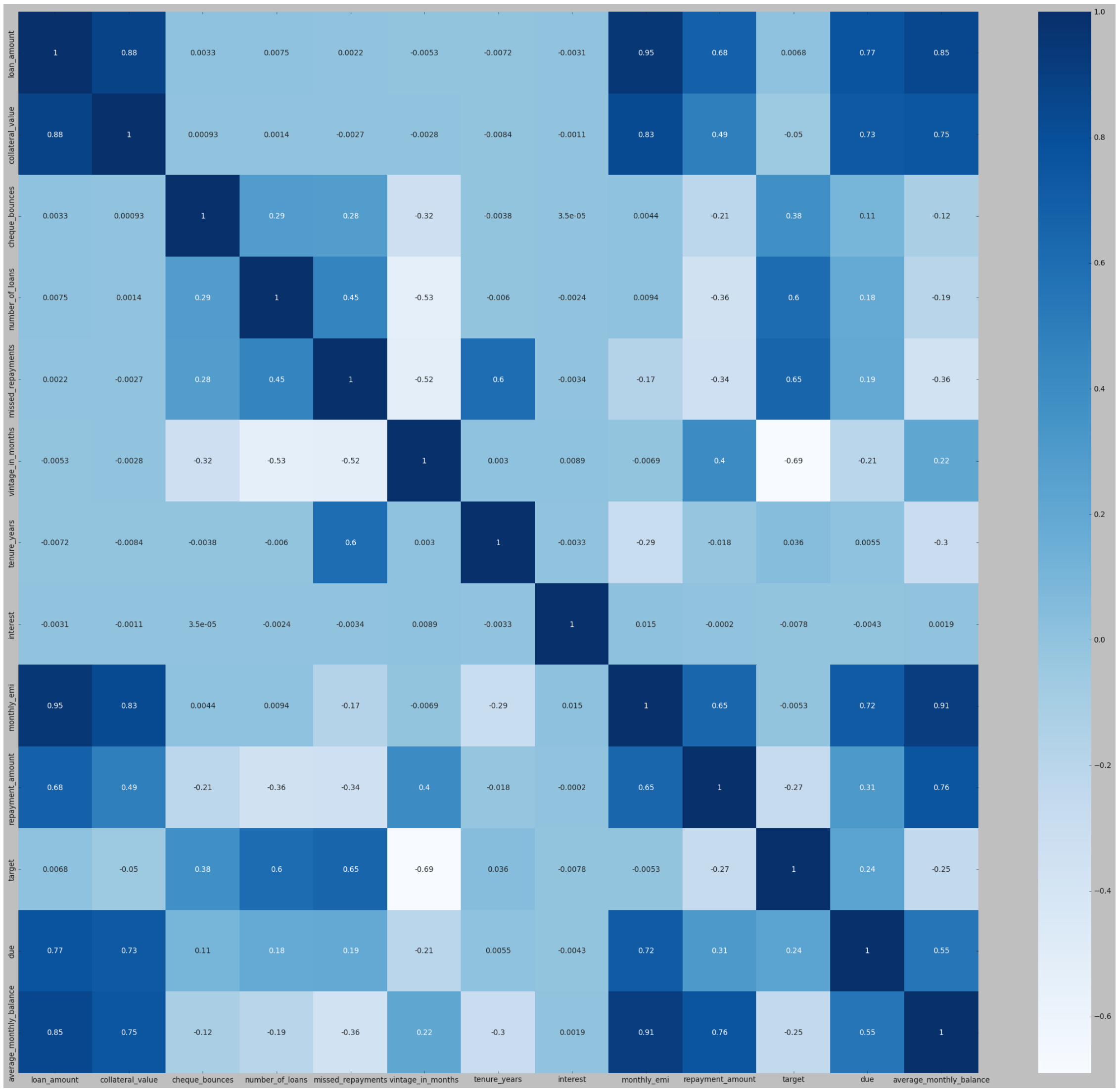
[ ]:

```

```

[78]:

```
#Correlation Plot: Heatmap
import seaborn as sns
plt.figure(figsize = (35,30))

corr = data.corr()
sns.heatmap(corr, cmap="Blues", annot=True)
```

[78]: <AxesSubplot:>



[79]:
```
categorical_data.columns
```

[79]:
```
Index(['loan_acc_num', 'customer_name', 'customer_address', 'loan_type',
       'disbursal_date', 'default_date'],
      dtype='object')
```

[80]:
```
curr_time = pd.to_datetime("now")
```

[81]:
```
#deriving new feature using date column
data['difference'] = (curr_time-data['default_date'])/np.timedelta64(1,'D')
```

[82]:
```
#deriving new feature using date column
test_data['difference'] = (curr_time-test_data['default_date'])/np.
 ↪timedelta64(1,'D')
```

[83]:
```
#dropping unnecessary columns for model building
data = data.drop(['disbursal_date', 'default_date','loan_acc_num',␣
↪'customer_name', 'customer_address'],1)
```

[84]:
```
#dropping unnecessary columns for model building
test_data = test_data.drop(['disbursal_date', 'default_date','loan_acc_num',␣
↪'customer_name', 'customer_address'],1)
```

[85]:
```
#encoding
encoded = pd.get_dummies(data['loan_type'],drop_first=True)
```

[86]:
```
#encoding
test_encoded = pd.get_dummies(test_data['loan_type'],drop_first=True)
```

[87]:
```
data = data.drop(['loan_type'],axis=1)
```

[88]:
```
test_data = test_data.drop(['loan_type'],axis=1)
```

[89]:
```
data = pd.concat([data,encoded],axis=1)
```

[90]:
```
test_data = pd.concat([test_data,test_encoded],axis=1)
```

[91]:
```
#downloadng the prepared dataset for PyCaret
data.to_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\pycaret_test.csv",index=False)
```

[92]:
```
#train-test split
from sklearn.model_selection import train_test_split
X = data.drop(columns=['target'])
y = data [['target']]
# Choose any random state
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪3,random_state=42)
```

[93]:
```
#Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[94]:
```
unseen = sc.fit_transform(test_data)
```

[95]:
```
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

Using Recusrive Feature Elimination

[96]:
```
# Running RFE with the output number of the variable equal to 10
lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, n_features_to_select=10)             # running RFE
rfe = rfe.fit(X_train, y_train)
```

[97]:
```
from sklearn.linear_model import *
from  sklearn import metrics
#Rsquared on test set
y_pred_lr = rfe.predict(X_test)
metrics.r2_score(y_test, y_pred_lr)
```

[97]:
```
0.7756349800595067
```

Using Multiple Linear Regression

[98]:
```
# Representing LinearRegression as lr(Creating LinearRegression Object)
lm = LinearRegression()

lm.fit(X_train, y_train)
#Rsquared on test set
y_pred_lr = lm.predict(X_test)
metrics.r2_score(y_test, y_pred_lr)
```

[98]:
```
0.7811962854570624
```

Using Random Forest Regressor

[99]:
```
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor(bootstrap= True,n_estimators = 200,␣
 ↪random_state = 42, max_depth=4,max_features=None,min_samples_leaf=␣
 ↪2,min_samples_split= 4)
rf_regressor.fit(X_train, y_train)
```

```
<ipython-input-99-066823086f1c>:6: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  rf_regressor.fit(X_train, y_train)
```

[99]:
```
RandomForestRegressor(max_depth=4, max_features=None, min_samples_leaf=2,
                      min_samples_split=4, n_estimators=200, random_state=42)
```

[100]:
```
y_pred_train = rf_regressor.predict(X_train)
print(r2_score(y_train, y_pred_train))
```

0.7245706459113583

[101]:
```
y_pred_test = rf_regressor.predict(X_test)
print(r2_score(y_test, y_pred_test))
```

0.7273636647483164

Using Gradient Boosting Regressor

[102]:
```
from sklearn.ensemble import GradientBoostingRegressor
# Hyperparameters for GradientBoostingRegressor
#
gbr_params = {'n_estimators': 500,
          'max_depth': 4,
          'min_samples_split': 4,
          'learning_rate': 0.01,
          'loss': 'ls'}
#
# Create an instance of gradient boosting regressor
#
gbr = GradientBoostingRegressor(**gbr_params)
#
# Fit the model
#
gbr.fit(X_train, y_train)

y_pred=gbr.predict(X_test)
#
# Print Coefficient of determination R^2
#
print("R squared: %.3f" % gbr.score(X_test, y_test))
```

```
C:\Users\naren\anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

R squared: 0.901

[103]:
```
y_pred_train=gbr.predict(X_train)
#
# Print Coefficient of determination R^2
print("R_squared in train set: ",r2_score(y_train, y_pred_train))
```

R_squared in train set: 0.9040959918040925

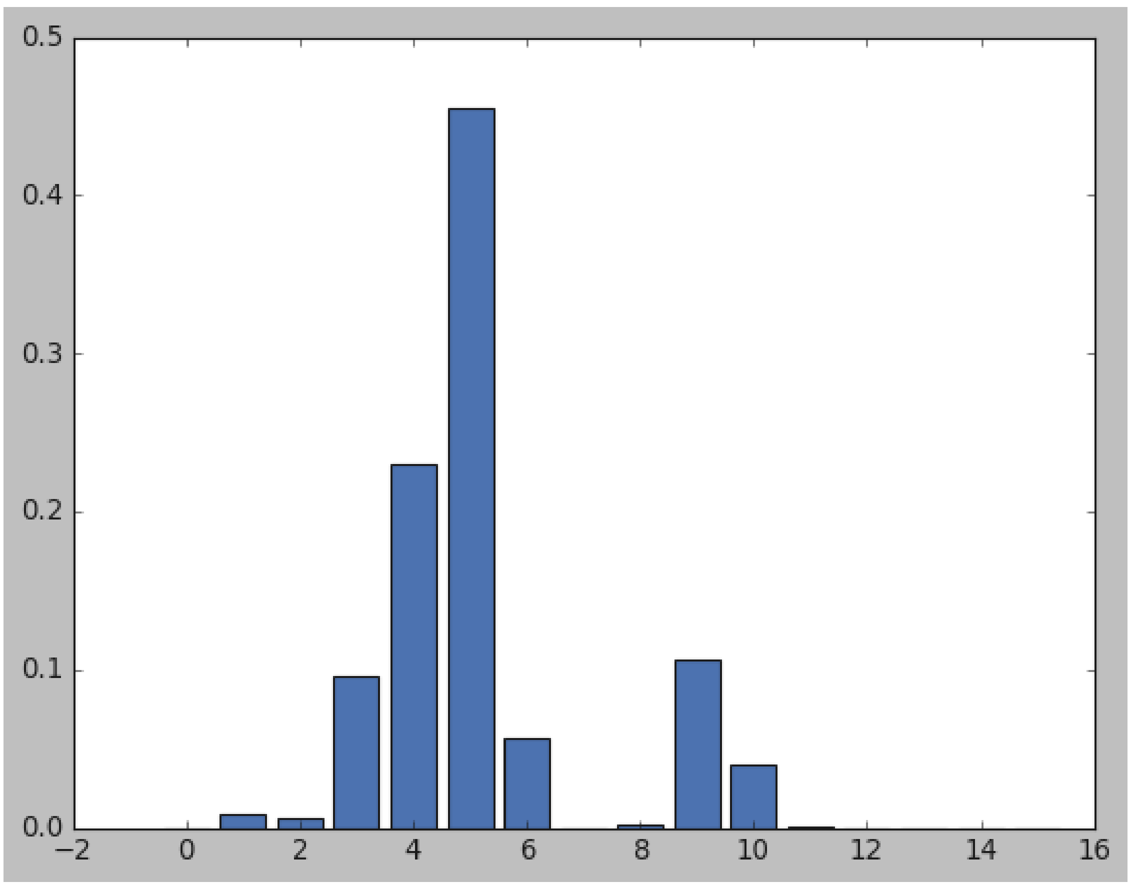
[104]:
```
y_pred_test=gbr.predict(X_test)
#
```

```
# Print Coefficient of determination R^2
print("R_squared in test set: ",r2_score(y_test, y_pred_test))
```

```
R_squared in test set:  0.9009826242325177
```

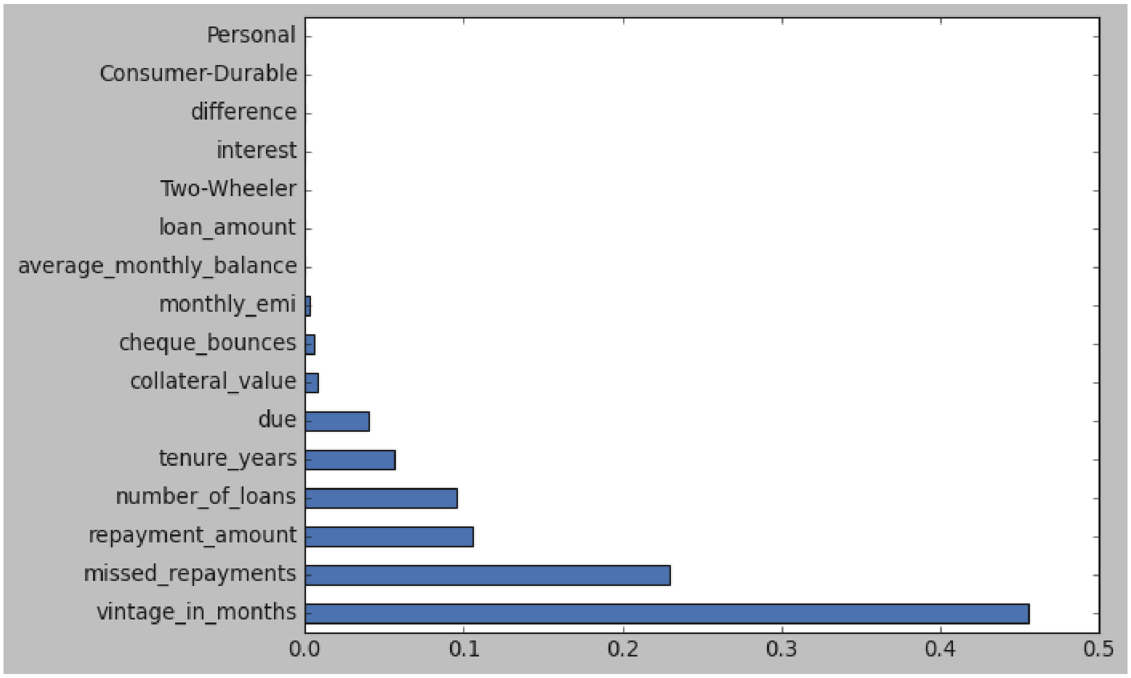
[105]:
```
# plot feature importance
# feature importance
print(gbr.feature_importances_)
# plot
plt.bar(range(len(gbr.feature_importances_)), gbr.feature_importances_)
plt.show()
```

```
[7.42212679e-20 8.27642038e-03 6.29420805e-03 9.57308377e-02
 2.29346406e-01 4.55337925e-01 5.66951780e-02 0.00000000e+00
 2.61179097e-03 1.05657884e-01 3.97555164e-02 2.93833105e-04
 0.00000000e+00 0.00000000e+00 0.00000000e+00 1.55474123e-20]
```



[106]:
```
feat_importances = pd.Series(gbr.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

[106]: <AxesSubplot:>



Pycaret

[107]:
```
dataset = pd.read_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
↪Assignment\\pycaret_test.csv")
data_ = dataset.sample(frac=0.8, random_state=786).reset_index(drop=True)
data_unseen = dataset.drop(data_.index).reset_index(drop=True)

print('Data for Modeling: ' + str(data_.shape))
print('Unseen Data For Predictions: ' + str(data_unseen.shape))
```

```
Data for Modeling: (39911, 17)
Unseen Data For Predictions: (9978, 17)
```

So, the highly important features are 1. repayment_amount 2. missed_repayments 3. due 4. vintage_in_months 5. tenure_years

[ ]:
```

```

Finalized the Catboost Regressor with 99.55% Rsqaured on test data

Further Experiments

Using XGBoost Regressor

[122]:
```
from numpy import absolute
from pandas import read_csv
```

```python
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from xgboost import XGBRegressor
```

[123]:
```python
# define model
model = XGBRegressor()
# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='r2', cv=cv,␣
↪n_jobs=-1)
# force scores to be positive
scores = absolute(scores)
print('R_squared: %.3f (%.3f)' % (scores.mean(), scores.std()) )
```

```
R_squared: 0.977 (0.001)
```

[124]:
```python
# evaluate model
scores = cross_val_score(model, X_test, y_test, scoring='r2', cv=cv, n_jobs=-1)
# force scores to be positive
scores = absolute(scores)
print('R_squared: %.3f (%.3f)' % (scores.mean(), scores.std()) )
```

```
R_squared: 0.970 (0.002)
```

[125]:
```python
from xgboost import XGBRegressor
# define model
RegModel=XGBRegressor(max_depth=4, learning_rate=0.01, n_estimators=500,␣
↪objective='reg:linear', booster='gbtree')
```

[126]:
```python
XGB=RegModel.fit(X_train,y_train)
prediction=XGB.predict(X_train)
print("train score: ", r2_score(y_train, prediction))
y_pred_test=XGB.predict(X_test)
print("test score: ", r2_score(y_test, y_pred_test))
```

```
C:\Users\naren\anaconda3\lib\site-packages\xgboost\core.py:160: UserWarning:
[19:39:30] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-
windows\src\objective\regression_obj.cu:209: reg:linear is now deprecated in
favor of reg:squarederror.
  warnings.warn(smsg, UserWarning)
```

```
train score:  0.8997984846286474
test score:  0.8964291662716752
```

Using Adaboost Regressor

[127]:
```python
from sklearn.ensemble import AdaBoostRegressor
ada_reg = AdaBoostRegressor(n_estimators=500,learning_rate=0.1)
Adaboost=ada_reg.fit(X_train,y_train)
prediction=Adaboost.predict(X_train)
print("train score: ", r2_score(y_train, prediction))
y_pred_test=Adaboost.predict(X_test)
print("test score: ", r2_score(y_test, y_pred_test))
```

```
C:\Users\naren\anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

```
train score:  0.7709186751471815
test score:  0.7755526672680789
```

Using ElasticNet : Hybrid Regularized Model

[128]:
```python
from sklearn.datasets import load_boston
from sklearn.linear_model import ElasticNet,ElasticNetCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
```

[129]:
```python
alphas = [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1]
```

[130]:
```python
for a in alphas:
    model = ElasticNet(alpha=a).fit(X_train,y_train)
    score = model.score(X_train,y_train)
    pred_y = model.predict(X_test)
    mse = mean_squared_error(y_test, pred_y)
    print("Alpha:{0:.4f}, R2:{1:.2f}, MSE:{2:.2f}, RMSE:{3:.2f}"
        .format(a, score, mse, np.sqrt(mse)))
```

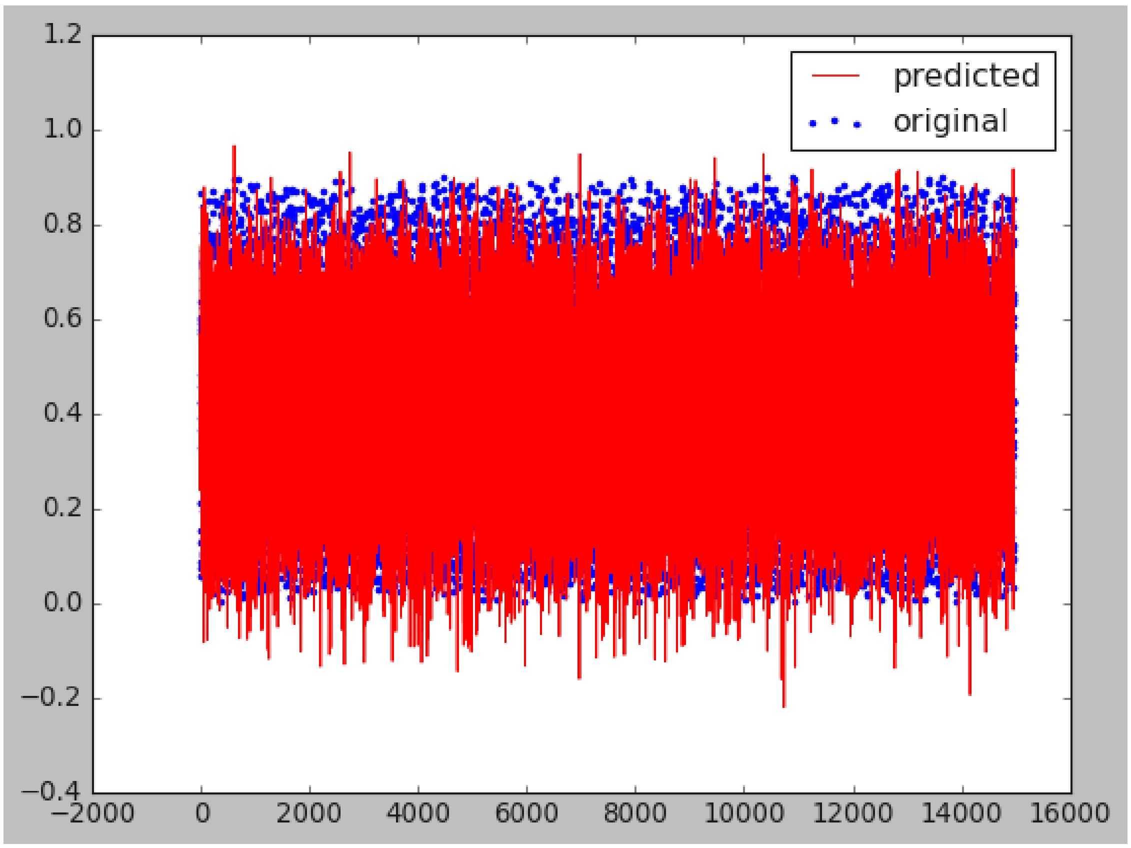
```
Alpha:0.0001, R2:0.78, MSE:0.01, RMSE:0.10
Alpha:0.0010, R2:0.77, MSE:0.01, RMSE:0.10
Alpha:0.0100, R2:0.75, MSE:0.01, RMSE:0.11
Alpha:0.1000, R2:0.55, MSE:0.02, RMSE:0.15
Alpha:0.3000, R2:0.01, MSE:0.05, RMSE:0.22
Alpha:0.5000, R2:0.00, MSE:0.05, RMSE:0.22
Alpha:0.7000, R2:0.00, MSE:0.05, RMSE:0.22
Alpha:1.0000, R2:0.00, MSE:0.05, RMSE:0.22
```

[131]:
```python
elastic=ElasticNet(alpha=0.001).fit(X_train, y_train)
y_pred = elastic.predict(X_test)
score = elastic.score(X_test, y_test)
```

```
mse = mean_squared_error(y_test, y_pred)
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}"
      .format(score, mse, np.sqrt(mse)))
```

```
R2:0.780, MSE:0.01, RMSE:0.10
```

[132]:
```
x_ax = range(len(X_test))
plt.scatter(x_ax, y_test, s=5, color="blue", label="original")
plt.plot(x_ax, y_pred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



[133]:
```
elastic_cv=ElasticNetCV(alphas=alphas, cv=5)
model = elastic_cv.fit(X_train, y_train)
print("Alpha: ",model.alpha_)
print("Intercept: ",model.intercept_)
```

```
C:\Users\naren\anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
```

```
  return f(*args, **kwargs)

Alpha:  0.0001
Intercept:  0.4241654676370259
```

[134]:
```
y_pred = model.predict(X_test)
score = model.score(X_test, y_test)
mse = mean_squared_error(y_test, y_pred)
print("R2:{0:.3f}, MSE:{1:.2f}, RMSE:{2:.2f}"
      .format(score, mse, np.sqrt(mse)))
```

```
R2:0.781, MSE:0.01, RMSE:0.10
```

Using LightGBM

[135]:
```
plt.style.use('ggplot')
import lightgbm as ltb
```

[136]:
```
model = ltb.LGBMRegressor()
model.fit(X_train, y_train)
print(); print(model)
```

```
C:\Users\naren\anaconda3\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.004639 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2134
[LightGBM] [Info] Number of data points in the train set: 34922, number of used
features: 16
[LightGBM] [Info] Start training from score 0.424165

LGBMRegressor()
```

[137]:
```
y_pred = model.predict(X_test)
print("Rsqaured on test data: ",metrics.r2_score(y_test, y_pred))
```

```
Rsqaured on test data:  0.965125001305115
```

[ ]:
```

```

Model Evaluation: Hyperparameter Tuning

[ ]:
```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from xgboost import XGBRegressor
```

```
params = { 'max_depth': [3,4,5],
           'learning_rate': [0.01, 0.05, 0.1,0.25,0.5,0.015,1],
           'n_estimators': [100, 500, 1000],
           'colsample_bytree': [0.3, 0.7]}

xgbr = xgb.XGBRegressor(seed = 20)

clf = GridSearchCV(estimator=xgbr,
                   param_grid=params,
                   scoring='r2',
                   verbose=1)
clf.fit(X_train, y_train)
print("Best parameters:", clf.best_params_)
```

[139]:
```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
# define the model with best resulted parameters
RegModel=XGBRegressor(colsample_bytree= 0.7, max_depth=3, learning_rate=0.5,␣
 ↪n_estimators=1000, objective='reg:linear', booster='gbtree')
xgbr = xgb.XGBRegressor(seed = 20)
XGB=RegModel.fit(X_train,y_train)
prediction=XGB.predict(X_train)
print("train score: ", r2_score(y_train, prediction))
y_pred_test=XGB.predict(X_test)
print("test score: ", r2_score(y_test, y_pred_test))
```

```
C:\Users\naren\anaconda3\lib\site-packages\xgboost\core.py:160: UserWarning:
[19:57:20] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-
windows\src\objective\regression_obj.cu:209: reg:linear is now deprecated in
favor of reg:squarederror.
  warnings.warn(smsg, UserWarning)
```

```
train score:  0.9959192885034708
test score:  0.9897083070417368
```

[140]:
```
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
# define the model with best resulted parameters
RegModel=XGBRegressor(colsample_bytree= 0.7, max_depth=3, learning_rate=0.5,␣
 ↪n_estimators=1000, objective='reg:linear', booster='gbtree')
xgbr = xgb.XGBRegressor(seed = 20)
XGB=RegModel.fit(X_train,y_train)
```

```
prediction=XGB.predict(X_train)
print("train score: ", r2_score(y_train, prediction))
y_pred_test=XGB.predict(X_test)
print("test score: ", r2_score(y_test, y_pred_test))
```

```
C:\Users\naren\anaconda3\lib\site-packages\xgboost\core.py:160: UserWarning:
[19:57:24] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-
windows\src\objective\regression_obj.cu:209: reg:linear is now deprecated in
favor of reg:squarederror.
  warnings.warn(smsg, UserWarning)
```

```
train score:  0.9959192885034708
test score:  0.9897083070417368
```

So, we can finalize fine tuned XGBoost as well as it is giving us 99.5% Rsqaured on test data

Prediction on Unseen Data

[141]:
```
data.columns
```

[141]:
```
Index(['loan_amount', 'collateral_value', 'cheque_bounces', 'number_of_loans',
       'missed_repayments', 'vintage_in_months', 'tenure_years', 'interest',
       'monthly_emi', 'repayment_amount', 'target', 'due',
       'average_monthly_balance', 'difference', 'Consumer-Durable', 'Personal',
       'Two-Wheeler'],
      dtype='object')
```

[142]:
```
test_data.columns
```

[142]:
```
Index(['loan_amount', 'collateral_value', 'cheque_bounces', 'number_of_loans',
       'missed_repayments', 'vintage_in_months', 'tenure_years', 'interest',
       'monthly_emi', 'repayment_amount', 'due', 'average_monthly_balance',
       'difference', 'Consumer-Durable', 'Personal', 'Two-Wheeler'],
      dtype='object')
```

[143]:
```
test_data = sc.fit_transform(test_data)
```

[144]:
```
test_data = pd.DataFrame(test_data)
```

[145]:
```
test_data.columns = ['loan_amount', 'collateral_value', 'cheque_bounces',␣
 ↪'number_of_loans',
       'missed_repayments', 'vintage_in_months', 'tenure_years', 'interest',
       'monthly_emi', 'repayment_amount', 'due',
       'average_monthly_balance', 'difference', 'Consumer-Durable', 'Personal',
       'Two-Wheeler']
```

[146]:
```
#Making predictions
final_predictions = XGB.predict(test_data)
```

```python
final_prediction_series = pd.Series(final_predictions)
```

[147]:
```python
#Combining the results into dataframe
submission_df = pd.DataFrame({'id':test['loan_acc_num'].values, 'LGD':
    ↪final_prediction_series.values})
```

[148]:
```python
submission_df.sample(10)
```

[148]:

	id	LGD
2697	LN61230359	0.055809
9176	LN85931981	0.239187
5647	LN19460566	0.107201
7391	LN32548392	0.405380
9173	LN70910974	0.422389
3103	LN85162535	0.435993
4000	LN91919615	0.153061
1494	LN52303333	0.647433
9172	LN30034915	0.692726
5022	LN65358503	0.186089

[ ]:
```python
submission_df.to_csv("C:\\Users\\naren\\Downloads\\BFSI Credit Risk␣
    ↪Assignment\\submission.csv",index=False)
```

[ ]:
```python

```