#Define Problem Statement and perform Exploratory Data Analysis

##**Problem Statement:**

Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

Determine the creditworthiness of potential borrowers using various attributes, to ensure that the loans are given to those who are most likely to repay them.

From the company's perspective:

● LoanTap is at the forefront of offering tailored financial solutions to millennials.

● Their innovative approach seeks to harness data science for refining their credit underwriting process.

● The focus here is the Personal Loan segment. A deep dive into the dataset can reveal patterns in borrower behavior and creditworthiness.

● Analyzing this dataset can provide crucial insights into the financial behaviors, spending habits, and potential risk associated with each borrower.

● The insights gained can optimize loan disbursal, balancing customer outreach with risk management.

From the learner's perspective:

● Tackling this case offers practical exposure to real-world financial data and its challenges.

● Logistic Regression, a foundational algorithm, is pivotal in binary outcomes like loan decisions.

● Participants will hone skills in data preprocessing, model evaluation, and understanding trade-offs, essential in the data science realm.

● The case emphasizes actionable insights, fostering the ability to drive data-informed strategies in financial sectors.

Dataset Explanation: LoanTapData.csv (Link: https://drive.google.com/file/d/1ZPYj7CZCfxntE8p2Lze_4QO4MyEOy6_d/view?usp=sharing)

1. loan_amnt: Amount borrower applied for.
2. term: Loan duration (36 or 60 months).
3. int_rate: Interest rate on loan.
4. installment: Monthly repayment amount.
5. grade: LoanTap assigned loan grade (Risk ratings by LoanTap.)
6. sub_grade: LoanTap assigned loan grade (Risk ratings by LoanTap.)
7. emp_title: Borrower's job title.
8. emp_length: Duration of borrower's employment (0-10 years).
9. home_ownership: Borrower's housing situation (own, rent, etc.).
10. annual_inc: Borrower's yearly income.
11. verification_status: Whether borrower's income was verified.

12. issue_d: Loan issuance month.
13. loan_status: Current status of the loan.
14. purpose: Borrower's reason for the loan.
15. title: The loan's title provided by the borrower.
16. dti (Debt-to-Income ratio): Monthly debt vs. monthly income ratio.
17. earliest_cr_line: Date of borrower's oldest credit account.
18. open_acc: Number of borrower's active credit lines.
19. pub_rec: Negative records on borrower's public credit profile.
20. revol_bal: Total credit balance.
21. revol_util: Usage percentage of 'revolving' accounts like credit cards.
22. total_acc: Total number of borrower's credit lines.
23. initial_list_status: Loan's first category ('W' or 'F').
24. application_type: Individual or joint application.
25. mort_acc: Number of borrower's mortgages.
26. pub_rec_bankruptcies: Bankruptcy records for borrower.
27. Address: Borrower's location.

What is Expected?

Assuming you are a data scientist at LoanTap, you are tasked with analyzing the dataset to determine the creditworthiness of potential borrowers. Your ultimate objective is to build a logistic regression model, evaluate its performance, and provide actionable insights for the underwriting process.

##Initial Analysis

Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

!pip install category_encoders
import category_encoders as ce

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import (
  confusion_matrix
  ,ConfusionMatrixDisplay
  ,accuracy_score
```

```
  ,precision_score
  ,recall_score
  ,f1_score
  ,roc_curve
  ,roc_auc_score
  ,precision_recall_curve
  ,auc
)

from statsmodels.stats.outliers_influence import
variance_inflation_factor

from sklearn.model_selection import KFold, cross_val_score

from imblearn.over_sampling import SMOTE

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.set_option('display.max_columns', None)
```

```
Requirement already satisfied: category_encoders in
/usr/local/lib/python3.10/dist-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.2.2)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(0.14.2)
Requirement already satisfied: pandas>=1.0.5 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(2.0.3)
Requirement already satisfied: patsy>=0.5.1 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2024.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
```

```
packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0-
>category_encoders) (24.0)
```

```
data = pd.read_csv('logistic_regression.csv')
```

```
data.shape
```

```
(396030, 27)
```

There are 396030 rows and 27 columns. Total no of features = 26 as 'Loan Status' is target variable which our model should be predicting. We have data about 396030 loan applications with their loan status which tells us whether borrower defaulted or not. Thus, it's supervised ML problem.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394274 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
```

```
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

```
data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade',
'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose',
'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status',
'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

```
data.nunique()
```

```
loan_amnt                 1397
term                         2
int_rate                   566
installment              55706
grade                        7
sub_grade                   35
emp_title               173105
emp_length                  11
home_ownership               6
annual_inc               27197
verification_status          3
issue_d                    115
loan_status                  2
purpose                     14
title                    48816
dti                       4262
earliest_cr_line           684
open_acc                    61
pub_rec                     20
revol_bal                55622
revol_util                1226
total_acc                  118
initial_list_status          2
```

```
application_type              3
mort_acc                     33
pub_rec_bankruptcies          9
address                  393700
dtype: int64
```

```
cat_columns = ['term', 'grade', 'sub_grade', 'emp_length',
'home_ownership', 'verification_status', 'loan_status', 'purpose',
'initial_list_status', 'application_type', 'emp_title', 'title']

for col in cat_columns:
    print(data[col].value_counts())
```

```
term
 36 months    302005
 60 months     94025
Name: count, dtype: int64
grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
sub_grade
B3    26655
B4    25601
C1    23662
C2    22580
B2    22495
B5    22085
C3    21221
C4    20280
B1    19182
A5    18526
C5    18244
D1    15993
A4    15789
D2    13951
D3    12223
D4    11657
A3    10576
A1     9729
D5     9700
A2     9567
E1     7917
E2     7431
E3     6207
```

```
E4     5361
E5     4572
F1     3536
F2     2766
F3     2286
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: count, dtype: int64
emp_length
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: count, dtype: int64
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64
verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64
loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64
purpose
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
```

```
small_business           5701
car                      4697
medical                  4196
moving                   2854
vacation                 2452
house                    2201
wedding                  1812
renewable_energy          329
educational               257
Name: count, dtype: int64
initial_list_status
f    238066
w    157964
Name: count, dtype: int64
application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64
emp_title
Teacher                        4389
Manager                        4250
Registered Nurse               1856
RN                             1846
Supervisor                     1830
                               ...
Postman                           1
McCarthy & Holthus, LLC           1
jp flooring                       1
Histology Technologist            1
Gracon Services, Inc              1
Name: count, Length: 173105, dtype: int64
title
Debt consolidation            152472
Credit card refinancing        51487
Home improvement               15264
Other                          12930
Debt Consolidation             11608
                               ...
Graduation/Travel Expenses         1
Daughter's Wedding Bill            1
gotta move                         1
creditcardrefi                     1
Toxic Debt Payoff                  1
Name: count, Length: 48816, dtype: int64
```

```
data.describe(include='all')
```

{"type":"dataframe"}

1. Loan Amount has a range of [500, 40000] with median being 12000.
2. The term for the loan is usually 36 months.
3. Median interest rate on the loan is 13.3%.
4. Monthly repayment amount is having median value of ~ 375.
5. Most of the borrowers fall under loan grade 'B' and sub-grade 'B3' (Risk rating by LoanTap)
6. The most common borrower's job title is 'Teacher' and duration of employment is '10+ years'.
7. Borrower's housing situation is usually 'Mortgage'.
8. Median Annual Income is 64000. And, it's verified in the most cases as it's necessary before approving the loan.
9. Loan issuance month is Oct-2014 for most of the data-points here.
10. Loan status is 'Fully Paid' in most cases here which means non-defaulter is majority which is great for the company.
11. In most loan application cases, borrower's reason for the loan and the loan's title provided by the borrower is 'Debt consolidation'.
12. The dti (Monthly debt vs. monthly income ratio) has median value of 16.91
13. Date of borrower's oldest credit account is Oct-2000 in most cases.
14. Number of borrower's active credit lines has median value of 10.
15. Negative records on borrower's public credit profile has median of 0 which is superb.
16. Total credit balance has median value of ~11,000
17. Usage percentage of 'revolving' accounts like credit cards has median of 54.8
18. Total number of borrower's credit lines has median of 24.
19. Loan's first category has value 'f' in most cases.
20. Application type is mostly 'Individual'.
21. Number of borrower's mortgages has median value 1.
22. Bankruptcy records for borrower is 0 in 50% of cases.
23. Borrower's location is 'AE 70466' in the most cases.

Data Exploration

```
data.groupby(by = 'loan_status')['loan_amnt'].describe()
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 2,\n  \"fields\": [\n    {\n      \"column\": \"loan_status\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          \"Fully Paid\",\n          \"Charged Off\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"count\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 170189.288523103,\n        \"min\": 77673.0,\n        \"max\": 318357.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n          318357.0,\n          77673.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"mean\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 890.5459748555348,\n        \"min\":
```

```
13866.878771316478,\n        \"max\": 15126.300966873945,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
13866.878771316478,\n          15126.300966873945\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"std\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 143.38064827557537,\n
\"min\": 8302.319699344323,\n        \"max\": 8505.090556717489,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
8302.319699344323,\n          8505.090556717489\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"min\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 353.5533905932738,\n
\"min\": 500.0,\n        \"max\": 1000.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          500.0,\n
1000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"25%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 724.7844507162113,\n        \"min\": 7500.0,\n
\"max\": 8525.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          7500.0,\n          8525.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"50%\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1414.213562373095,\n
\"min\": 12000.0,\n        \"max\": 14000.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          12000.0,\
n          14000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"75%\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 548.0077554195743,\n        \"min\": 19225.0,\n
\"max\": 20000.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          19225.0,\n          20000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\":
40000.0,\n        \"max\": 40000.0,\n        \"num_unique_values\":
1,\n        \"samples\": [\n          40000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


The no of people those who have fully paid are 318357 and that of Charged Off are 77673.

```
data['home_ownership'].value_counts()
```

```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
```

```
ANY                  3
Name: count, dtype: int64
```

The majority of people have home ownership as Mortgage and Rent.

Combining the minority classes as 'OTHER'.

```
data.loc[(data['home_ownership'] == 'ANY') | (data['home_ownership']
== 'NONE'), 'home_ownership'] = 'OTHER'
data['home_ownership'].value_counts()
```

```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: count, dtype: int64
```

```
#Checking the distribution of OTHER

data.loc[data['home_ownership'] == 'OTHER',
'loan_status'].value_counts()
```

```
loan_status
Fully Paid     123
Charged Off     23
Name: count, dtype: int64
```

Issues in title, looks like values were manually entered

```
data['title'].value_counts()[:20]
```

```
title
Debt consolidation           152472
Credit card refinancing       51487
Home improvement              15264
Other                         12930
Debt Consolidation            11608
Major purchase                 4769
Consolidation                  3852
debt consolidation             3547
Business                       2949
Debt Consolidation Loan        2864
Medical expenses               2742
Car financing                  2139
Credit Card Consolidation      1775
Vacation                       1717
Moving and relocation          1689
consolidation                  1595
Personal Loan                  1591
```

```
Consolidation Loan                 1299
Home Improvement                   1268
Home buying                        1183
Name: count, dtype: int64
```

```
data['title'] = data.title.str.lower()
```

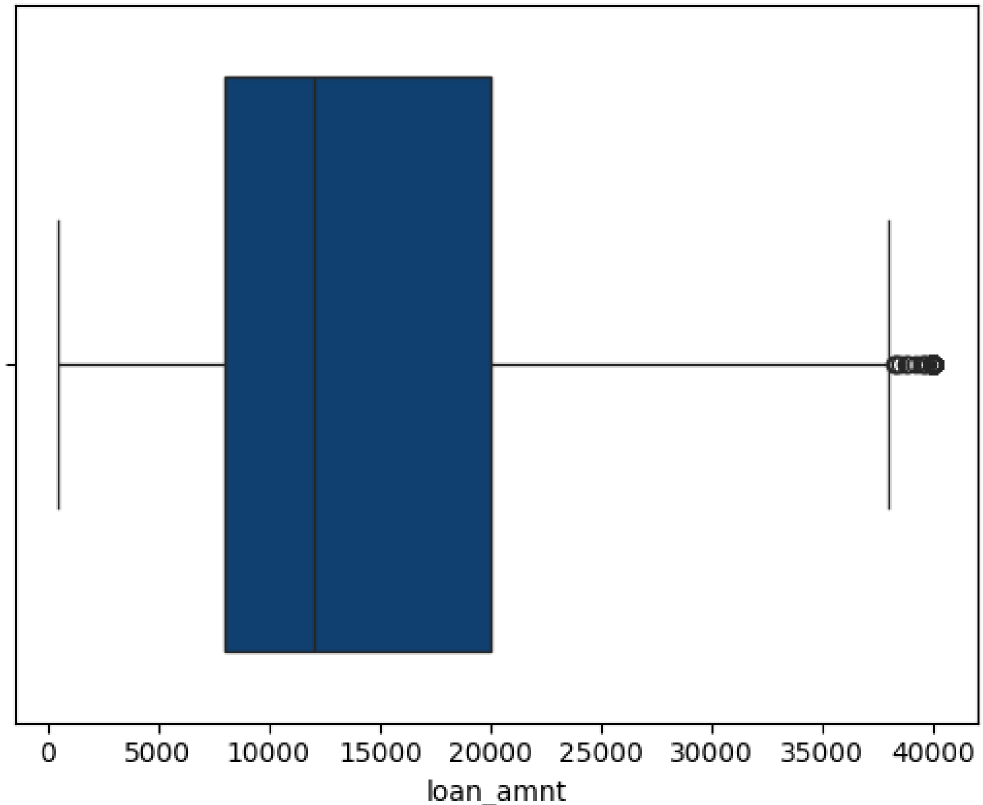
```
data['title'].value_counts()[:10]
```

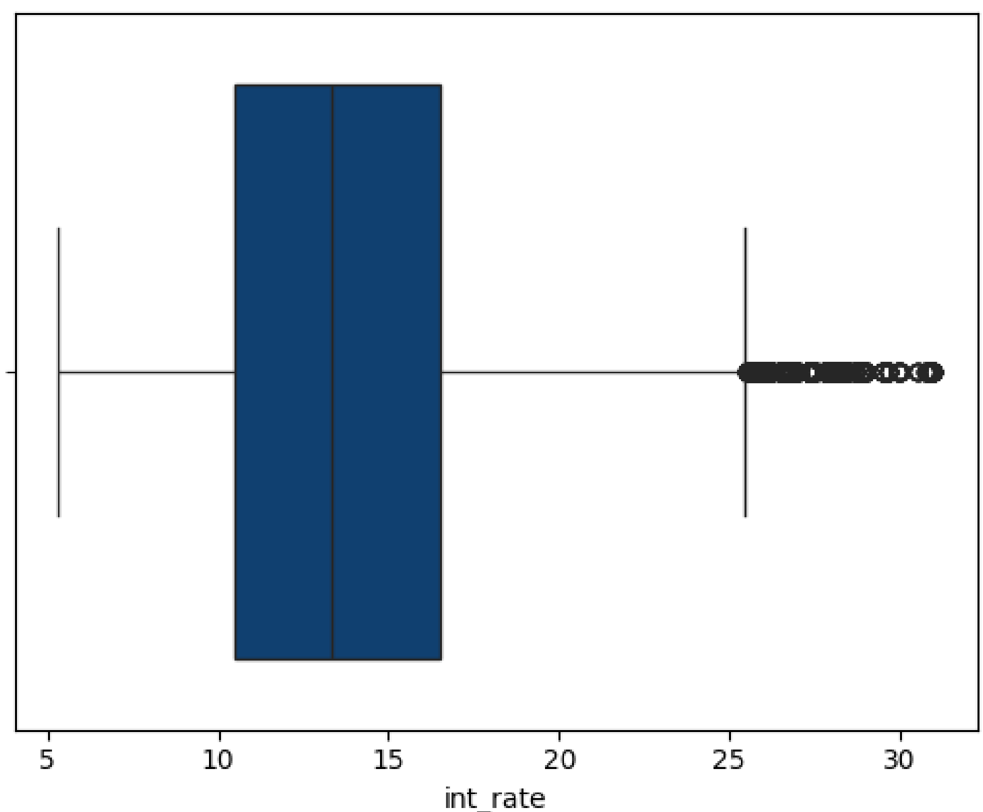
```
title
debt consolidation               168108
credit card refinancing           51781
home improvement                  17117
other                             12993
consolidation                      5583
major purchase                     4998
debt consolidation loan            3513
business                           3017
medical expenses                   2820
credit card consolidation          2638
Name: count, dtype: int64
```

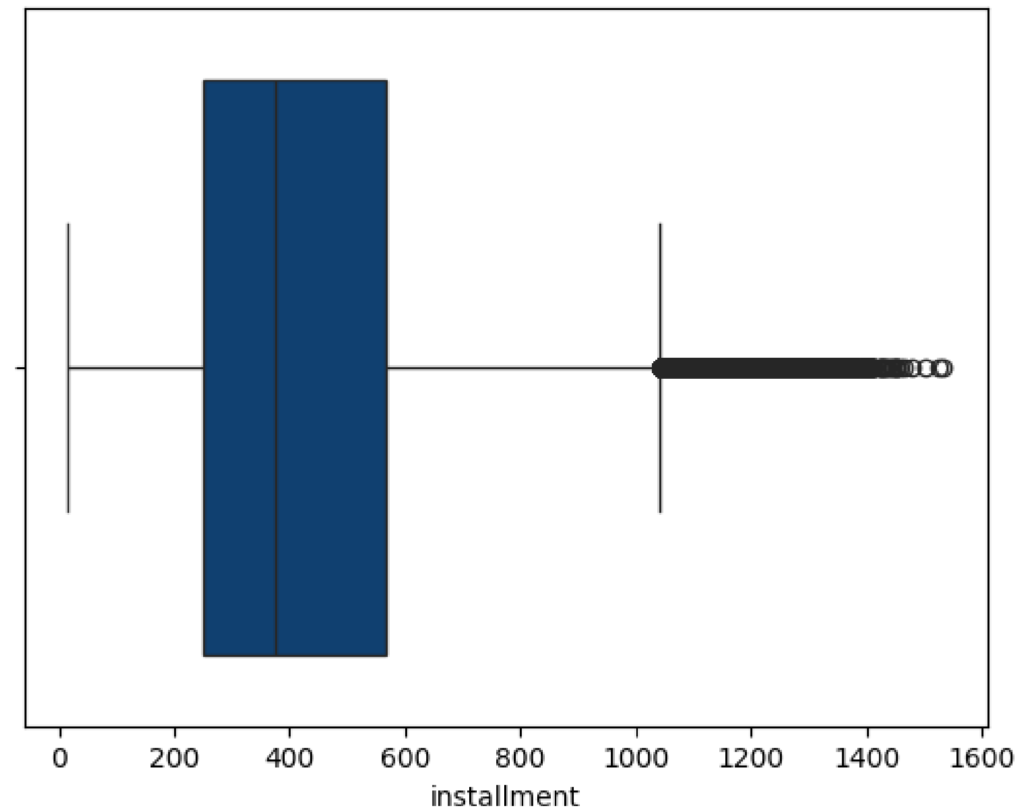
##Univariate Analysis

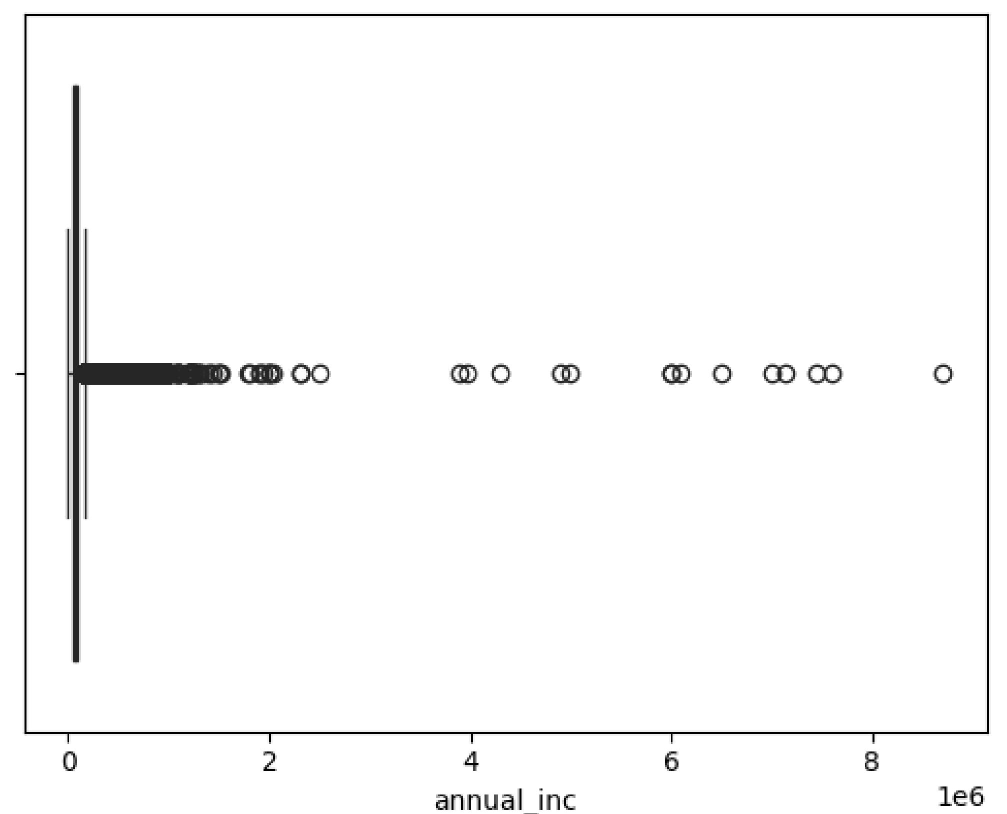
###Continuous Variables

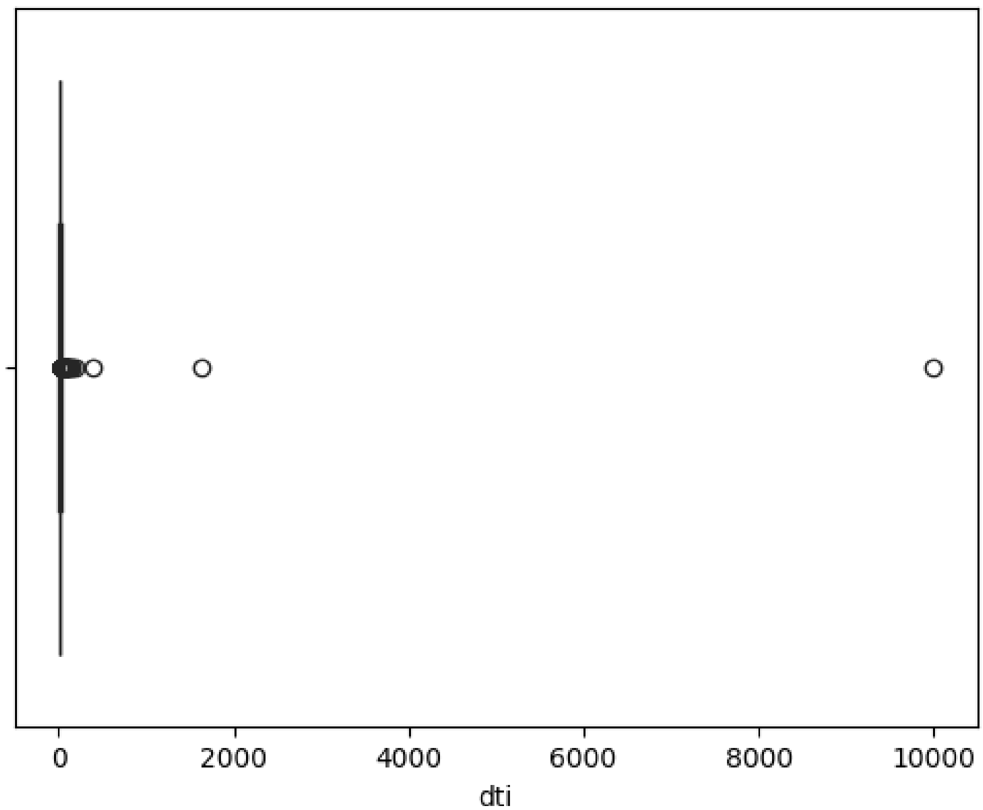
```
for col in data.columns:
  if data[col].dtype in ('float64', 'int64'):
    sns.boxplot(data = data, x = col, palette = 'ocean')
    plt.show()
```

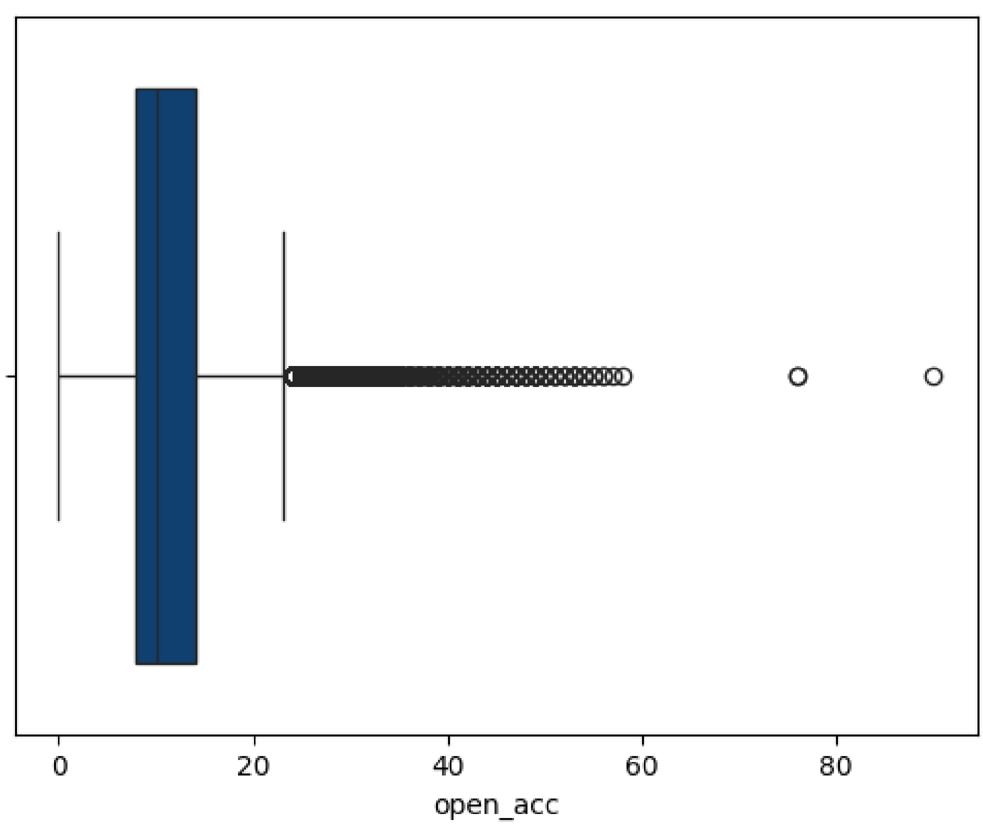

0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt

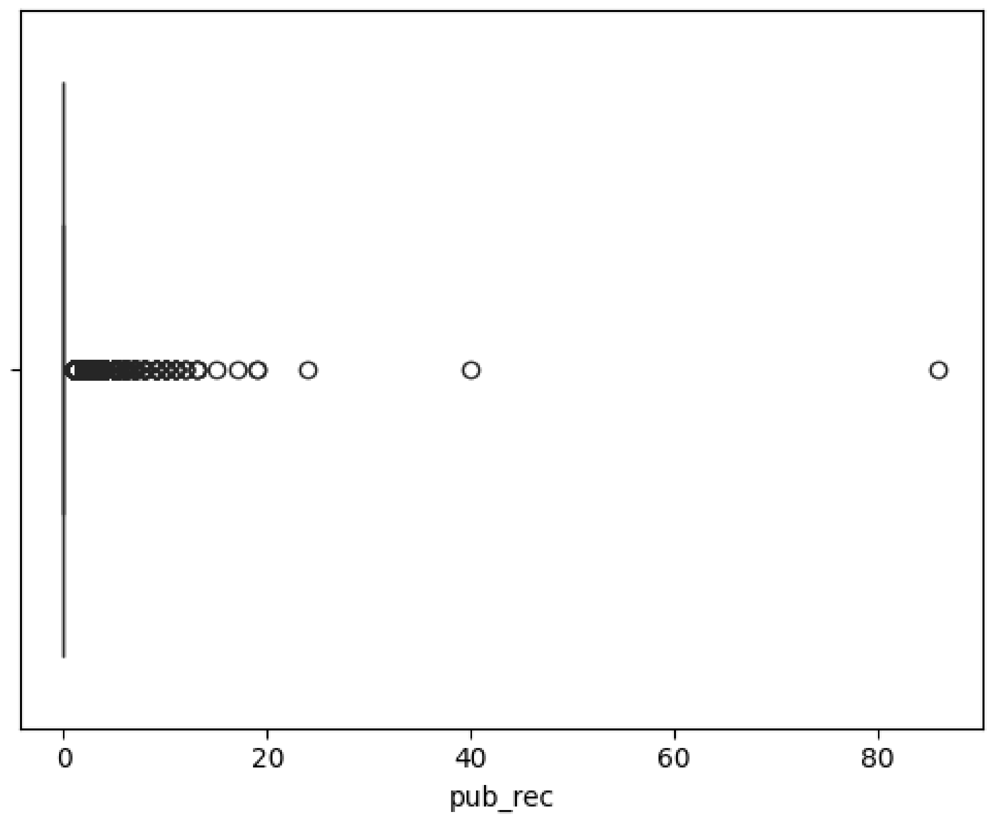

5
10
15
20
25
30
int_rate

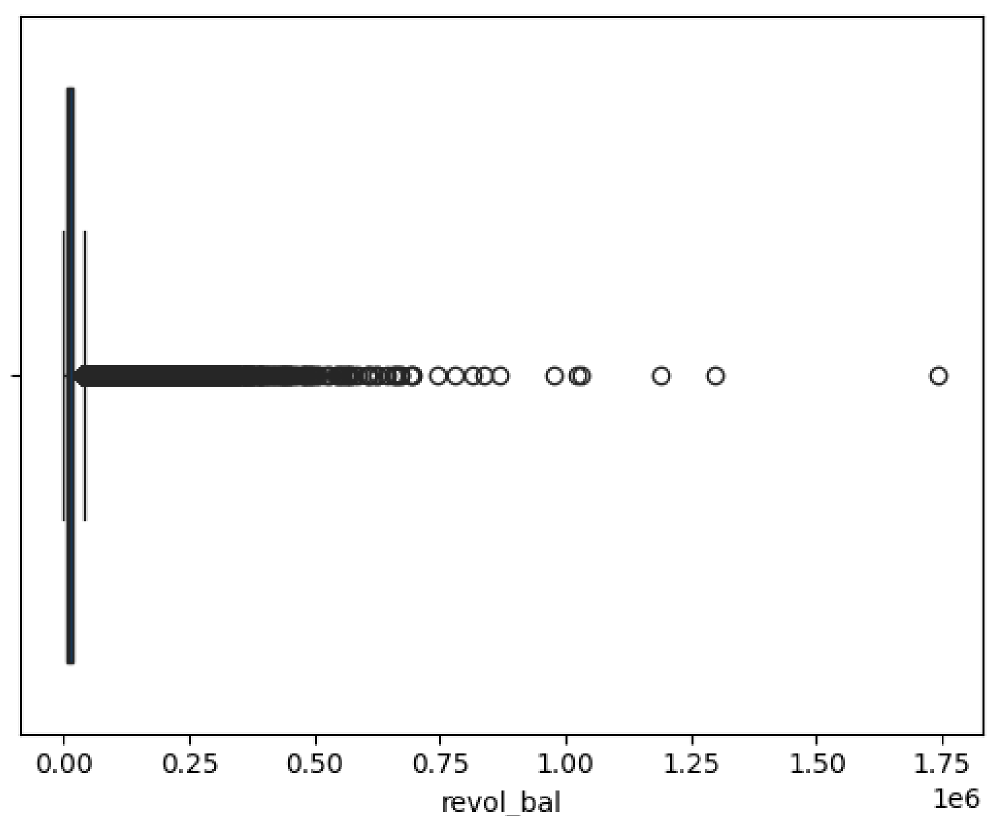

0
200
400
600
800
1000
1200
1400
1600
installment

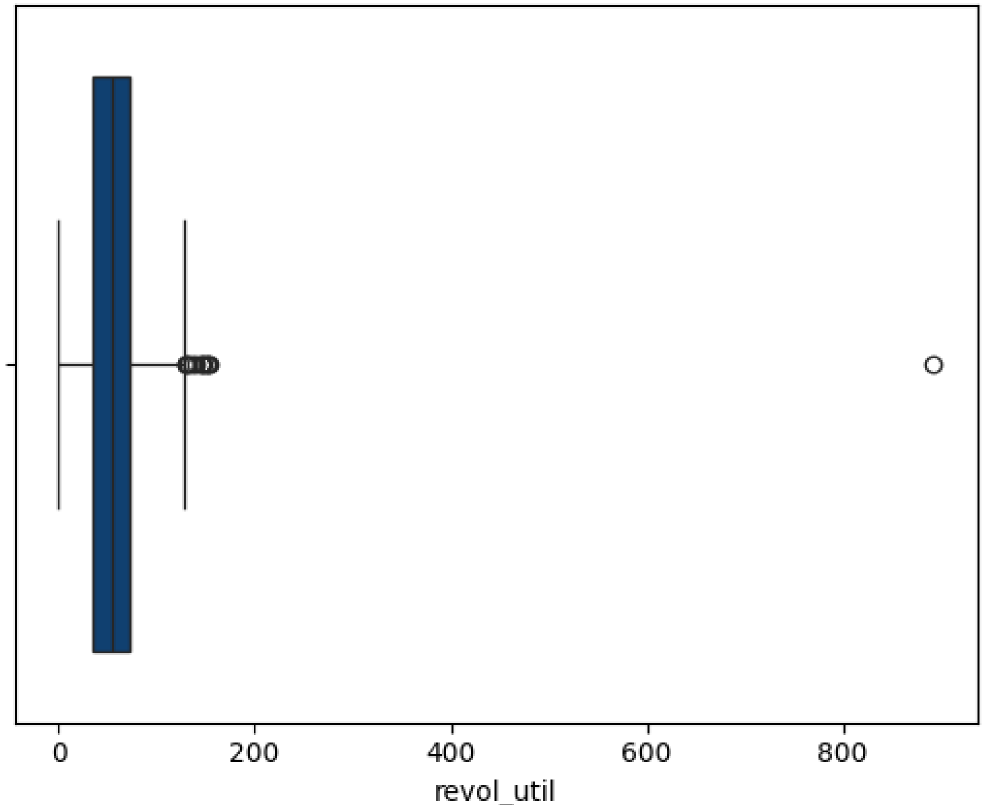

0
2
4
6
8
1e6
annual_inc

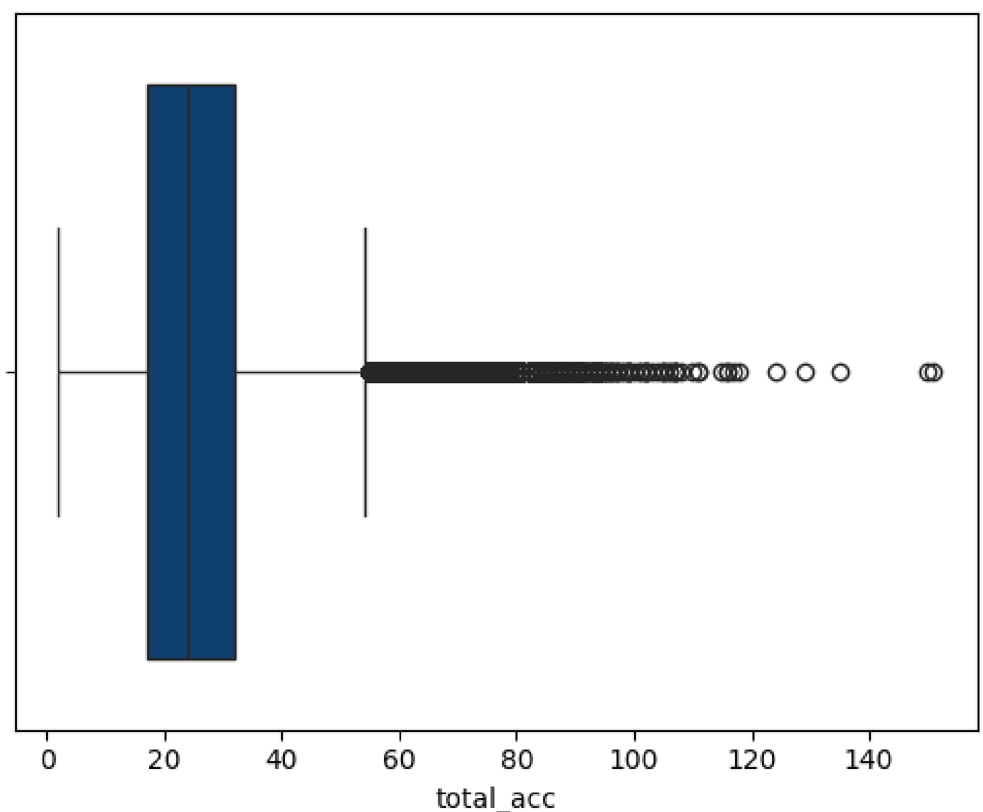

0
2000
4000
6000
8000
10000
dti

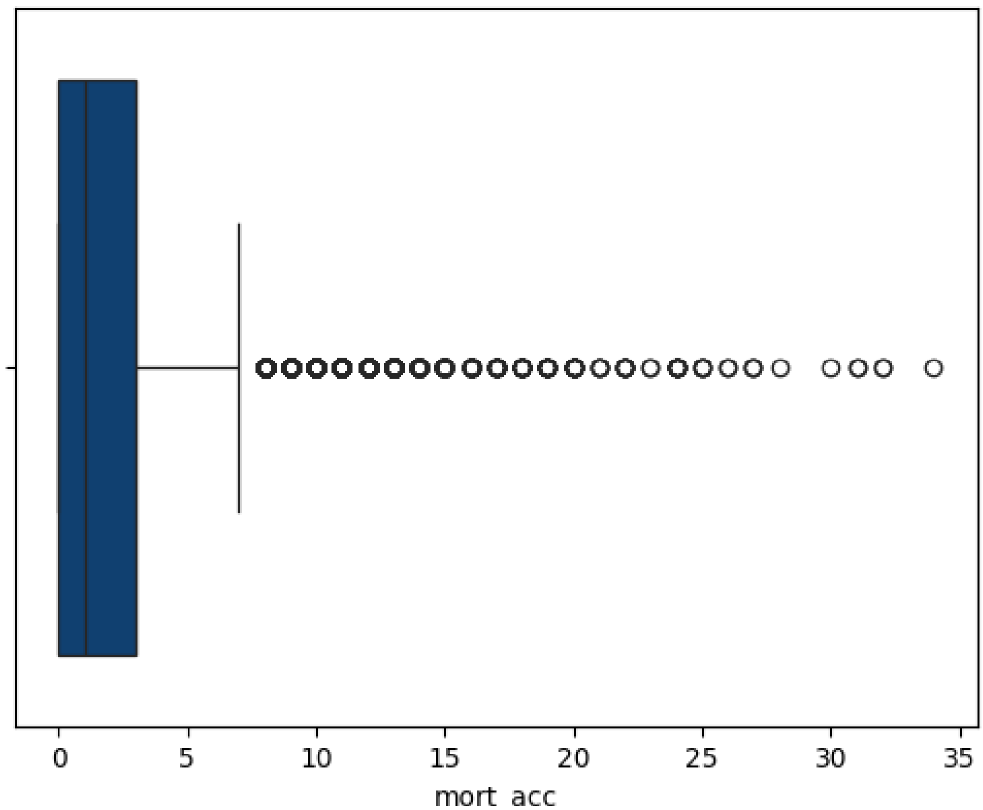

0
20
40
60
80
open_acc

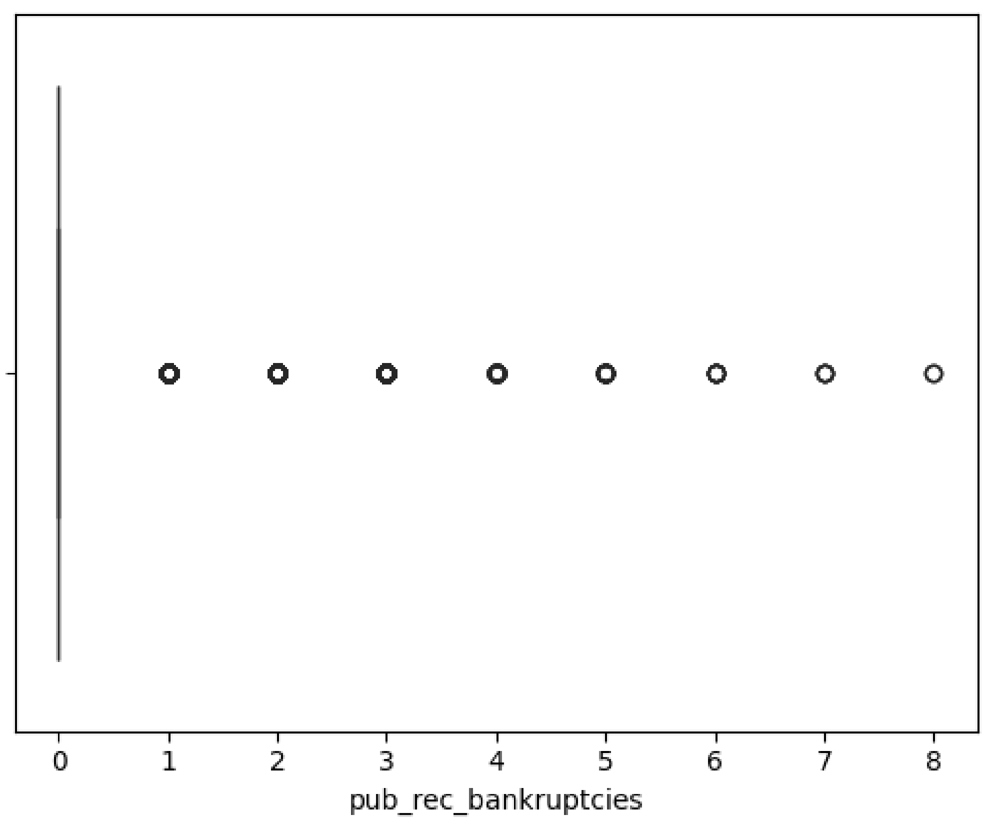

0
20
40
60
80
pub_rec


0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal


0
200
400
600
800
revol_util


0
20
40
60
80
100
120
140
total_acc

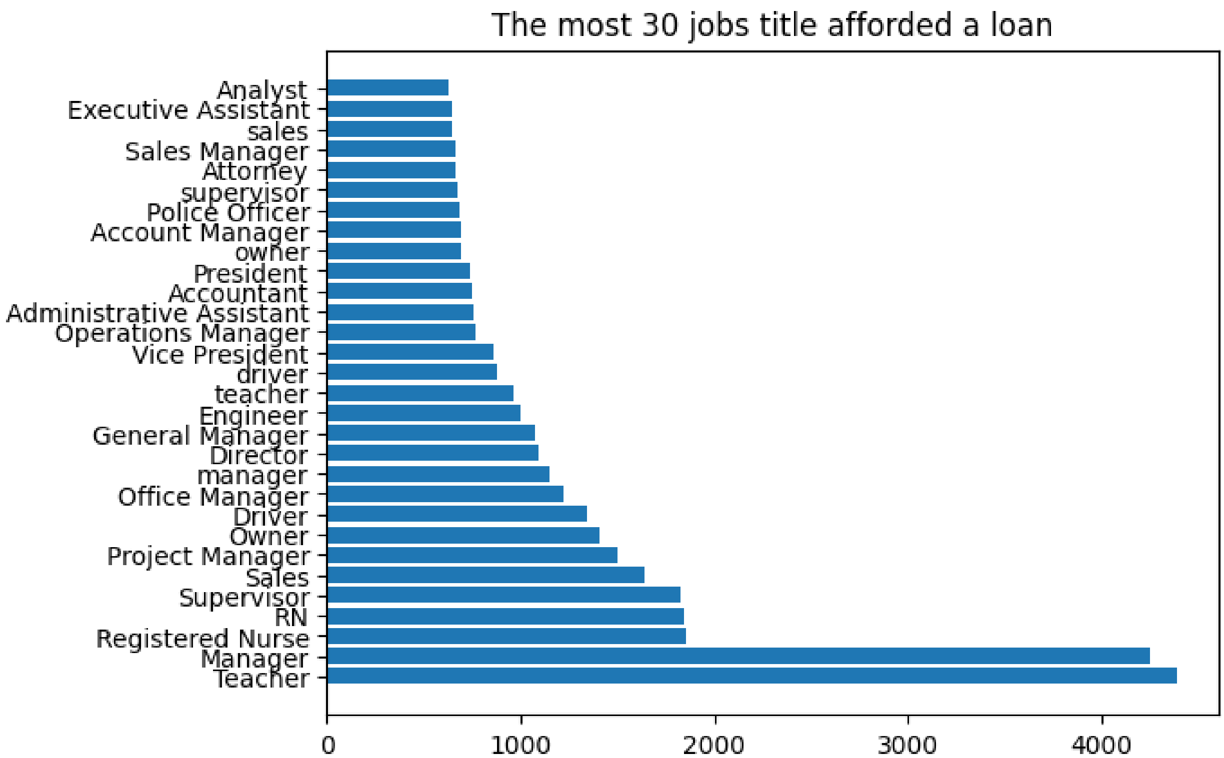

0
5
10
15
20
25
30
35
mort_acc


0
1
2
3
4
5
6
7
8
pub_rec_bankruptcies

The above box-plots tell us that there are outliers present in all the continuous variables.

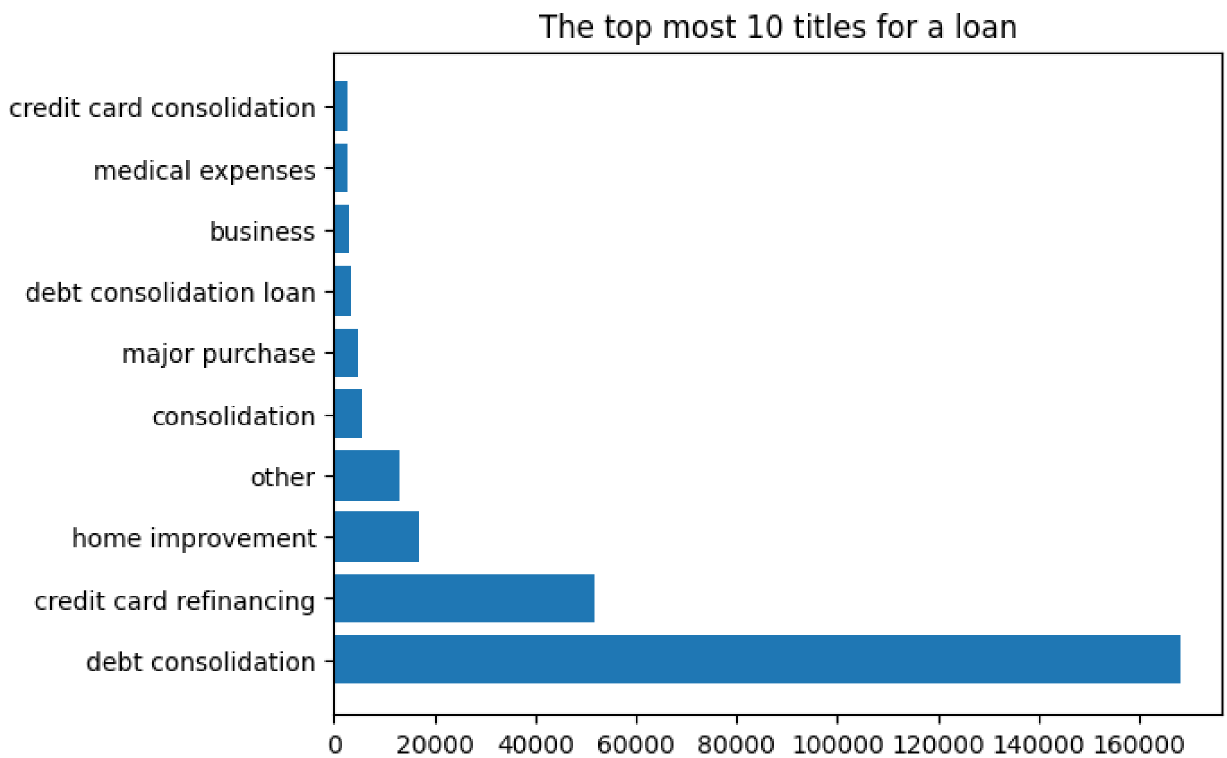
###Categorical Variables

```
plt.barh(data.emp_title.value_counts()[:30].index,
data.emp_title.value_counts()[:30])
plt.title("The most 30 jobs title afforded a loan")
plt.show()
```



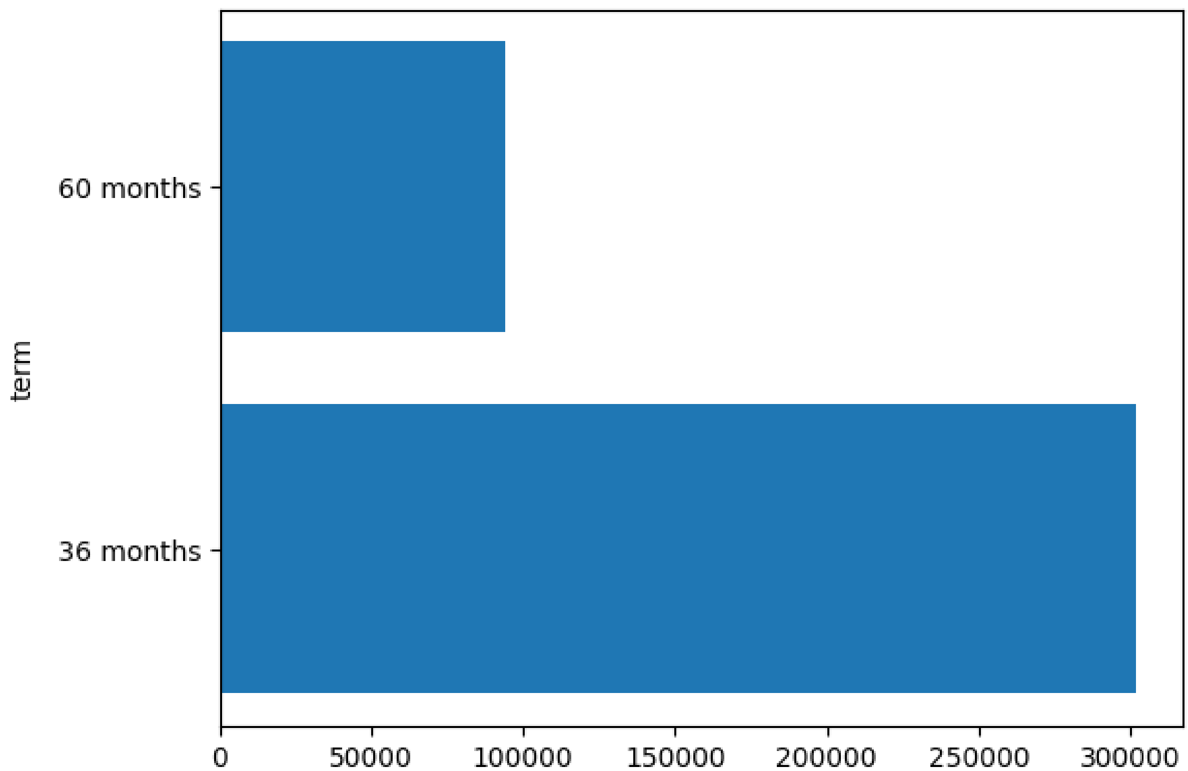
The top 2 employee job roles for which we have the highest no of loan applications are 'Teacher' followed by 'Manager'.

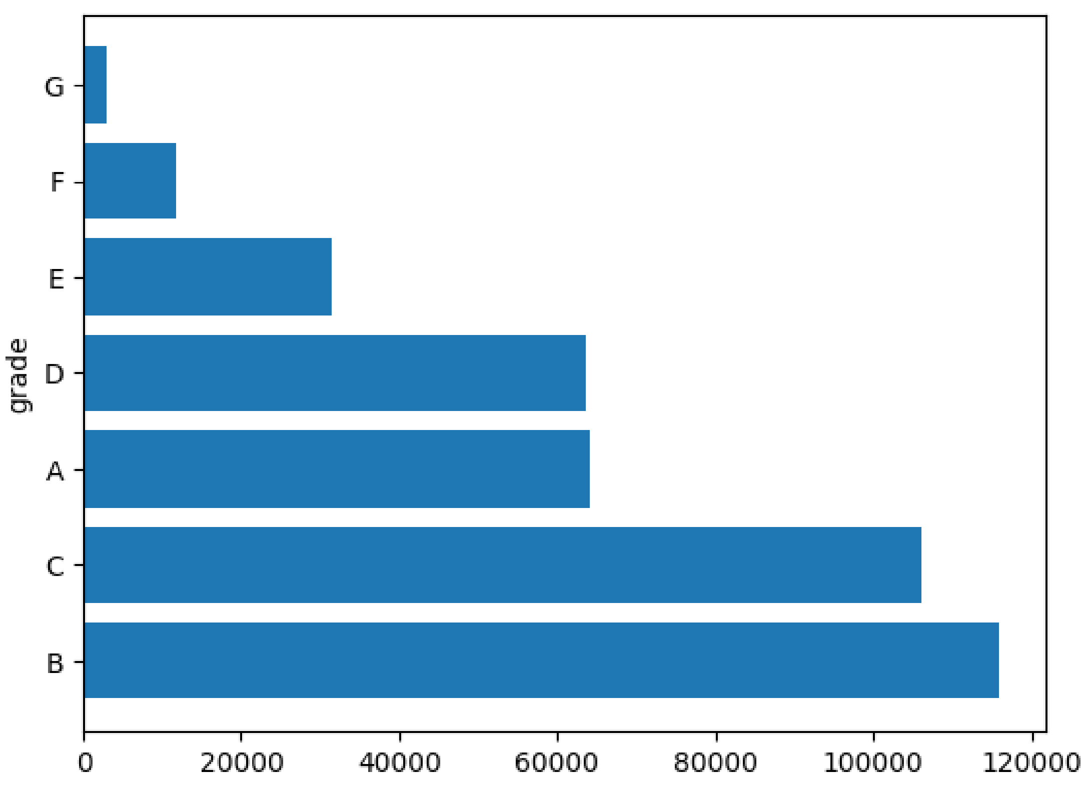
```
plt.barh(data.title.value_counts()[:10].index,
data.title.value_counts()[:10])
plt.title("The top most 10 titles for a loan")
plt.show()
```

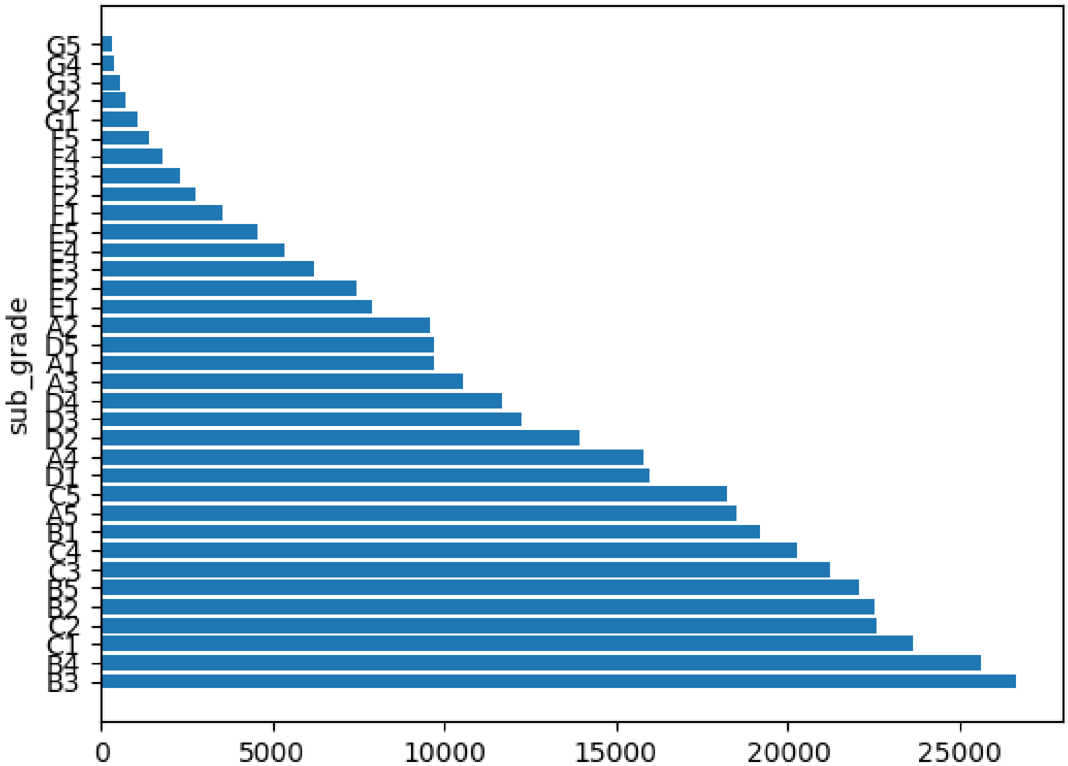


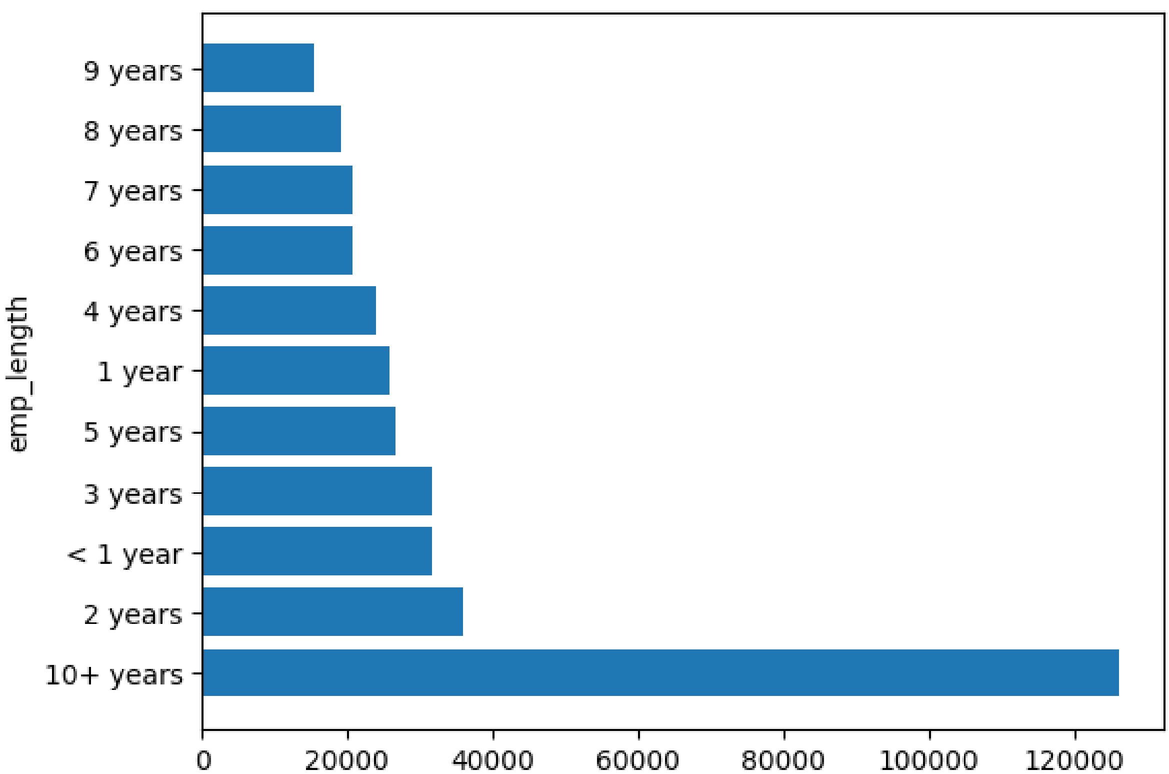
The loan's title provided by the borrower is 'Debt Consolidation' for the most no of applications.

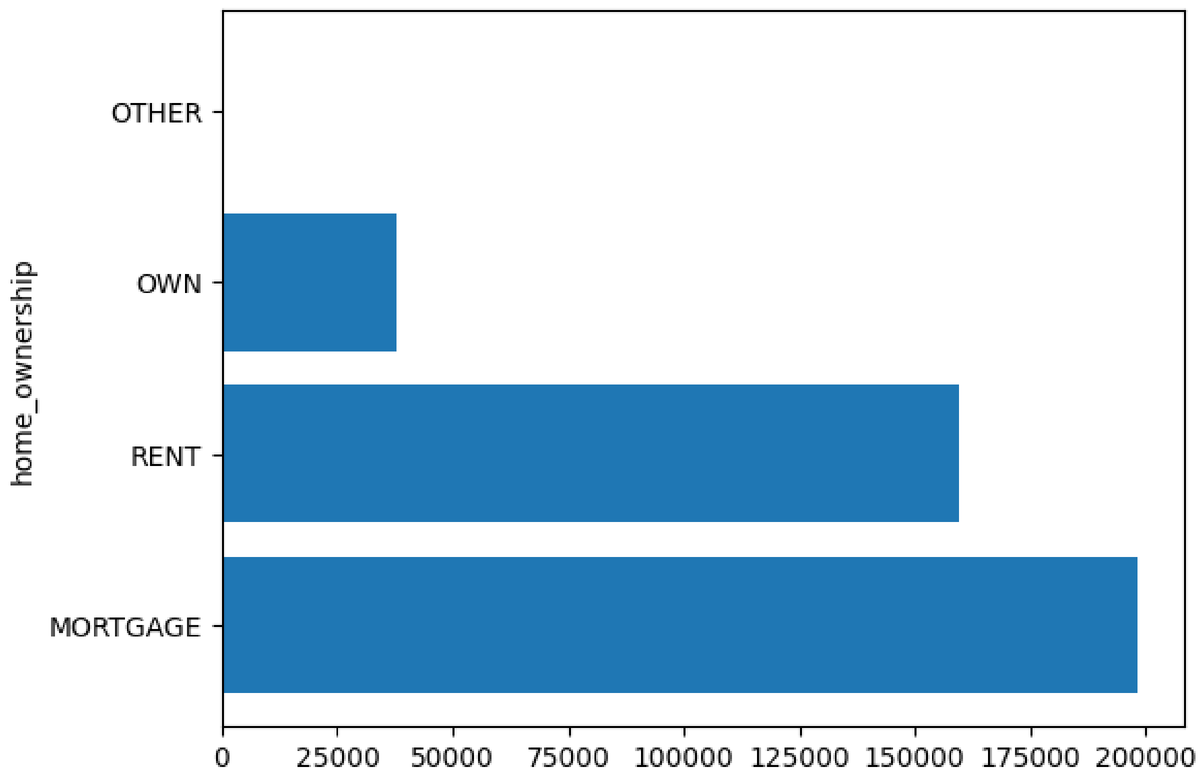
```
for col in cat_columns:
  if col not in ('emp_title', 'title'):
    plt.barh(data[col].value_counts().index, data[col].value_counts())
    plt.ylabel(col)
    plt.show()
```

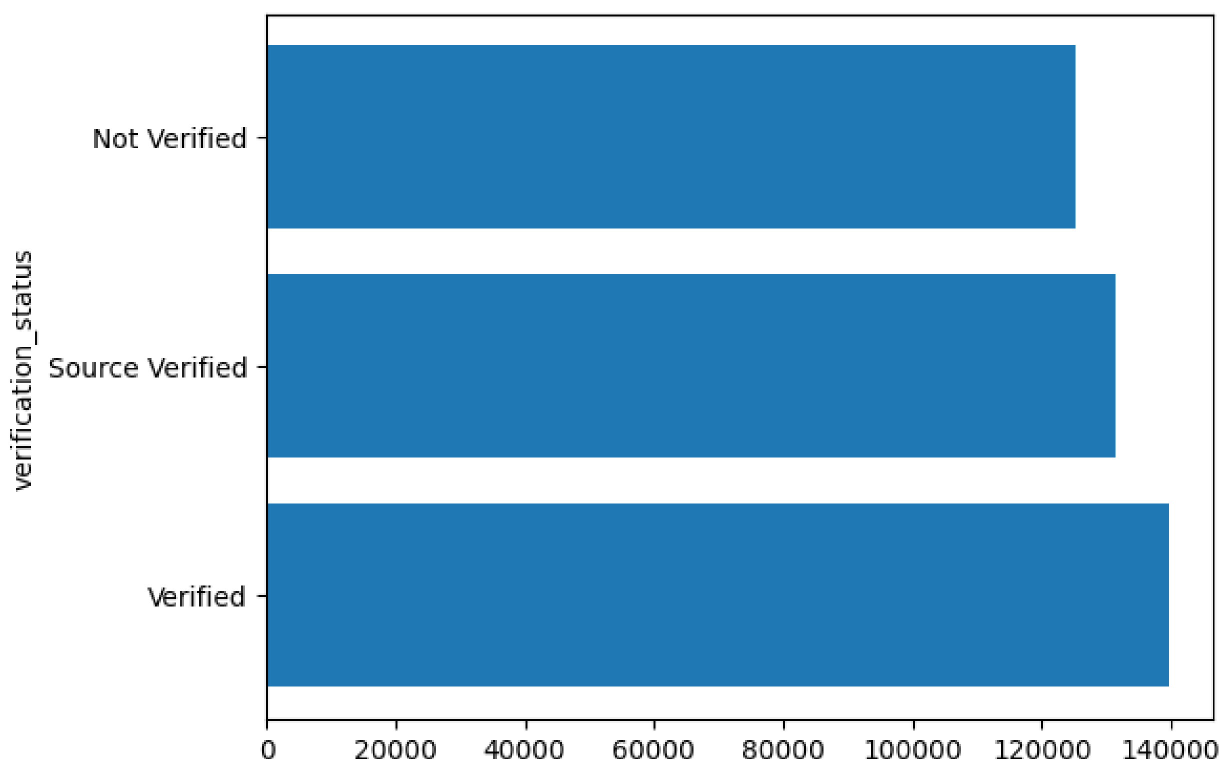

60 months
36 months
term
0
50000
100000
150000
200000
250000
300000

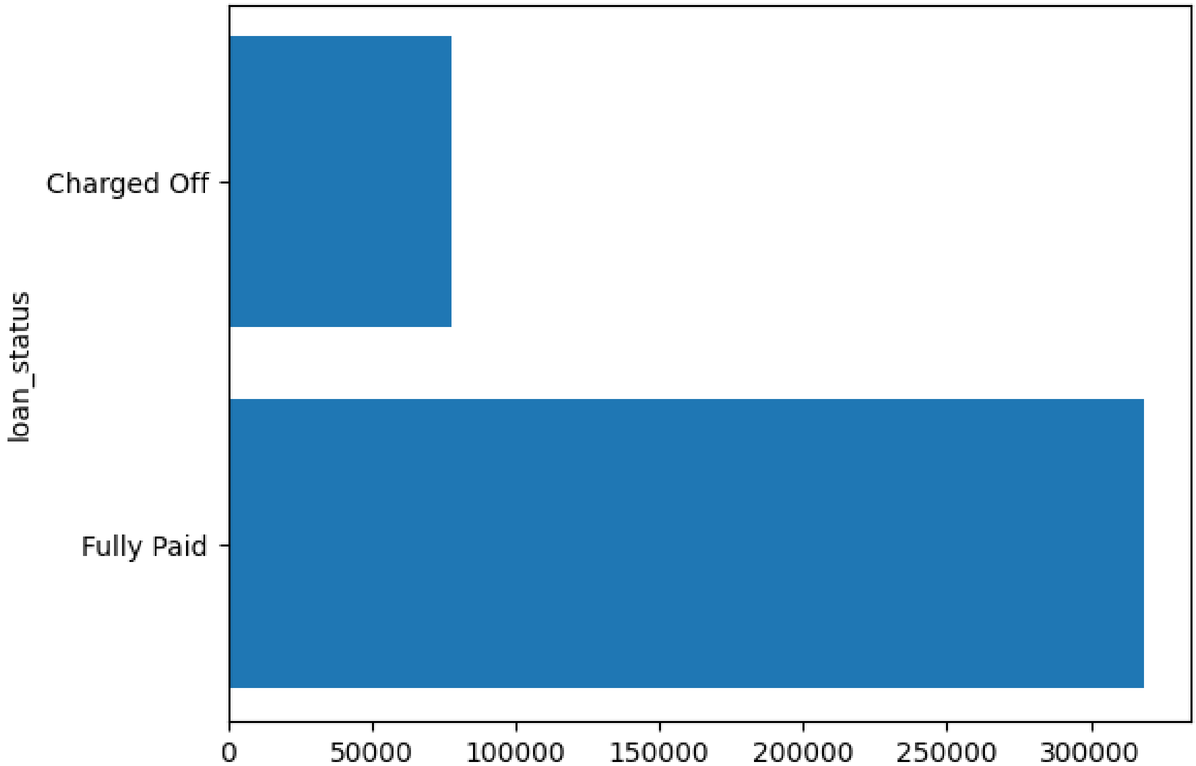

G
F
E
D
A
C
B
grade
0
20000
40000
60000
80000
100000
120000

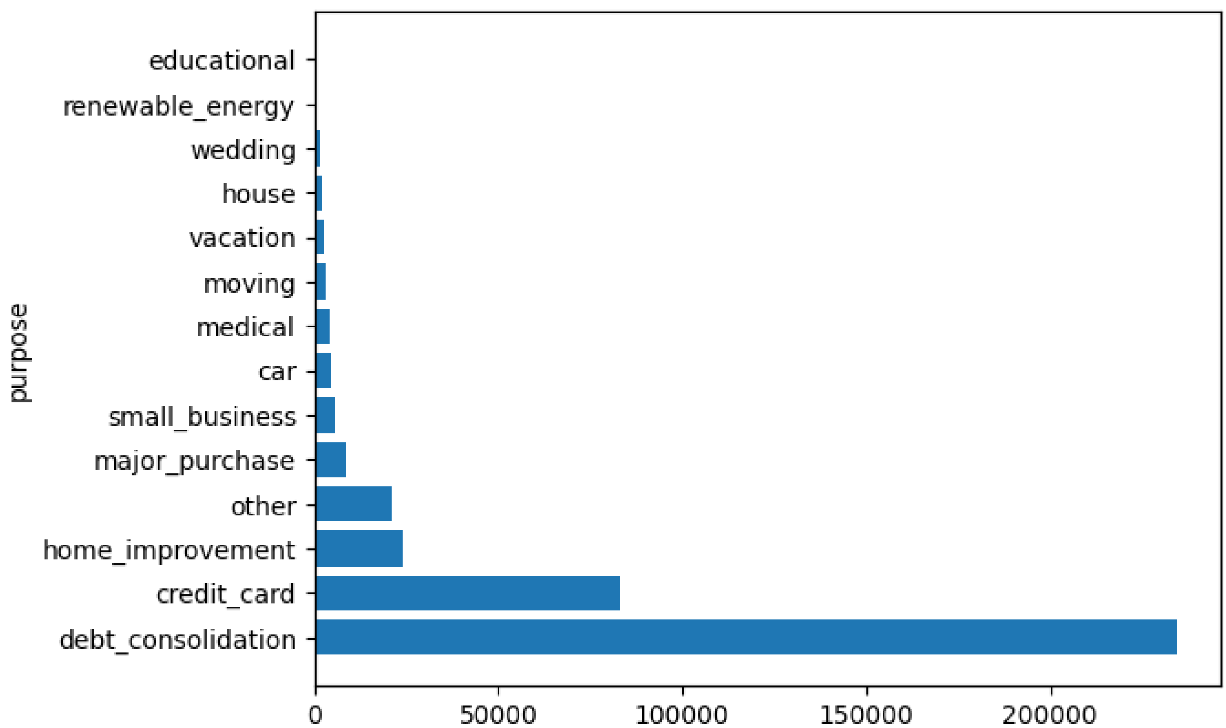

sub_grade
G5
G4
G3
G2
G1
F5
F4
F3
F2
F1
E5
E4
E3
E2
E1
A2
D5
A1
A3
D4
D3
D2
A4
D1
C5
A5
B1
C4
C3
B5
B2
C2
C1
B4
B3
0
5000
10000
15000
20000
25000

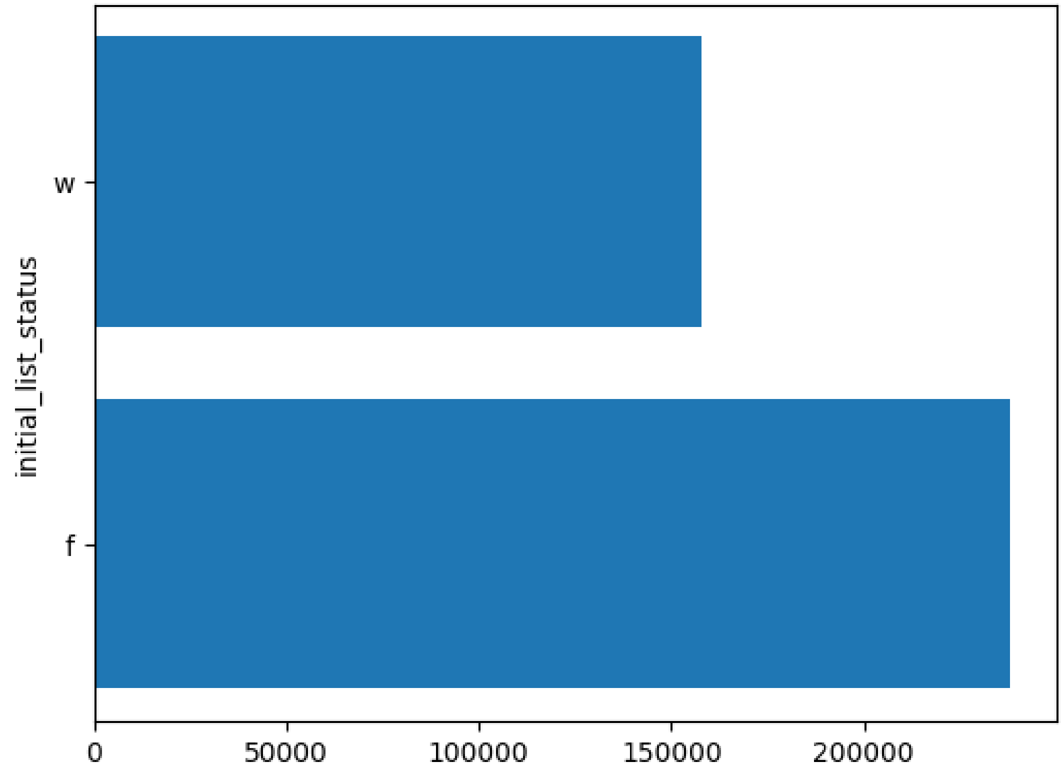

emp_length
9 years
8 years
7 years
6 years
4 years
1 year
5 years
3 years
< 1 year
2 years
10+ years
0
20000
40000
60000
80000
100000
120000

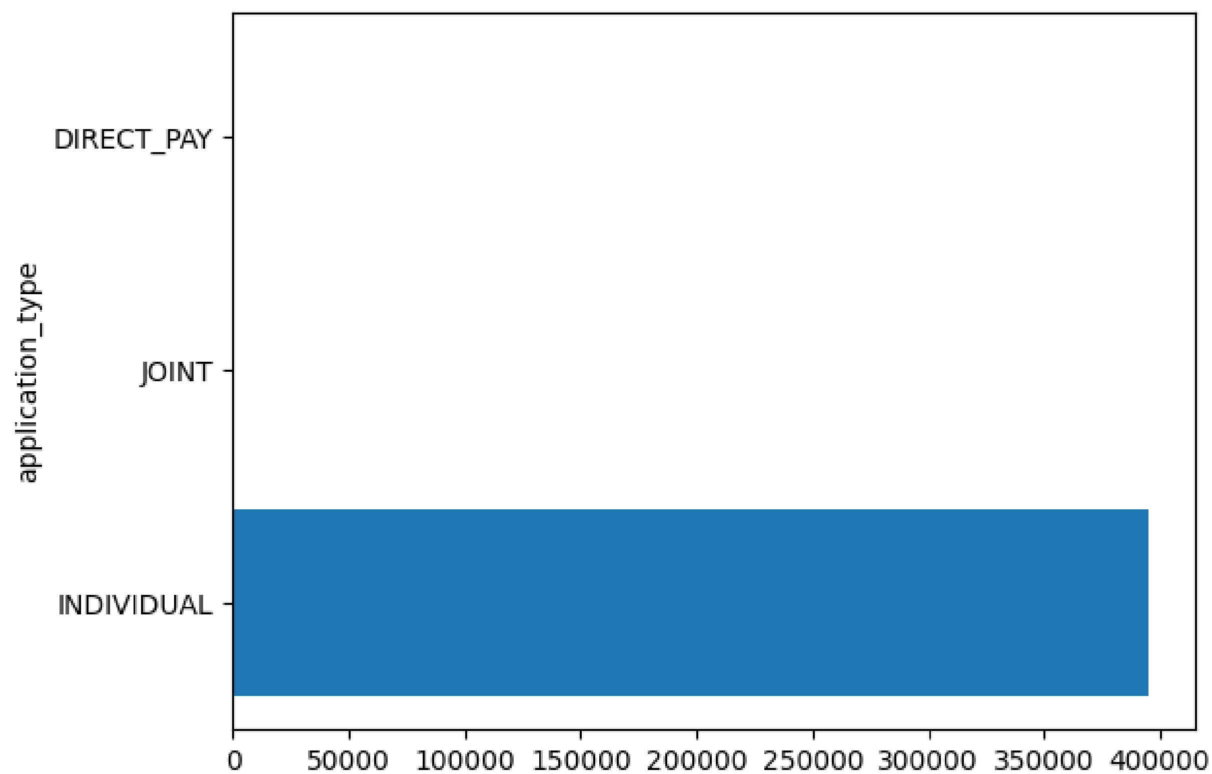

OTHER
OWN
RENT
MORTGAGE
home_ownership
0
25000
50000
75000
100000
125000
150000
175000
200000


Not Verified
Source Verified
Verified
verification_status
0
20000
40000
60000
80000
100000
120000
140000


Charged Off
Fully Paid
loan_status
0
50000
100000
150000
200000
250000
300000


educational
renewable_energy
wedding
house
vacation
moving
medical
car
small_business
major_purchase
other
home_improvement
credit_card
debt_consolidation
purpose
0
50000
100000
150000
200000





1. We have majority of loan applications for the loan term of '36 months'. Which tells us that borrowers prefer lesser duration for the loan term.

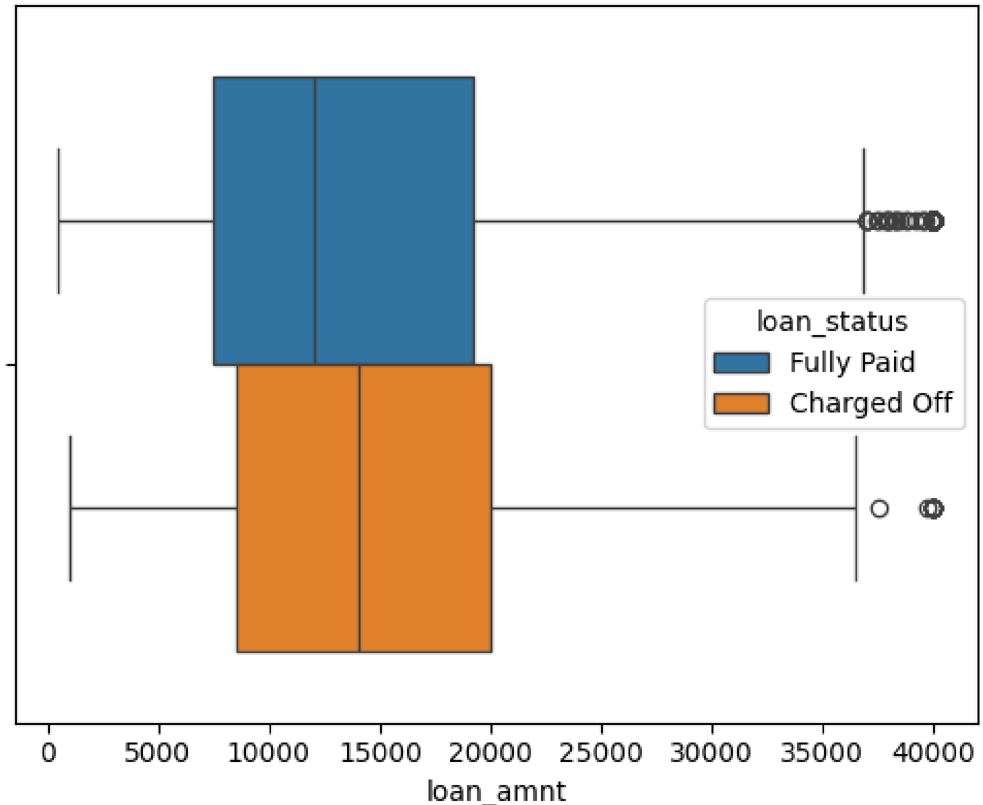
2. The loan applications fall under the grades 'B' followed by 'C', and sub-grade 'B3' followed by 'B4'.
3. Majority of the borrowers have the employment duration more than 10 years.
4. Top home ownership options observed are 'Mortgage' followed by 'Rent'.
5. The no of loan applications among different borrower income verification status ('Not Verified', 'Source Verified' and 'Verified') is almost similar which is not good as it's important to verify the income before approving the loan to minimize the defaulters and the subsequent money loss.
6. Majority of the loan applications are fully paid which is good.
7. Top 2 purposes for which loan was taken are 'Debt Consolidation' and 'Credit Card'.
8. The initial list status for majority of loan applications is 'f' which means fractional.
9. Loan application type is mostly 'Individual'.

##Bivariate Analysis

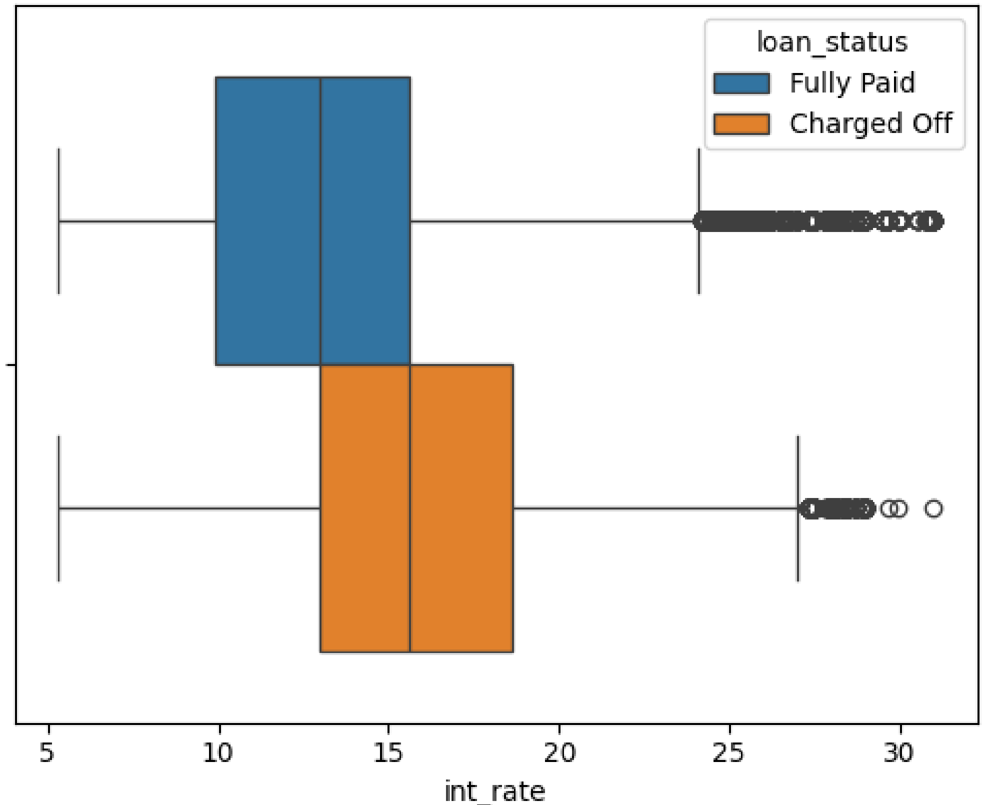
Target (Categorical) vs Features (Continuous)

```
for col in data.columns:
  if data[col].dtype != 'object':
    sns.boxplot(data = data, x = col, hue = 'loan_status', orient = 'v')
    plt.show()
```

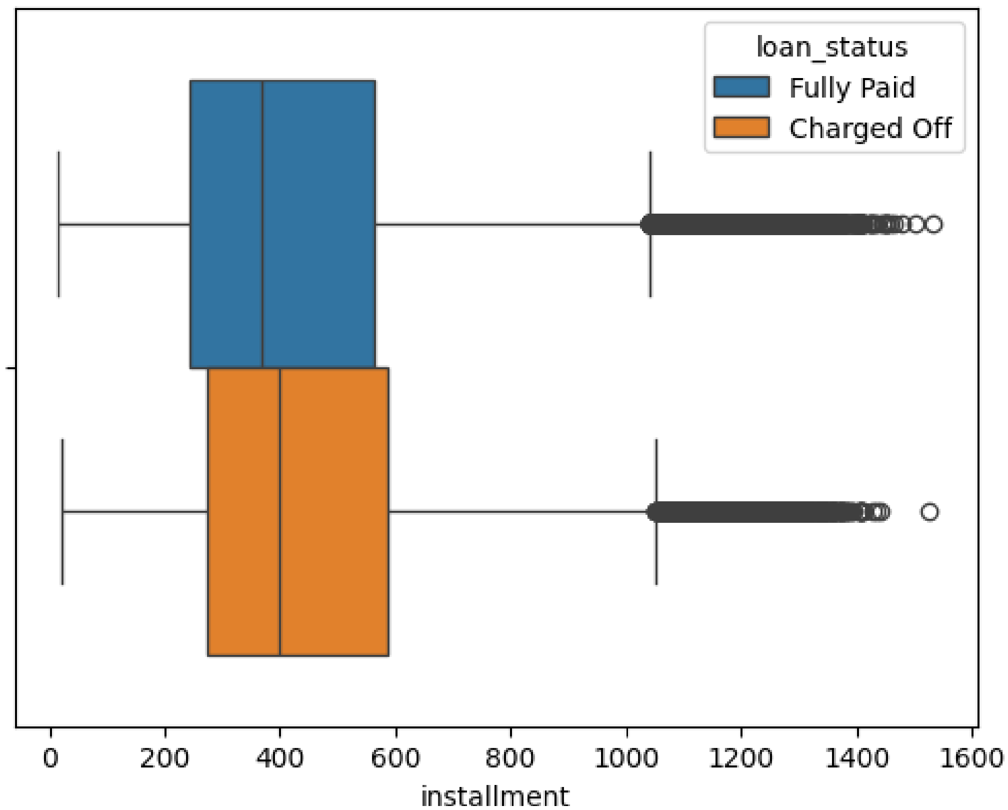
```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



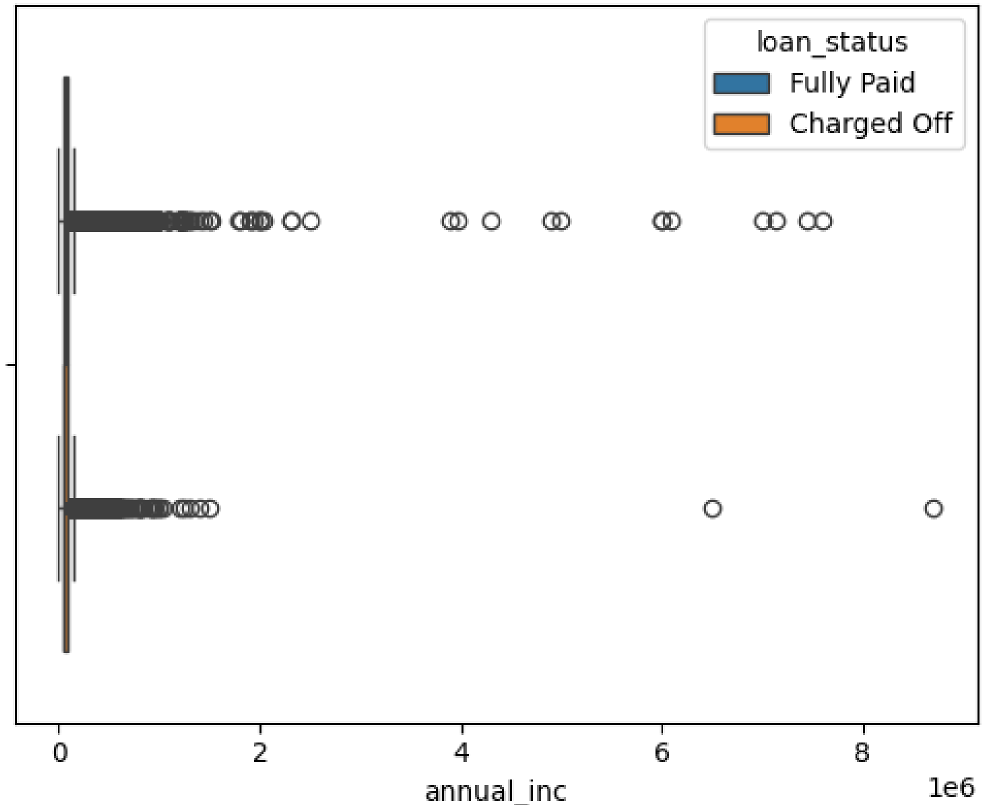

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



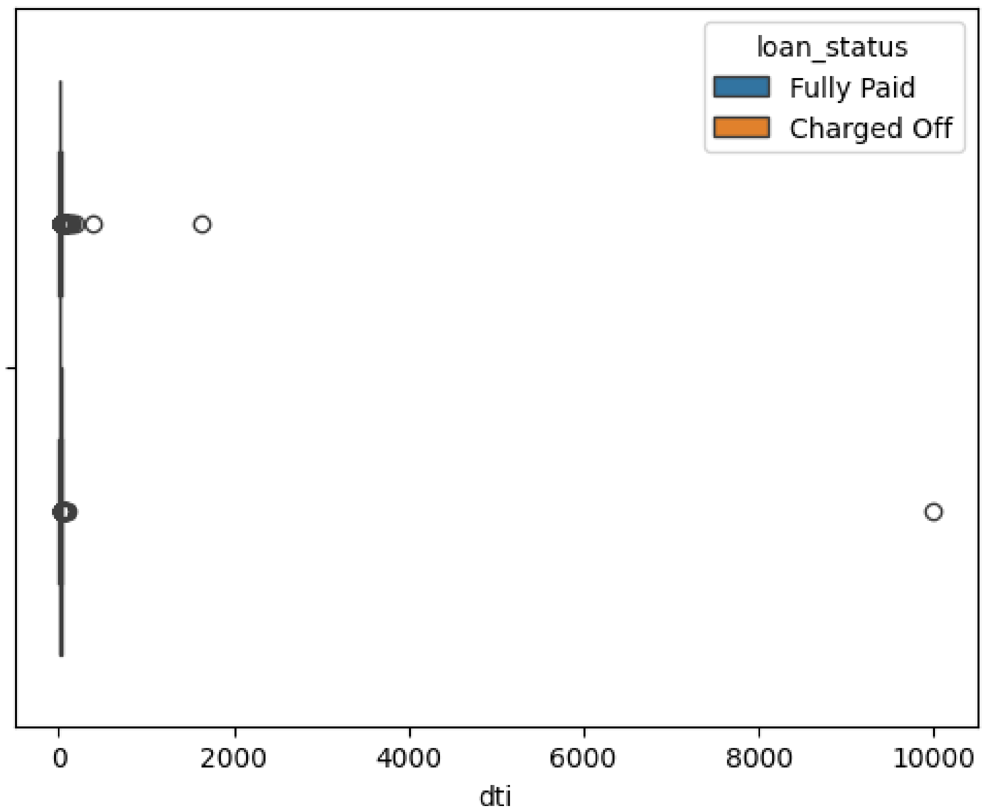

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



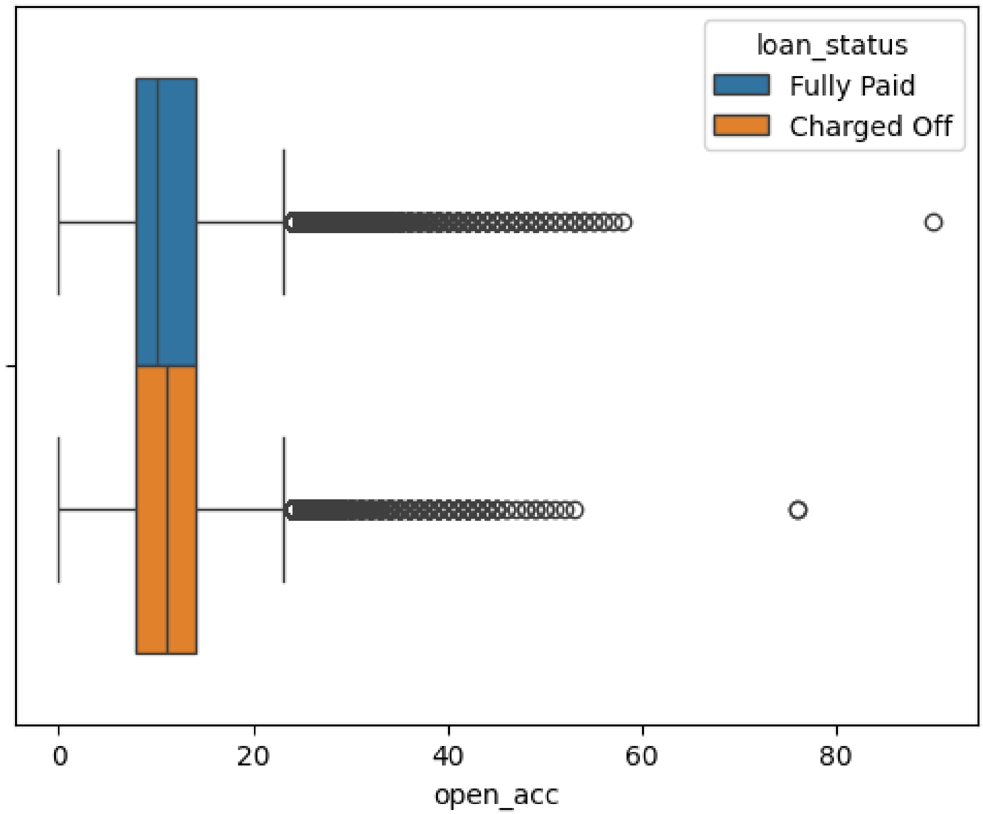

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



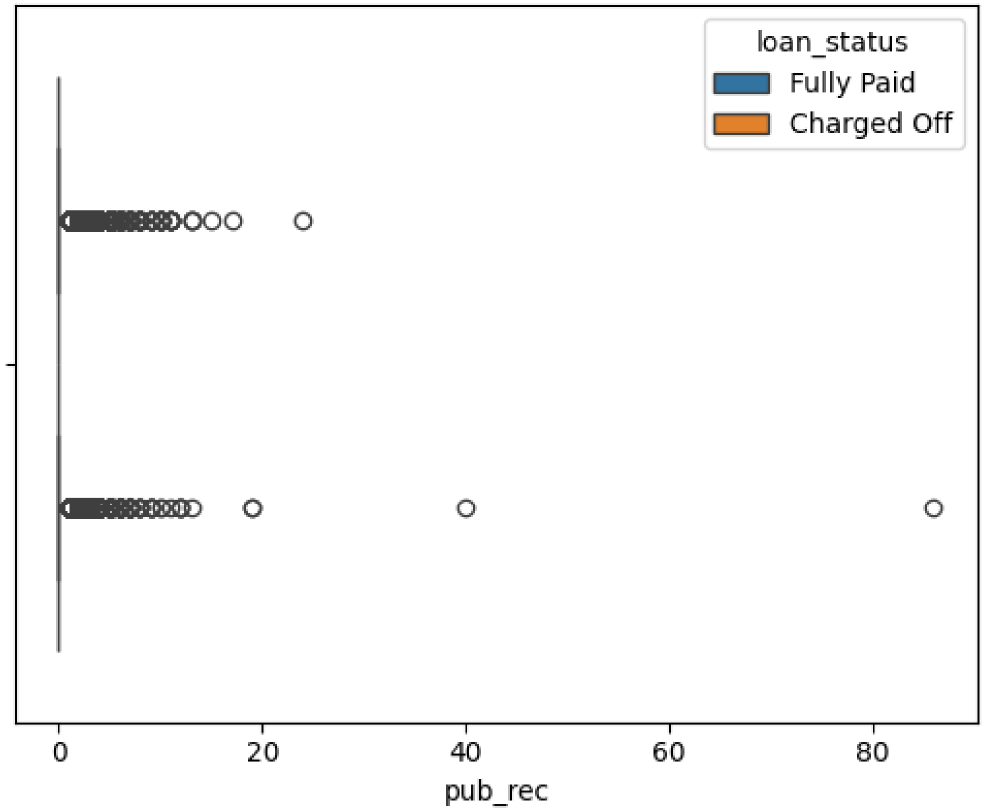

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



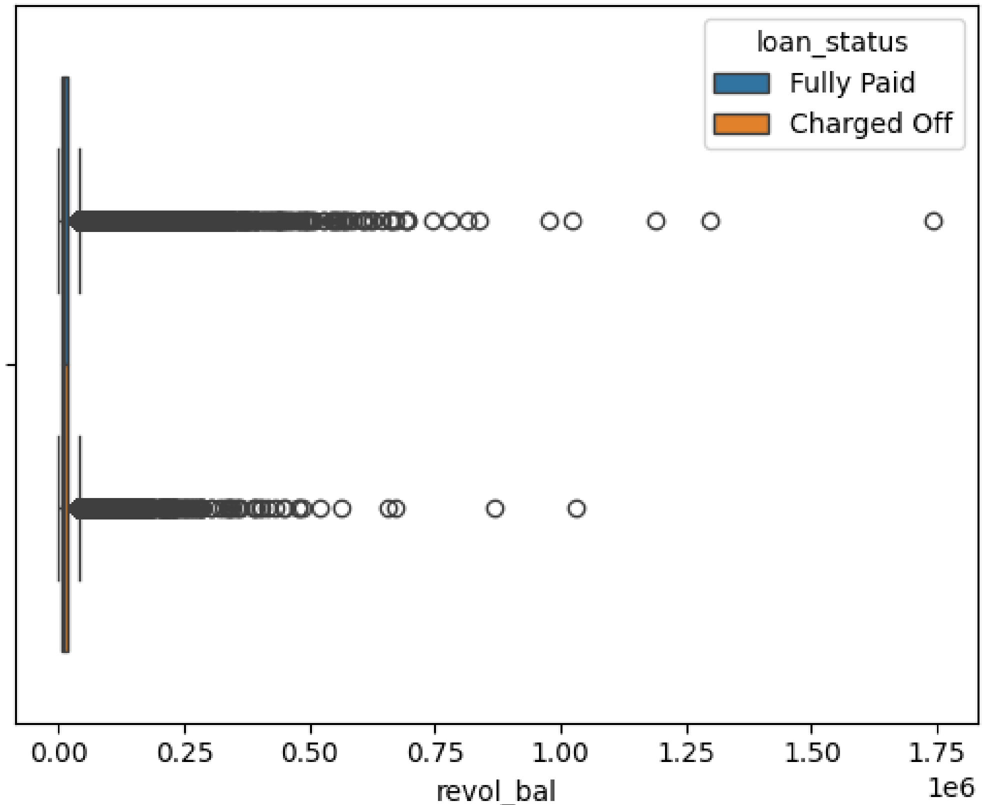

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



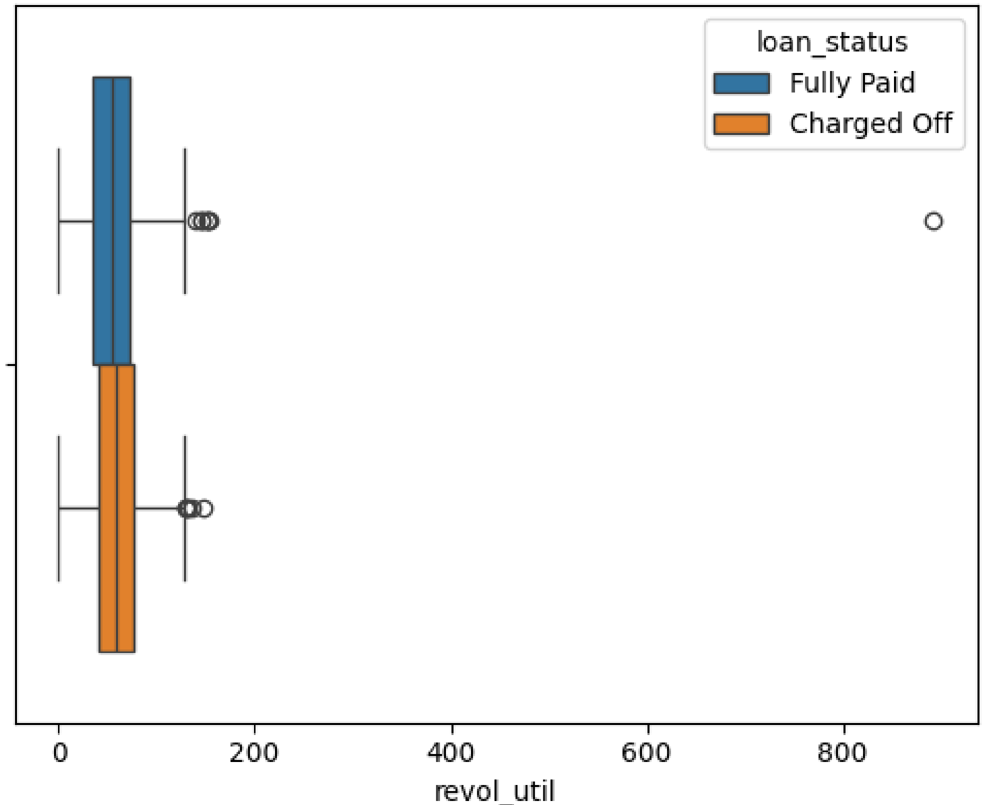

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



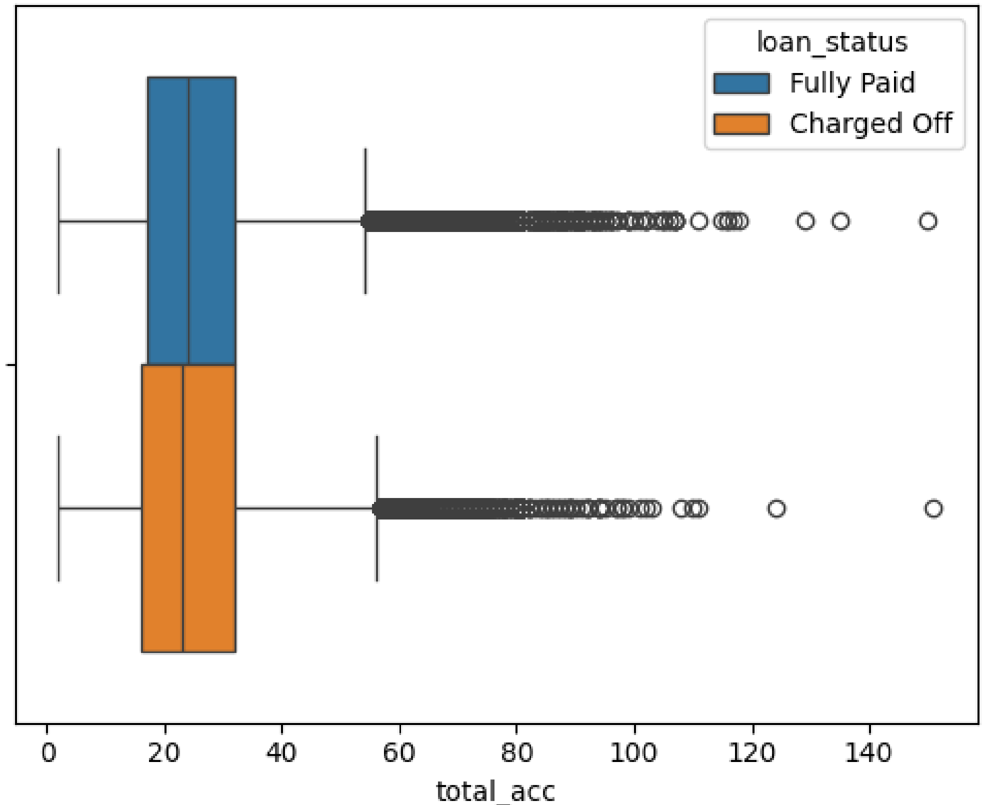

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



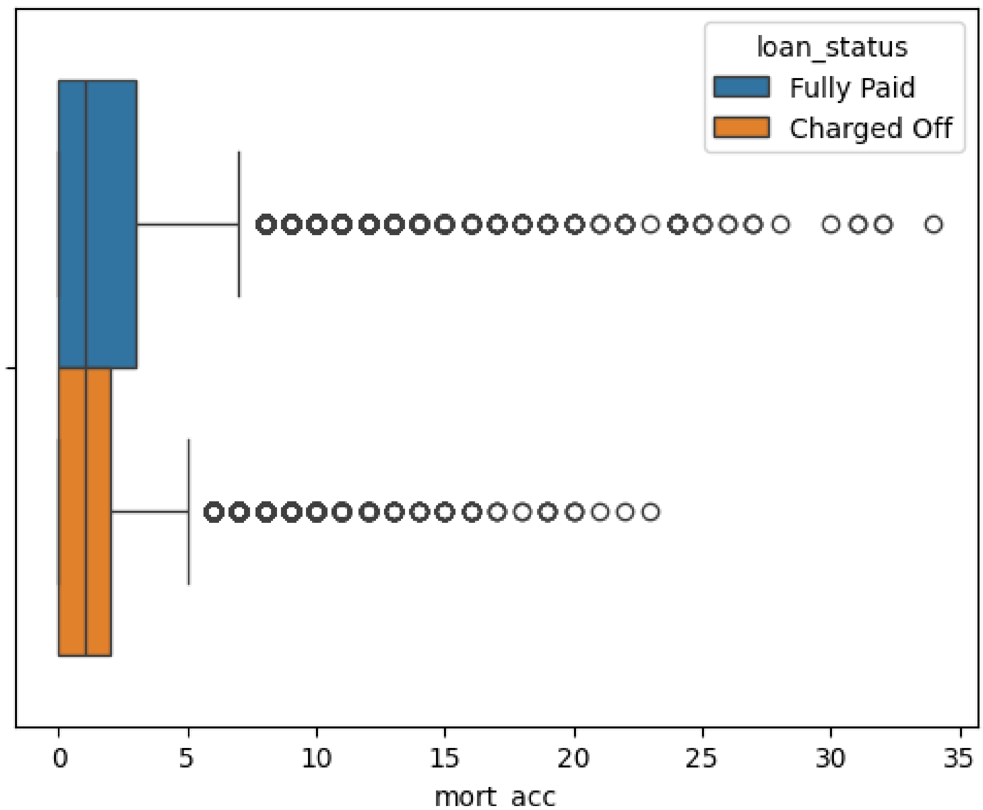

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



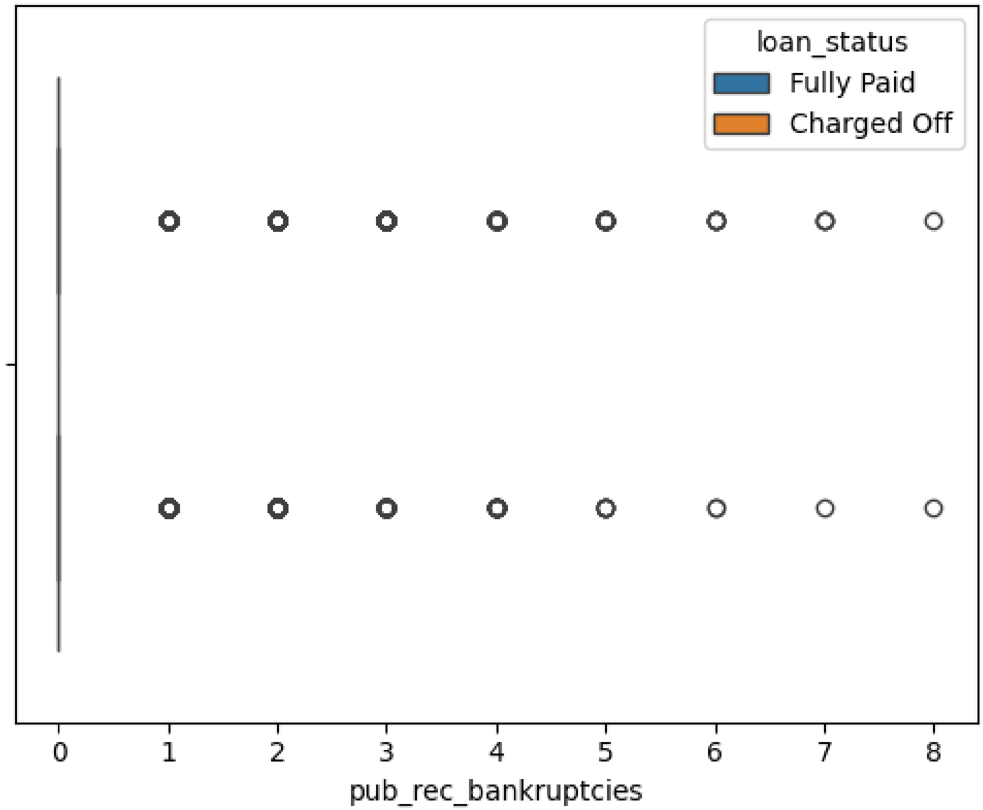

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



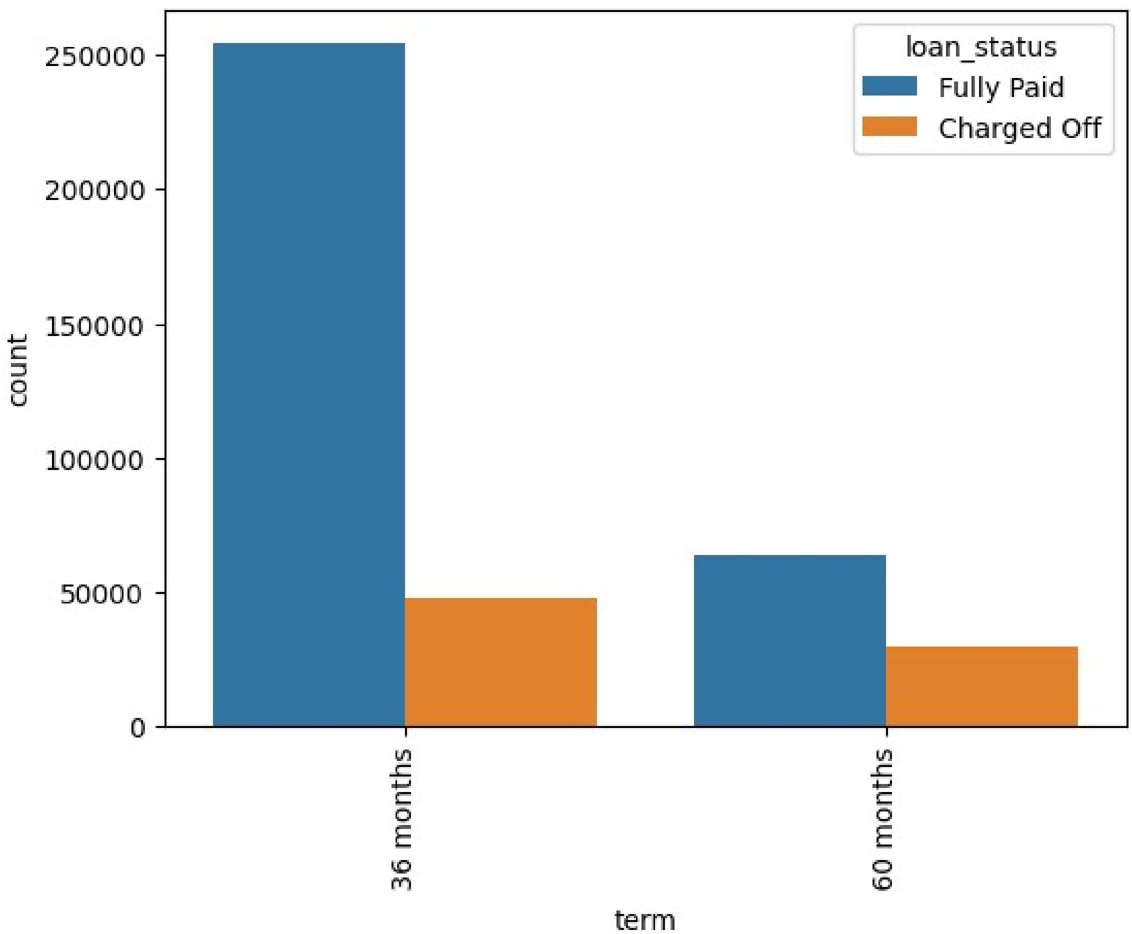

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:1608:
UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



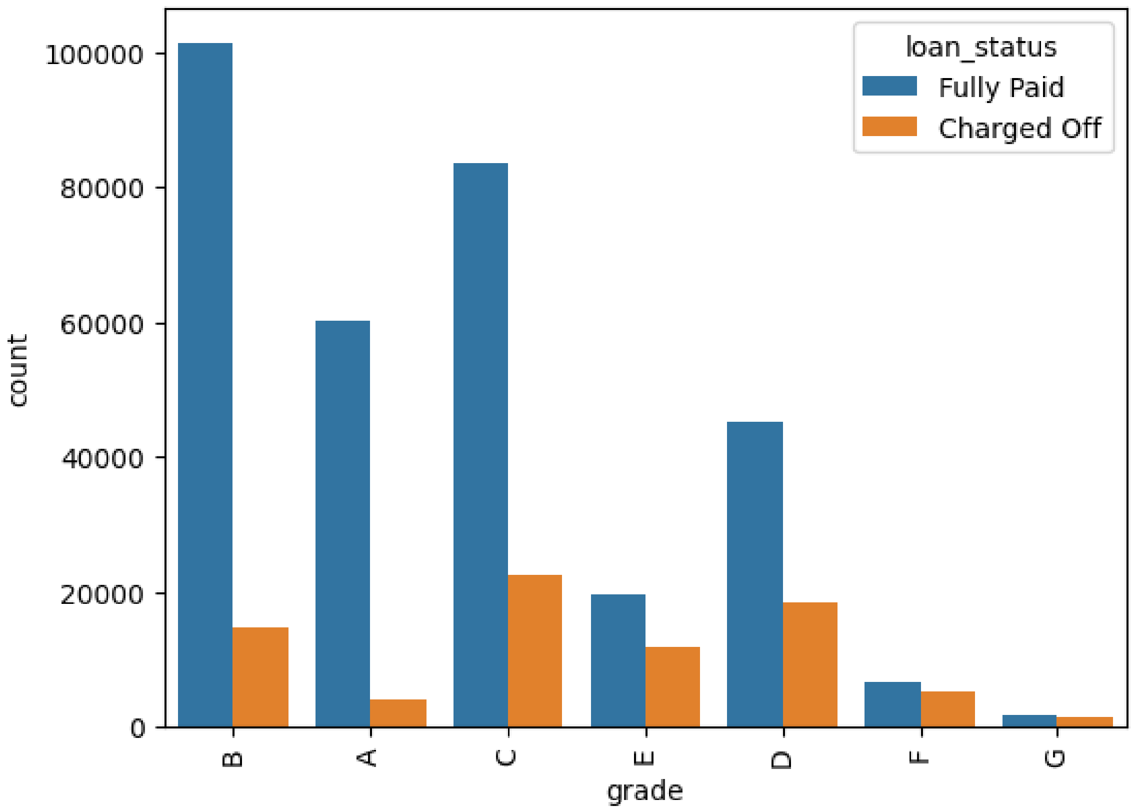
The median values for loan amount, installment, interest rate, no of active credit lines are slightly greater for the defaulters which makes sense as they couldn't pay off the loan.

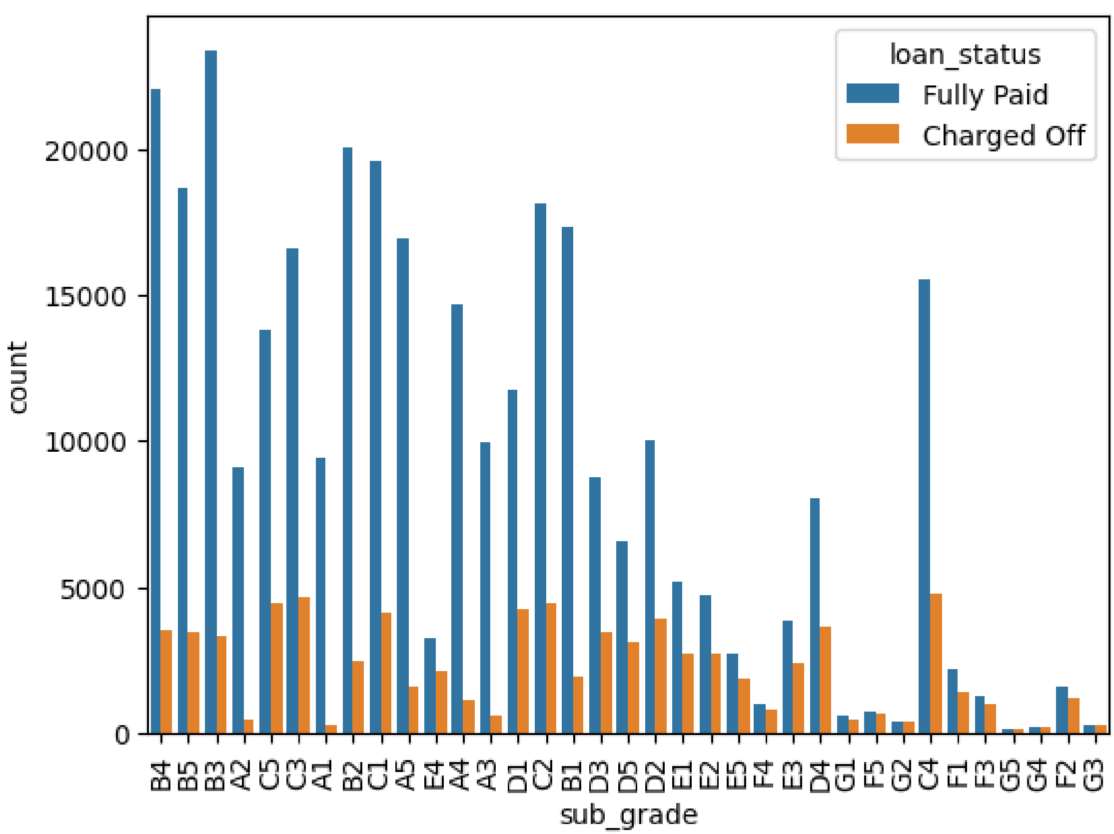
Target (Categorical) vs Features (Categorical)

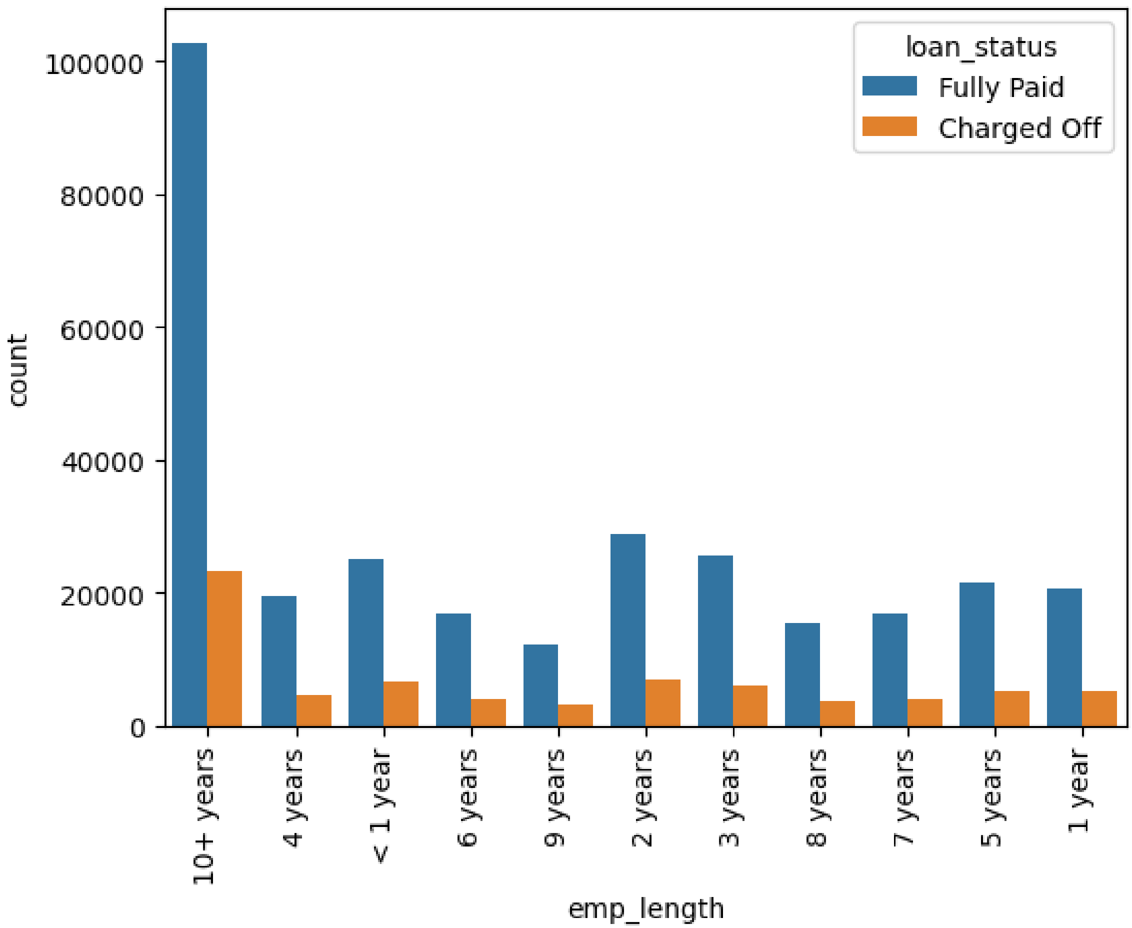
```
for col in cat_columns:
  if col not in ('emp_title', 'title'):
    sns.countplot(data = data, x = col, fill = 'True', hue =
'loan_status')
    plt.xticks(rotation = 90)
    plt.show()
```



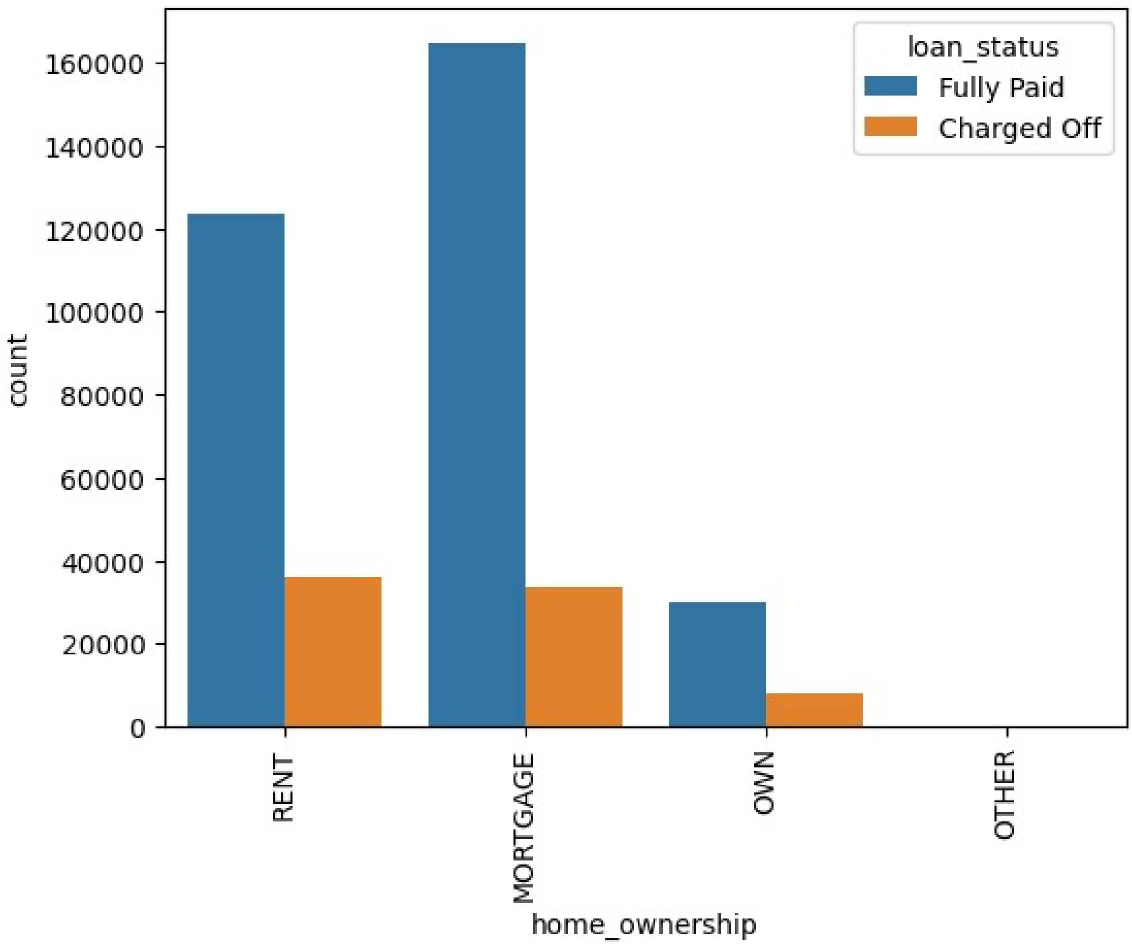
loan_status
Fully Paid
Charged Off
250000
200000
150000
100000
50000
0
count
36 months
60 months
term

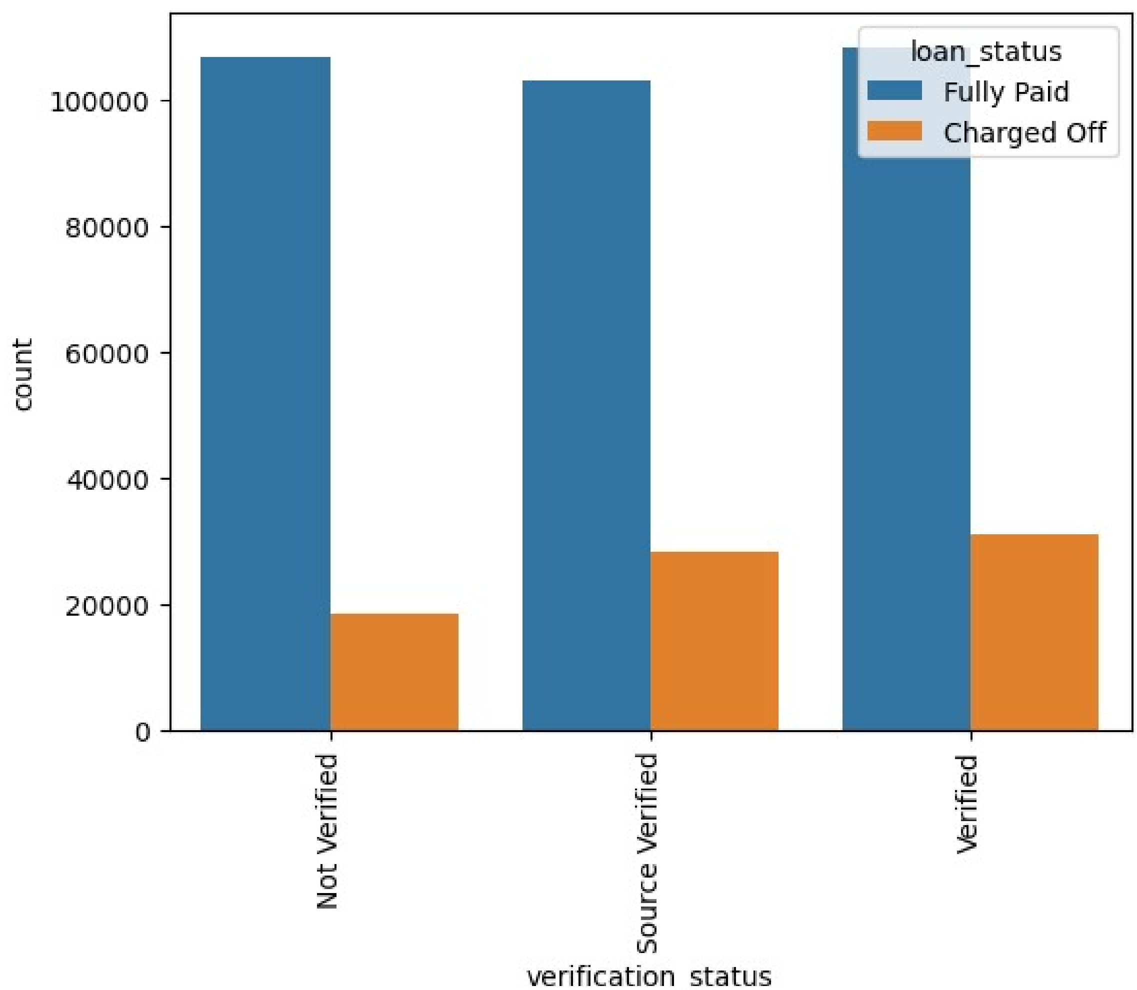

loan_status
Fully Paid
Charged Off
count
100000
80000
60000
40000
20000
0
B
A
C
E
D
F
G
grade

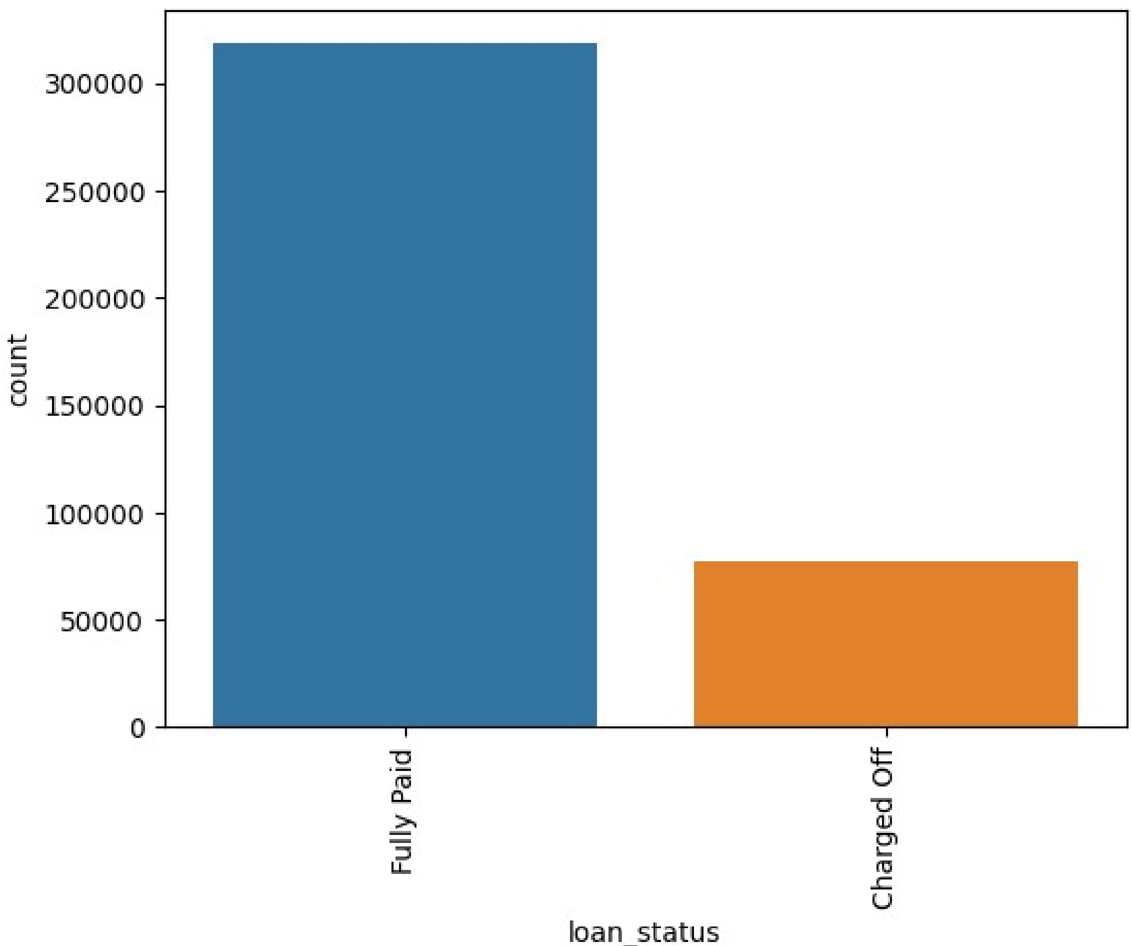

loan_status
Fully Paid
Charged Off
count
20000
15000
10000
5000
0
B4 B5 B3 A2 C5 C3 A1 B2 C1 A5 E4 A4 A3 D1 C2 B1 D3 D5 D2 E1 E2 E5 F4 E3 D4 G1 F5 G2 C4 F1 F3 G5 G4 F2 G3
sub_grade

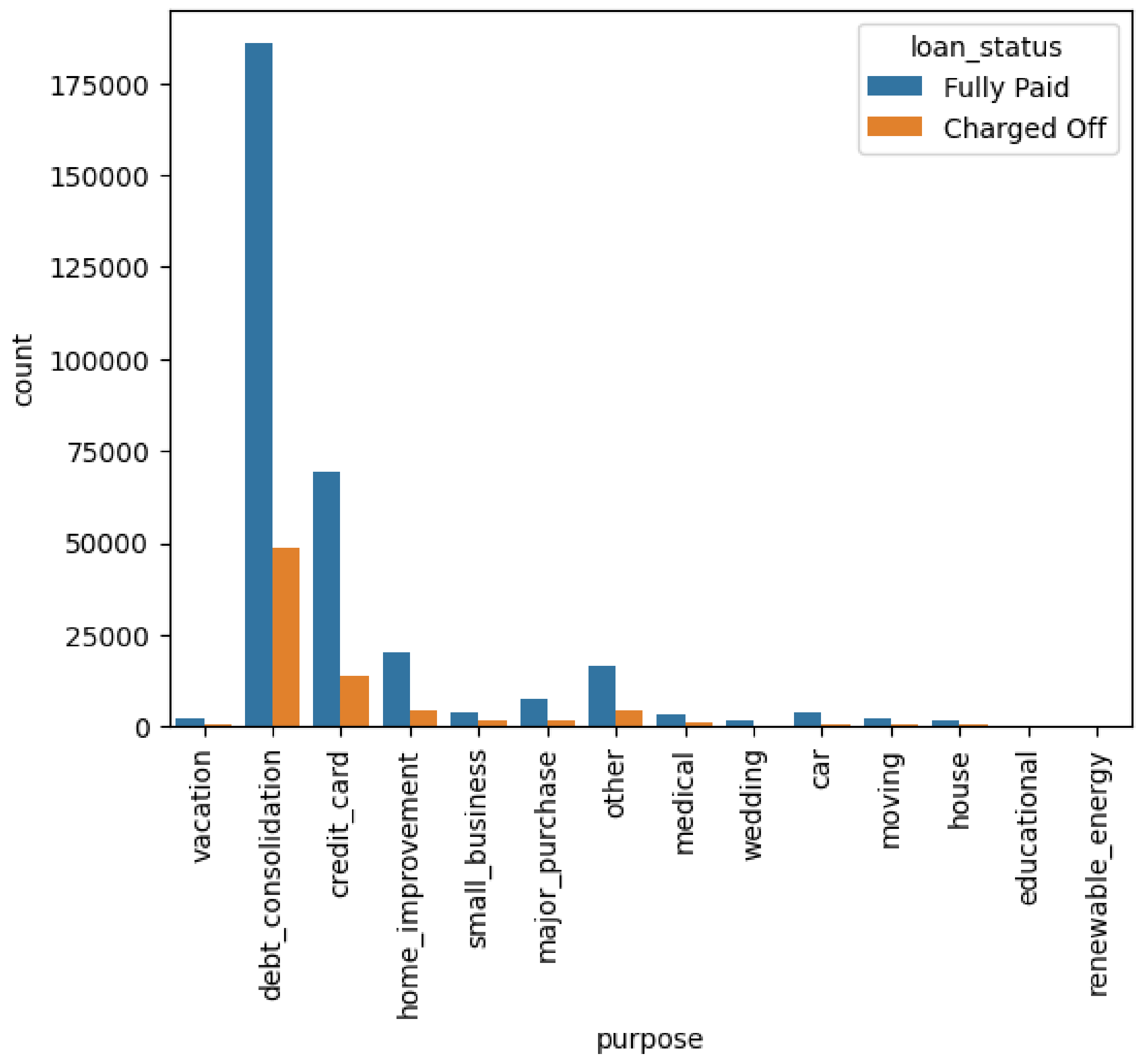


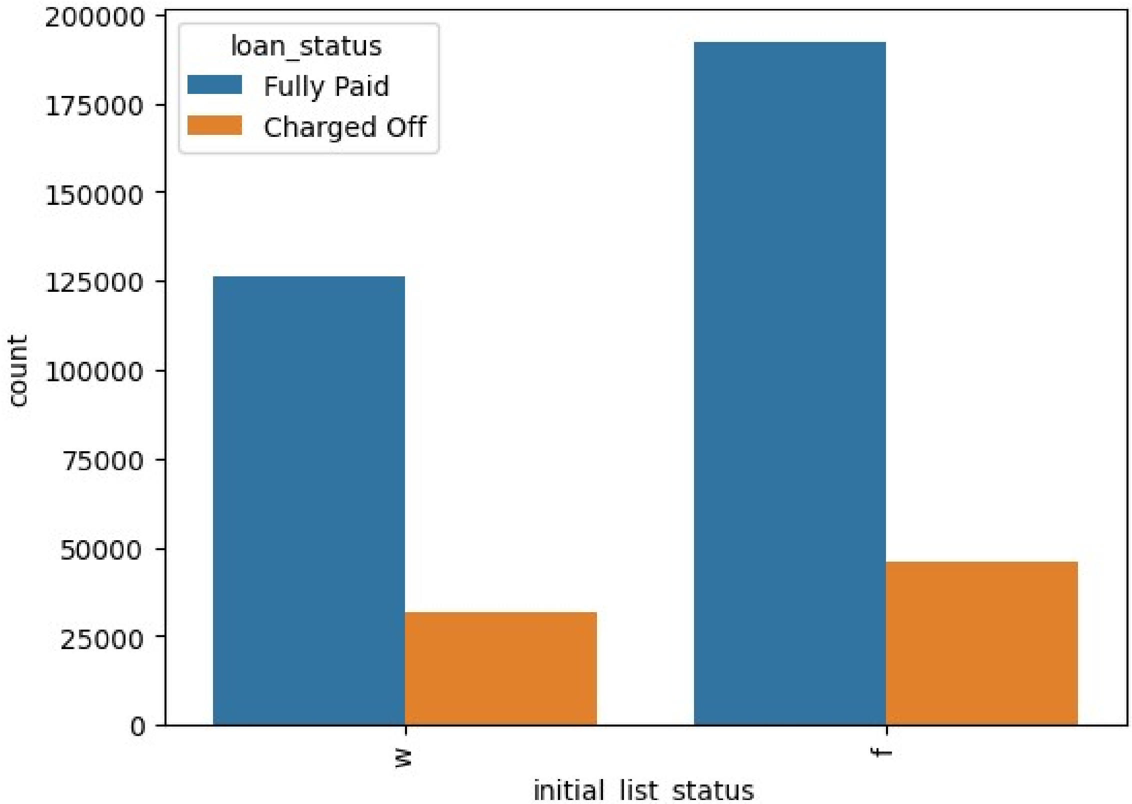
loan_status
Fully Paid
Charged Off
count
100000
80000
60000
40000
20000
0
10+ years
4 years
< 1 year
6 years
9 years
2 years
3 years
8 years
7 years
5 years
1 year
emp_length

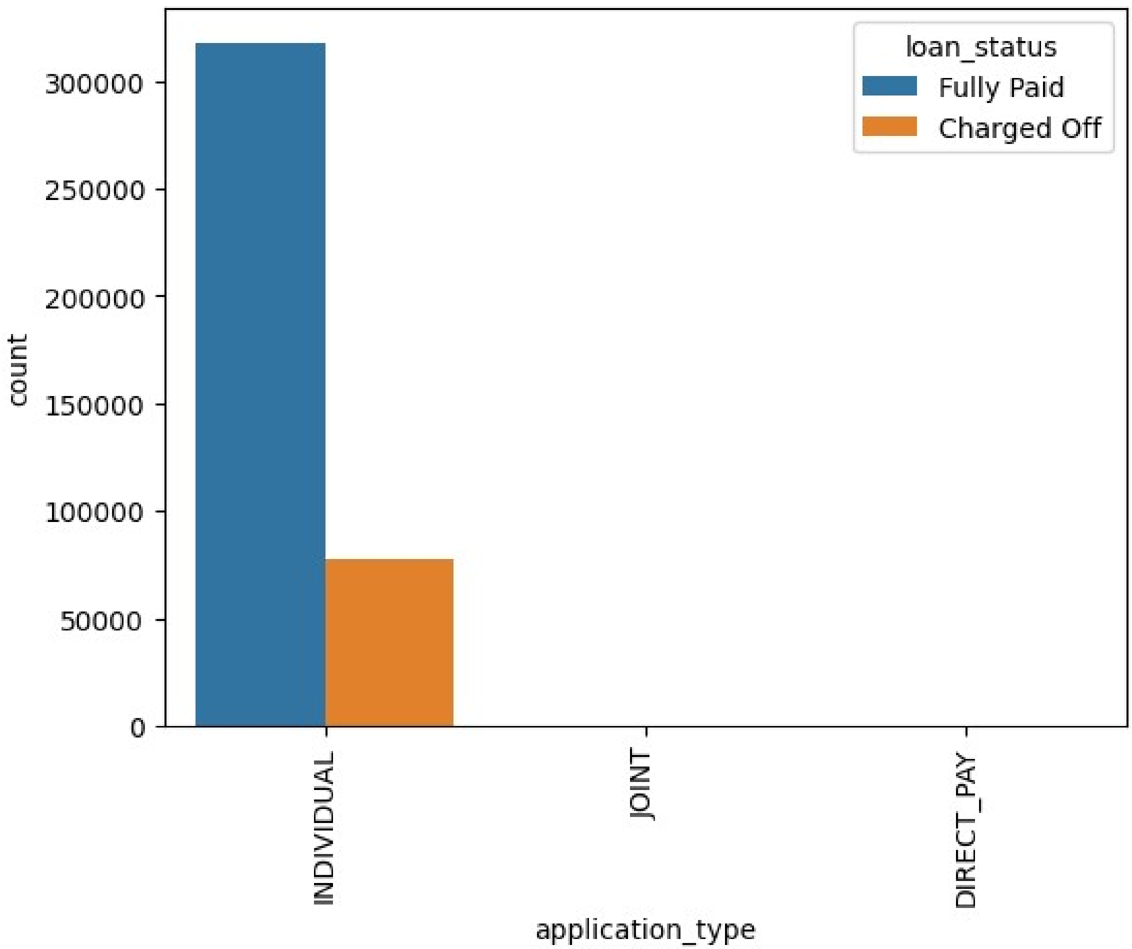

loan_status
Fully Paid
Charged Off
count
160000
140000
120000
100000
80000
60000
40000
20000
0
RENT
MORTGAGE
OWN
OTHER
home_ownership


loan_status
Fully Paid
Charged Off
count
0
20000
40000
60000
80000
100000
Not Verified
Source Verified
Verified
verification_status


300000
250000
200000
150000
100000
50000
0
count
Fully Paid
Charged Off
loan_status


loan_status
Fully Paid
Charged Off
count
175000
150000
125000
100000
75000
50000
25000
0
vacation
debt_consolidation
credit_card
home_improvement
small_business
major_purchase
other
medical
wedding
car
moving
house
educational
renewable_energy
purpose

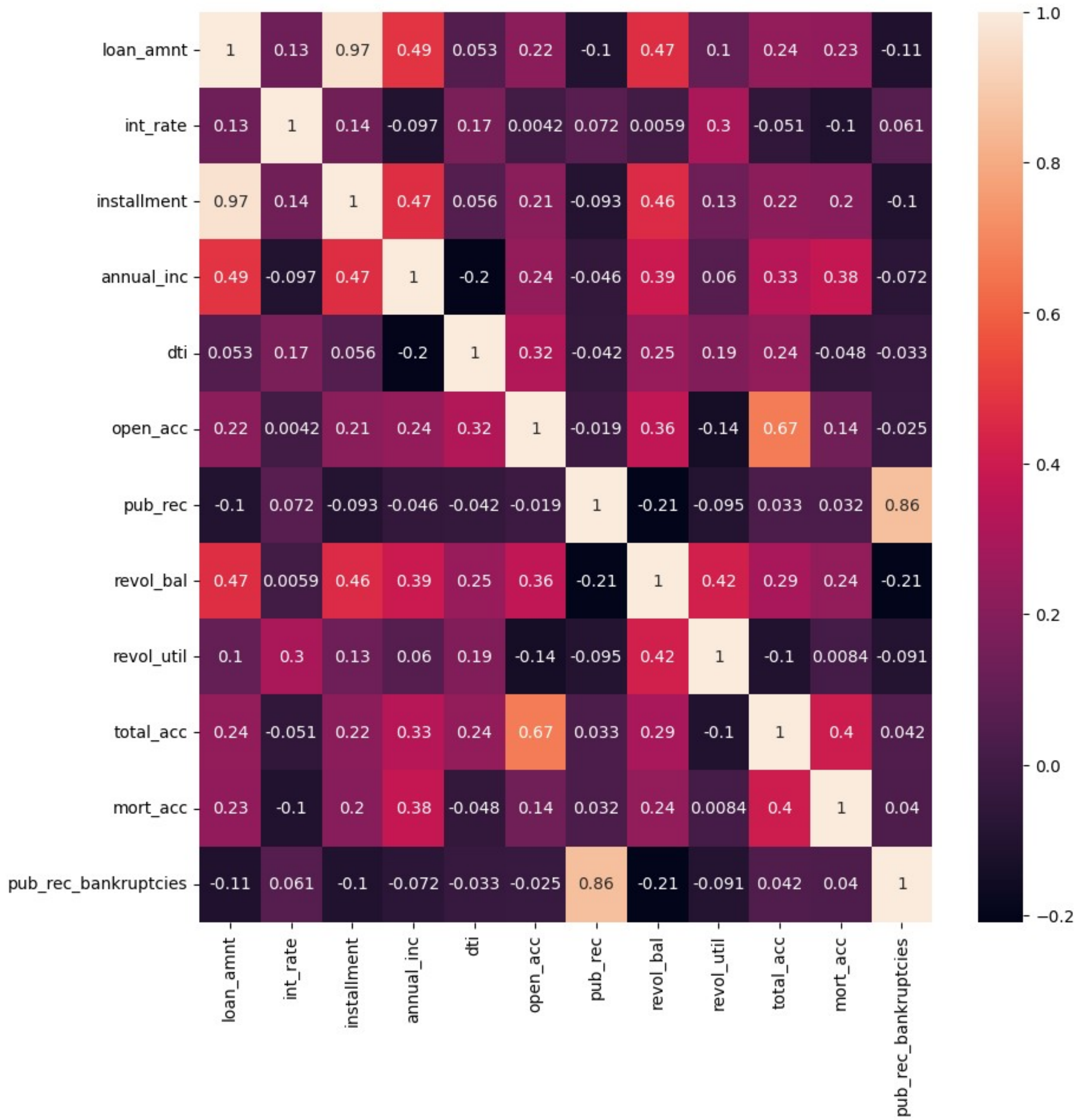

loan_status
Fully Paid
Charged Off
count
initial_list_status
w
f
0
25000
50000
75000
100000
125000
150000
175000
200000



1. For loan term '60 months', there is no huge difference between the no of defaulters and non-defaulters which means for greater duration the risk of defaulting is also high.
2. For the loan ratings 'A', 'B', 'C' and 'D', there's a huge difference between the no of defaulters and non-defaulters and no of non-defaulters is high which means more likely loan will be paid but for grades 'E', 'F' ad 'G', these nos are almost same that means high risk ratings.
3. Thus, we can say that grades 'A', 'B' and 'C' are low risk, grades 'D' and 'E' are moderate risk and 'F' and 'G' are high risk.
4. Similar pattern observed for the sub-grades with 1 being low risk in that grade and 5 being high risk.
5. So overall, A1 is lowest risk and G5 is the highest risk.
6. Among the borrowers with initial list status as 'w' i.e. whole is difference is lesser as compared to 'f' which means that more chance of defaulting when entire amount is approved.
7. Employee tenure, income verification status, purpose of loan, home ownership status and loan application type do not make much impact on defaulting.

Features : Continuous vs Continuous

```
num_cols = [col for col in data.columns if data[col].dtype !=
'object']

fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data[num_cols].corr(method = 'spearman'), annot = True, ax
= ax)
plt.show()
```



1. The correlation coefficient is the highest for loan amount and installment features which is +0.97 which tells us that greater the loan amount, greater will be the monthly installment amount.
2. Also, the correlation is quite high (+0.86) between pub_rec (Negative records on borrower's public credit profile) and pub_rec_bankruptcies (Bankruptcy records for borrower) which is justified as the no of bankruptcies increases, the no of negative records would also increase.
3. Another pair having higher correlation (+0.67) is open_acc (Number of borrower's active credit lines) and total_acc (Total number of borrower's credit lines)
4. There's negartive correlation (-0.1) between loan amount and pub_rec_bankruptcies or pub_rec as no of negative records or bankruptcies increases loan amount would decrease and it becomes quite risky to assume that borrower won't default.
5. Also, annual income and dti have negative correlation (-0.2) as monthly debt to monthly income ratio would decrease with increase in the income.

We noticed almost perfect correlation between "loan_amnt" the "installment" feature.

- installment: The monthly payment owed by the borrower if the loan originates.
- loan_amnt: The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

So, we can drop either one of those columns.

```
data.drop(columns=['installment'], axis = 1, inplace = True)
```

#Data Preprocessing

Duplicate value check

```
data.duplicated().sum()
```

```
0
```

There are no duplicate rows present in the dataset.

Missing Value Detection

```
data.isna().sum()
```

```
loan_amnt                   0
term                        0
int_rate                    0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
```

```
verification_status          0
issue_d                      0
loan_status                  0
purpose                      0
title                     1756
dti                          0
earliest_cr_line             0
open_acc                     0
pub_rec                      0
revol_bal                    0
revol_util                 276
total_acc                    0
initial_list_status          0
application_type             0
mort_acc                 37795
pub_rec_bankruptcies       535
address                      0
dtype: int64
```

```
data.isna().sum() * 100.0/len(data)
```

```
loan_amnt                0.000000
term                     0.000000
int_rate                 0.000000
grade                    0.000000
sub_grade                0.000000
emp_title                5.789208
emp_length               4.621115
home_ownership           0.000000
annual_inc               0.000000
verification_status      0.000000
issue_d                  0.000000
loan_status              0.000000
purpose                  0.000000
title                    0.443401
dti                      0.000000
earliest_cr_line         0.000000
open_acc                 0.000000
pub_rec                  0.000000
revol_bal                0.000000
revol_util               0.069692
total_acc                0.000000
initial_list_status      0.000000
application_type         0.000000
mort_acc                 9.543469
pub_rec_bankruptcies     0.135091
address                  0.000000
dtype: float64
```

There are missing values present in the columns:

1. emp_title
2. emp_length
3. title
4. revol_util
5. mort_acc
6. pub_rec_bankruptcies

Missing Value Treatment

We will be using Simple Imputer to fill the missing values by using 'Most Frequent' strategy for the categorical variables such as emp_title, title and emp_length.

```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data[['emp_title_filled', 'emp_length_filled', 'title_filled']] =
imp.fit_transform(data[['emp_title', 'emp_length', 'title']])
```

Checking if missing values got filled correctly for categorical variables emp_title, title and emp_length.

```
data[['emp_title']].mode()
```

{"summary":"{\n \"name\": \"data[['emp_title']]\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"emp_title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"Teacher\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
data[['emp_length']].mode()
```

{"summary":"{\n \"name\": \"data[['emp_length']]\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"emp_length\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"10+ years\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
data[['title']].mode()
```

{"summary":"{\n \"name\": \"data[['title']]\",\n \"rows\": 1,\n \"fields\": [\n   {\n   \"column\": \"title\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"debt consolidation\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
data[data[['title']].isna().any(axis = 1)][['emp_title',
'emp_title_filled', 'emp_length', 'emp_length_filled', 'title',
'title_filled']]
```

{"repr_error":"0","type":"dataframe"}

Using strategy 'Median' for the continuous variables like revol_util, mort_acc and pub_rec_bankruptcies.

```
imp = SimpleImputer(missing_values=np.nan, strategy='median')
data[['mort_acc_filled', 'revol_util_filled',
'pub_rec_bankruptcies_filled']] = imp.fit_transform(data[['mort_acc',
'revol_util', 'pub_rec_bankruptcies']])

data[['mort_acc']].median()

mort_acc    1.0
dtype: float64

data[['revol_util']].median()

revol_util    54.8
dtype: float64

data[['pub_rec_bankruptcies']].median()

pub_rec_bankruptcies    0.0
dtype: float64
```

Drop original columns for these 6 that we filled and rename the _filled ones with the original names.

```
data.drop(['emp_title', 'emp_length', 'title', 'mort_acc',
'revol_util', 'pub_rec_bankruptcies'], axis =1, inplace = True)

data.isna().sum()

loan_amnt                      0
term                           0
int_rate                       0
grade                          0
sub_grade                      0
home_ownership                 0
annual_inc                     0
verification_status            0
issue_d                        0
loan_status                    0
purpose                        0
dti                            0
earliest_cr_line               0
open_acc                       0
pub_rec                        0
revol_bal                      0
total_acc                      0
```

```
initial_list_status             0
application_type                0
address                         0
emp_title_filled                0
emp_length_filled               0
title_filled                    0
mort_acc_filled                 0
revol_util_filled               0
pub_rec_bankruptcies_filled     0
dtype: int64
```

```
data = data.rename(columns = {'emp_title_filled': 'emp_title',
'emp_length_filled' : 'emp_length', 'title_filled' : 'title',
'mort_acc_filled' : 'mort_acc', 'revol_util_filled' : 'revol_util',
'pub_rec_bankruptcies_filled' : 'pub_rec_bankruptcies'})
```

Outlier Detection and Treatment

Using IQR method to detect and treat the outliers.

```
#Calculating few more statistical measures such as 'Range', 'IQR',
'Lower Whisker' and 'Upper Whisker'

descriptive_stats = data.describe()
descriptive_stats =
descriptive_stats.reindex(descriptive_stats.index.values.tolist()+
['Range', 'IQR', 'Lower Whisker', 'Upper Whisker'])

for col in descriptive_stats.columns:
  descriptive_stats.loc['Range'][col] = descriptive_stats.loc['max']
[col] - descriptive_stats.loc['min'][col]
  descriptive_stats.loc['IQR'][col] = descriptive_stats.loc['75%']
[col] - descriptive_stats.loc['25%'][col]
  descriptive_stats.loc['Lower Whisker'][col] =
descriptive_stats.loc['25%'][col] - (1.5 *
descriptive_stats.loc['IQR'][col])
  descriptive_stats.loc['Upper Whisker'][col] =
descriptive_stats.loc['75%'][col] + (1.5 *
descriptive_stats.loc['IQR'][col])

descriptive_stats
```

{"summary":"{\n \"name\": \"descriptive_stats\",\n \"rows\": 12,\n \"fields\": [\n   {\n   \"column\": \"loan_amnt\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 110649.59456444613,\n   \"min\": -10000.0,\n   \"max\": 396030.0,\n   \"num_unique_values\": 11,\n   \"samples\": [\n   12000.0,\n   396030.0,\n   -10000.0\n   ],\n   \"semantic_type\": \"\",\n

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"int_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 114319.98871432777,\n        \"min\":
1.4900000000000038,\n        \"max\": 396030.0,\n
\"num_unique_values\": 12,\n        \"samples\": [\n
1.4900000000000038,\n          5.999999999999998,\n          396030.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"annual_inc\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3355267.087918833,\n        \"min\": -
22500.0,\n        \"max\": 8706582.0,\n        \"num_unique_values\":
10,\n        \"samples\": [\n          -22500.0,\n
74203.17579771738,\n          64000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"dti\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 113860.8635870796,\n
\"min\": -6.270000000000001,\n        \"max\": 396030.0,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          16.91,\n
396030.0,\n          -6.270000000000001\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"open_acc\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
114317.28801162555,\n        \"min\": -1.0,\n        \"max\":
396030.0,\n        \"num_unique_values\": 11,\n        \"samples\": [\
n          10.0,\n          396030.0,\n          -1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"pub_rec\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
114319.48592479543,\n        \"min\": 0.0,\n        \"max\":
396030.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.17819104613286874,\n          86.0,\n          0.530670600474012\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"revol_bal\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
667965.7781961308,\n        \"min\": -14367.5,\n        \"max\":
1743266.0,\n        \"num_unique_values\": 11,\n        \"samples\":
[\n          11181.0,\n          396030.0,\n          -14367.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_acc\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
114311.52591669752,\n        \"min\": -5.5,\n        \"max\":
396030.0,\n        \"num_unique_values\": 12,\n        \"samples\": [\
n          -5.5,\n          15.0,\n          396030.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mort_acc\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
114321.86775896826,\n        \"min\": -4.5,\n        \"max\":
396030.0,\n        \"num_unique_values\": 9,\n        \"samples\": [\n
-4.5,\n          1.736307855465495,\n          3.0\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"revol_util\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
114267.48240899428,\n        \"min\": -19.600000000000016,\n
\"max\": 396030.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          54.8,\n          396030.0,\n          -
19.600000000000016\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pub_rec_bankruptcies\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 114323.58117393513,\n        \"min\":
0.0,\n        \"max\": 396030.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.12148322096810847,\n          8.0,\n
0.35596165879827396\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"descriptive_stats"}
```


No of outliers present in the respective column

```
for col in descriptive_stats.columns:
  if col not in cat_columns:
    print(col, ':', data[(data[col] < descriptive_stats.loc['Lower
Whisker'][col]) | (data[col] > descriptive_stats.loc['Upper Whisker']
[col])][col].count())
```

```
loan_amnt : 191
int_rate : 3777
annual_inc : 16700
dti : 275
open_acc : 10307
pub_rec : 57758
revol_bal : 21259
total_acc : 8499
mort_acc : 6843
revol_util : 12
pub_rec_bankruptcies : 45115
```

```
for col in descriptive_stats.columns:
  if col not in cat_columns:
    print(col, ':', data[(data[col] < descriptive_stats.loc['Lower
Whisker'][col]) | (data[col] > descriptive_stats.loc['Upper Whisker']
[col])][col].count()/len(data))
```

```
loan_amnt : 0.0004822866954523647
int_rate : 0.009537156276039694
annual_inc : 0.042168522586672724
dti : 0.0006943918390020958
open_acc : 0.02602580612579855
pub_rec : 0.145842486680302
revol_bal : 0.05368027674671111
total_acc : 0.021460495417013864
```

```
mort_acc : 0.017278994015604877
revol_util : 3.0300734792818725e-05
pub_rec_bankruptcies : 0.11391813751483473
```

We can see that pub_rec and pub_rec_bankruptcies have highest % of outliers among others.

```
for col in data.columns:
    if (col not in cat_columns) and (col not in
descriptive_stats.columns):
        print(col)
```

```
issue_d
earliest_cr_line
address
```

Clipping outliers to LW if < min and UW if > max.

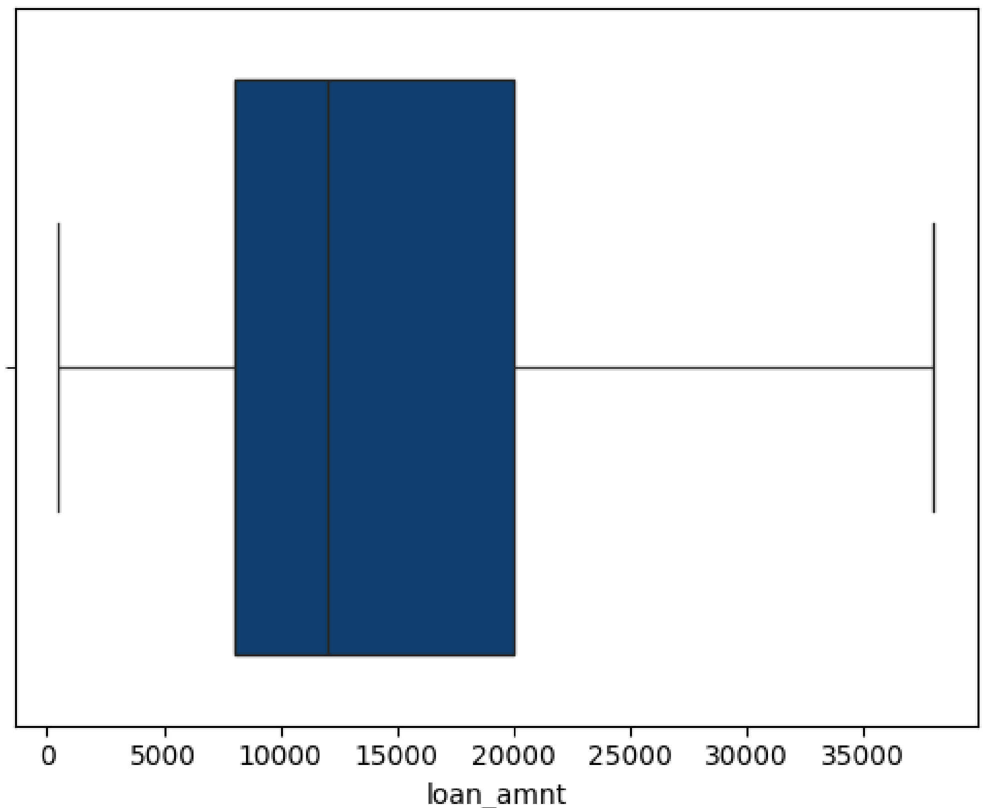
```
for col in descriptive_stats.columns:
  if col not in cat_columns and col not in ('pub_rec', 'mort_acc',
'pub_rec_bankruptcies'):
    min = descriptive_stats.loc['Lower Whisker'][col]
    max = descriptive_stats.loc['Upper Whisker'][col]

    data.loc[data[col] < descriptive_stats.loc['Lower Whisker'][col],
col] = min
    data.loc[data[col] > descriptive_stats.loc['Upper Whisker'][col],
col] = max
```

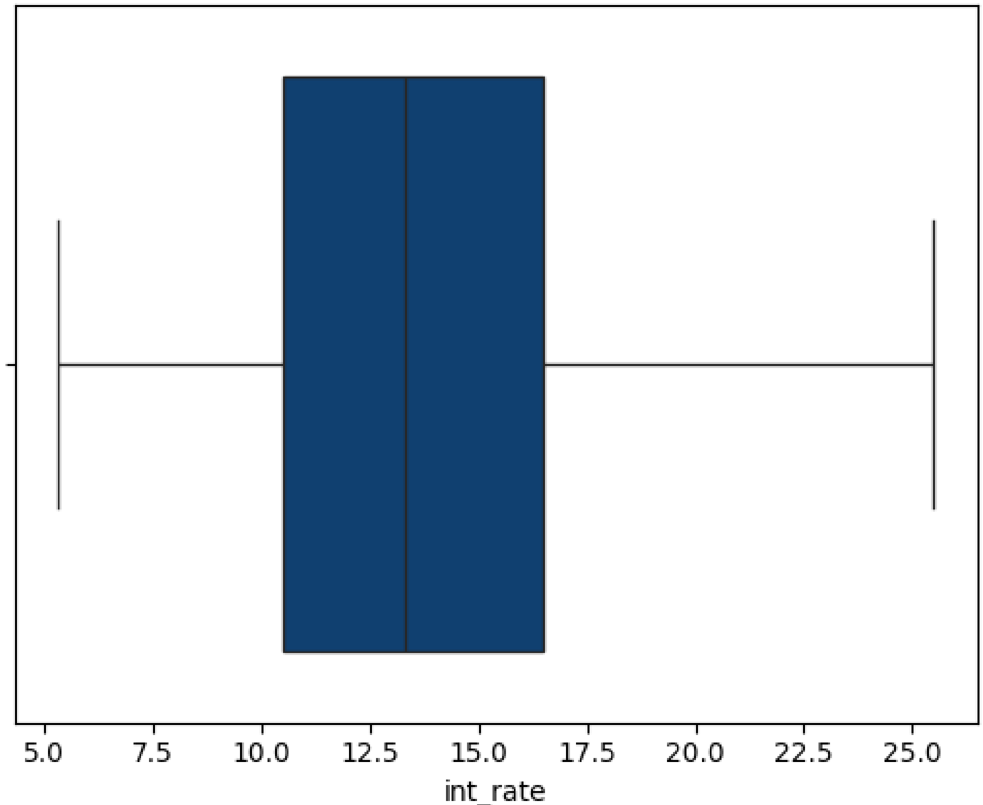
Not dealing with the outliers present in pub_rec, mort_acc and pub_rec_bankruptcies as we are going to convert them to categorical columns by creating flags based on condition if > 1 then 1 else 0.

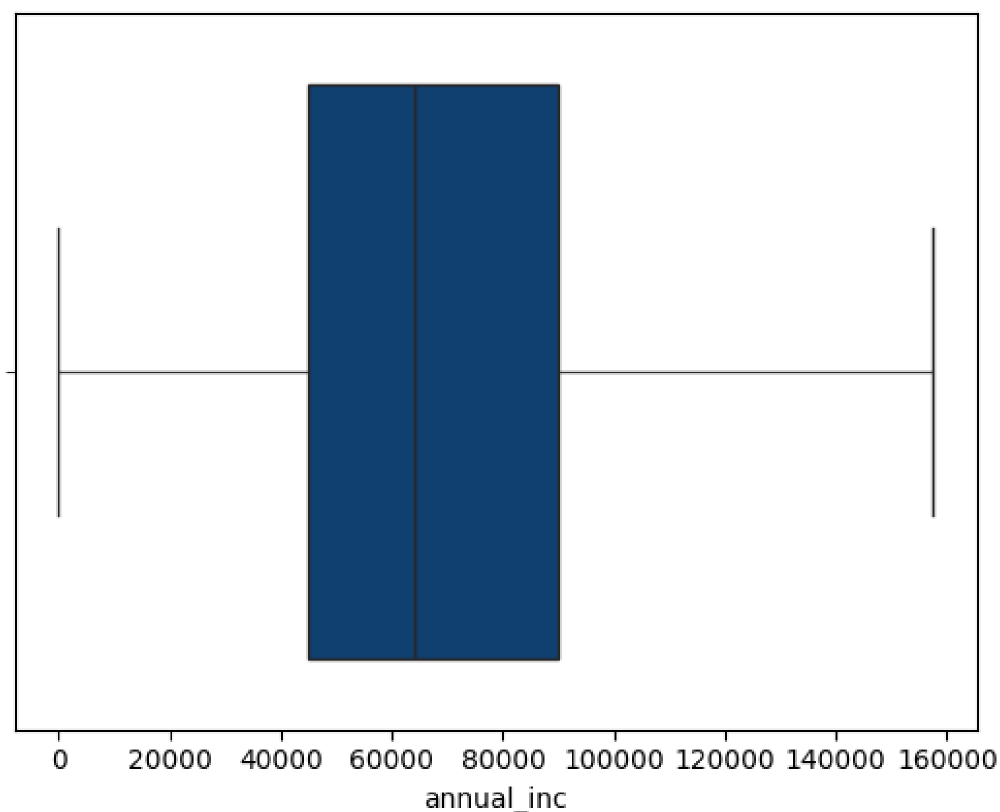
```
for col in descriptive_stats.columns:
  if col not in cat_columns:
    print(col, ':', data[(data[col] < descriptive_stats.loc['Lower
Whisker'][col]) | (data[col] > descriptive_stats.loc['Upper Whisker']
[col])][col].count())
```

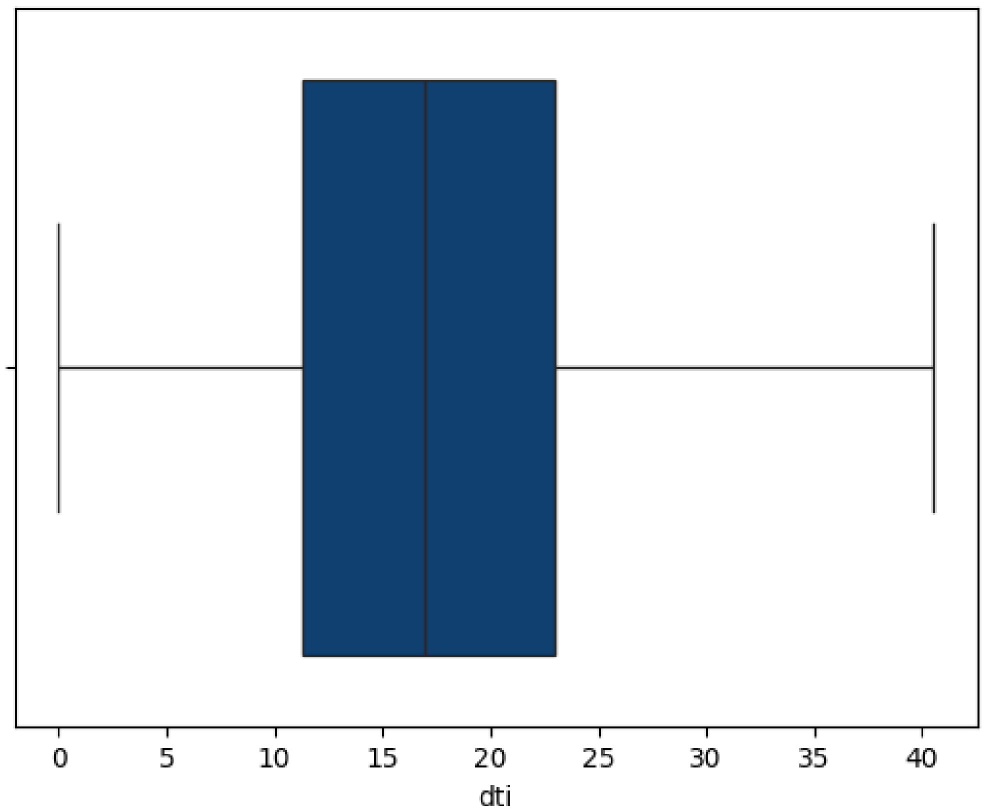
```
loan_amnt : 0
int_rate : 0
annual_inc : 0
dti : 0
open_acc : 0
pub_rec : 57758
revol_bal : 0
total_acc : 0
mort_acc : 6843
revol_util : 0
pub_rec_bankruptcies : 45115
```

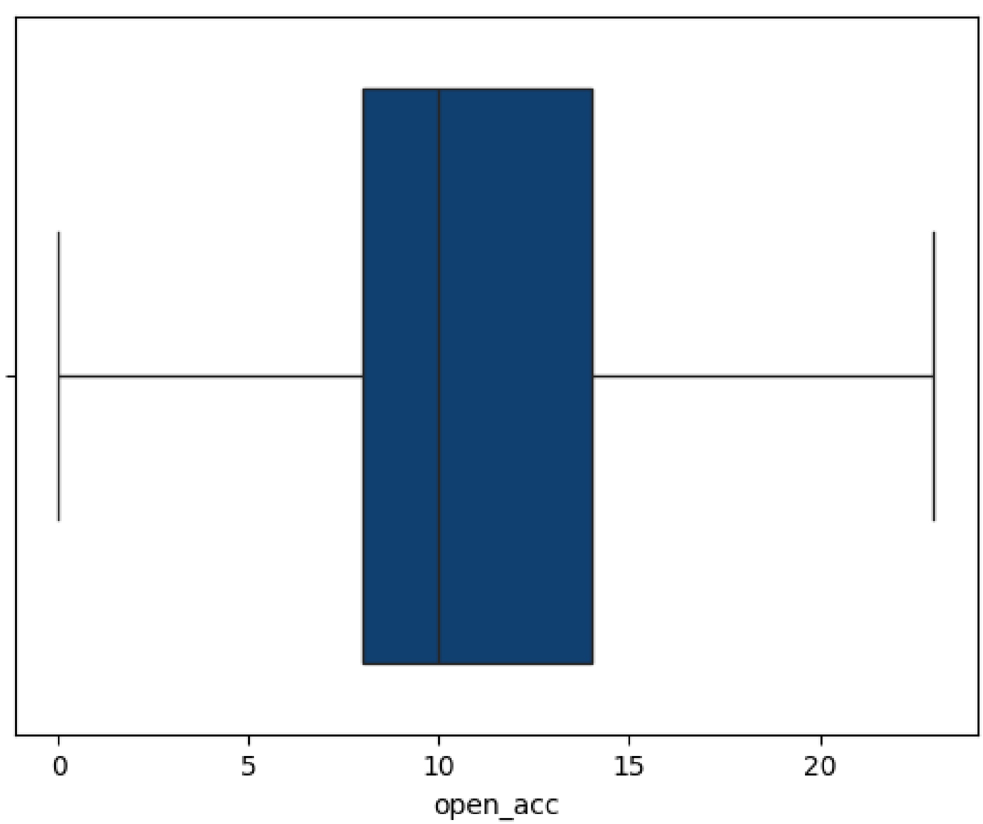
```
for col in data.columns:
  if data[col].dtype != 'object':
    sns.boxplot(data = data, x = col, palette = 'ocean')
    plt.show()
```

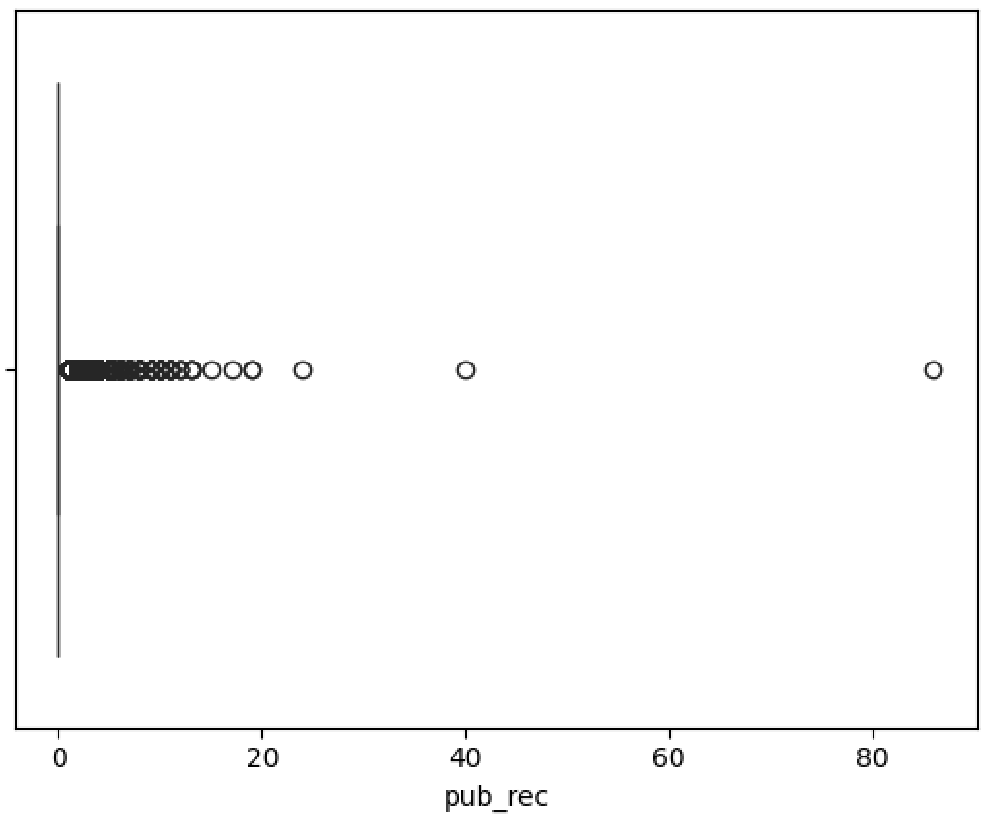


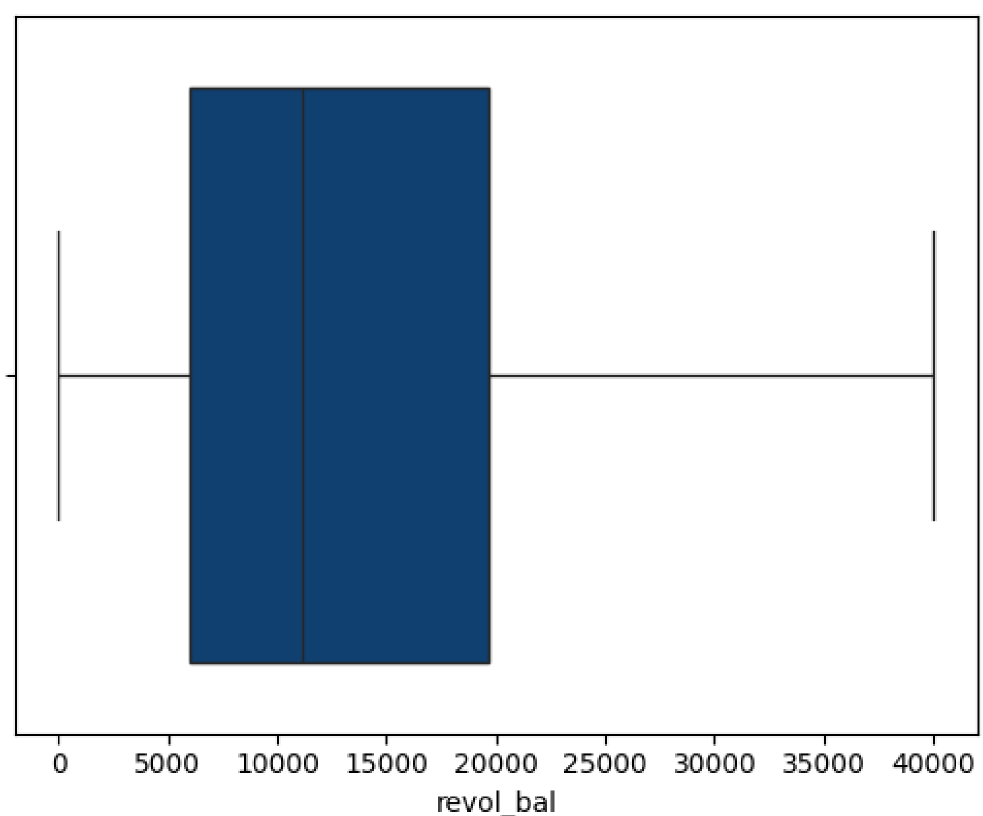

5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
int_rate

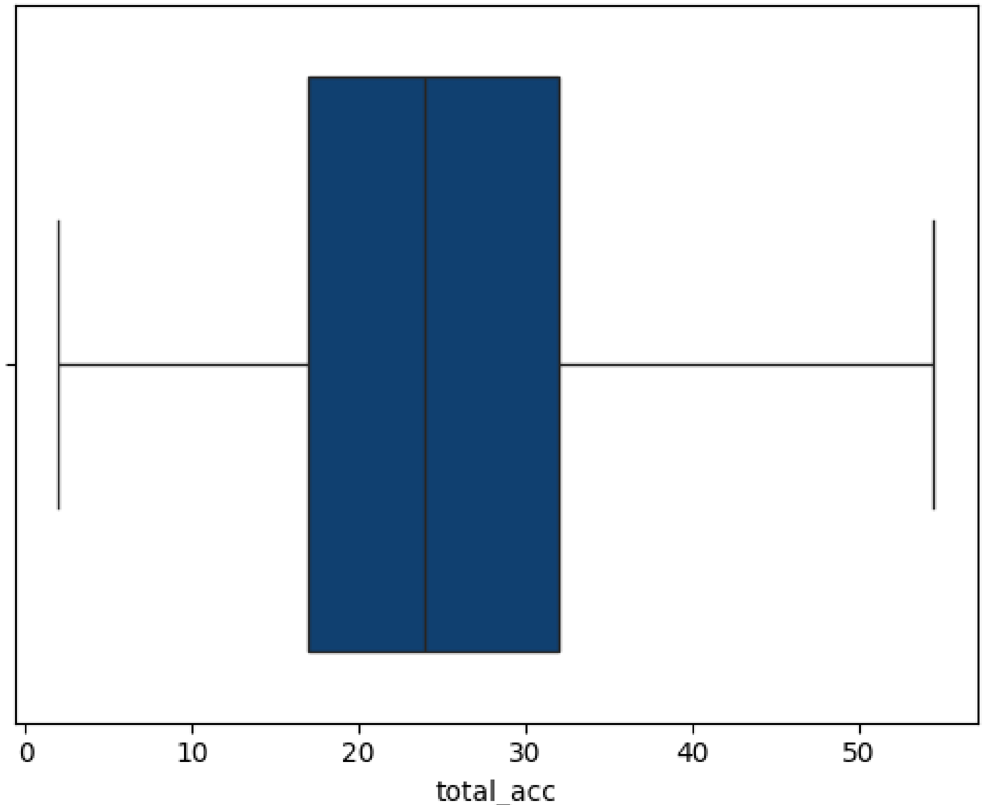

0
20000
40000
60000
80000
100000
120000
140000
160000
annual_inc

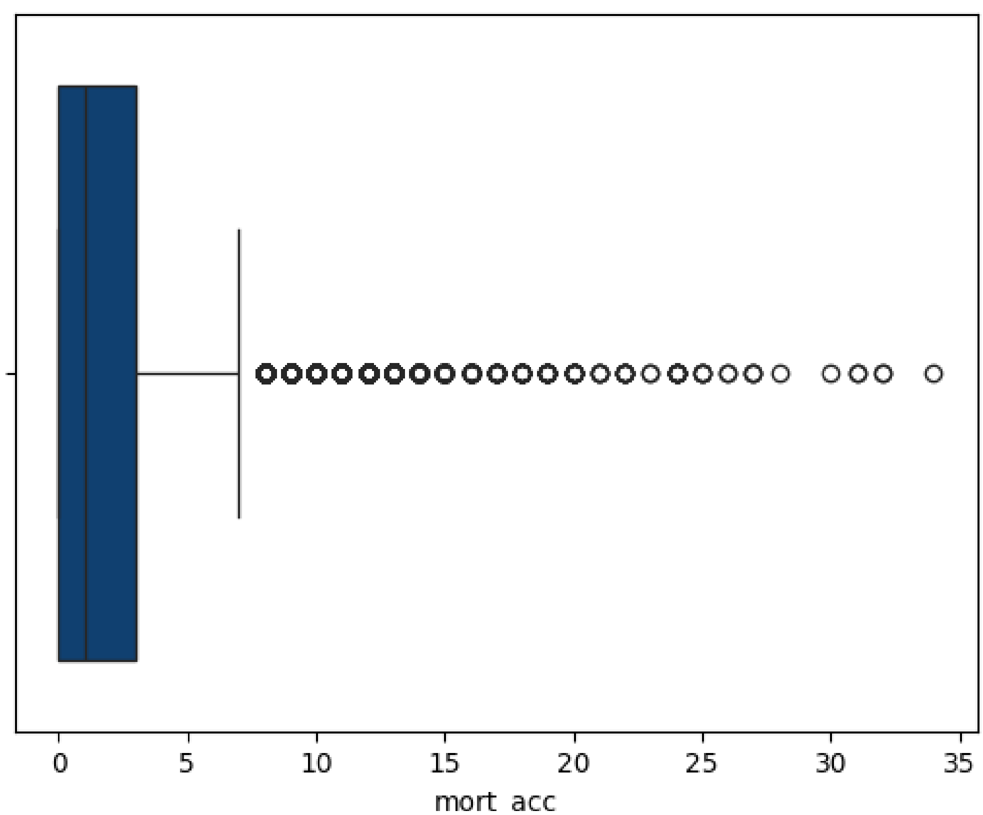

0
5
10
15
20
25
30
35
40
dti

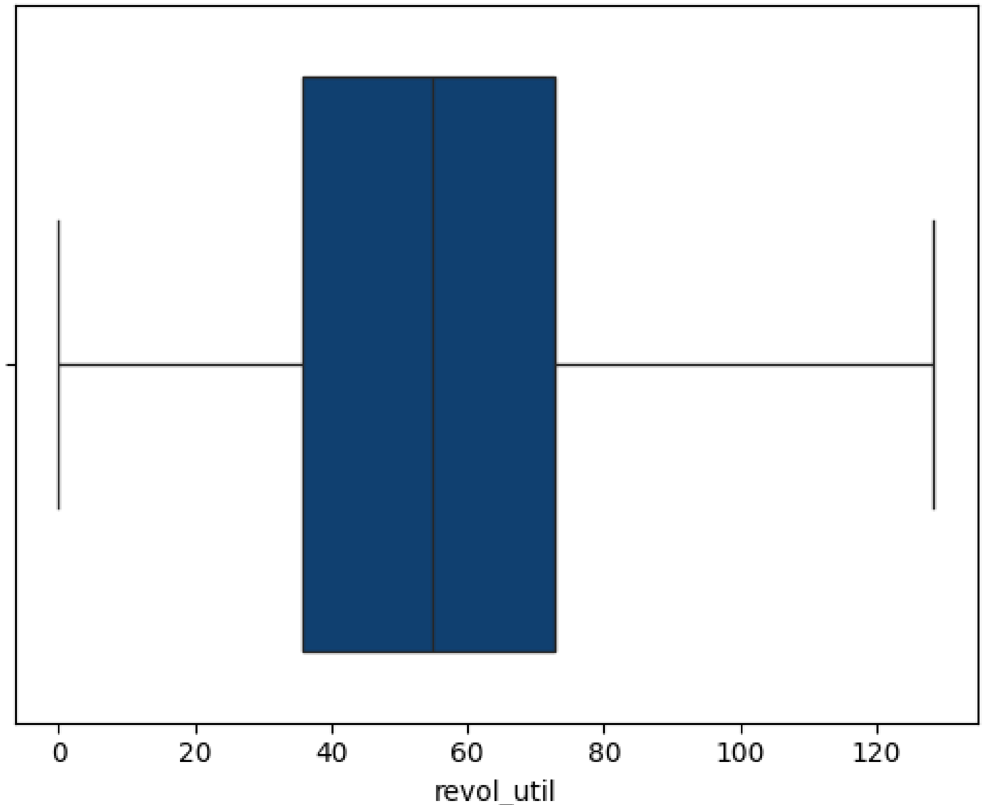

0
5
10
15
20
open_acc

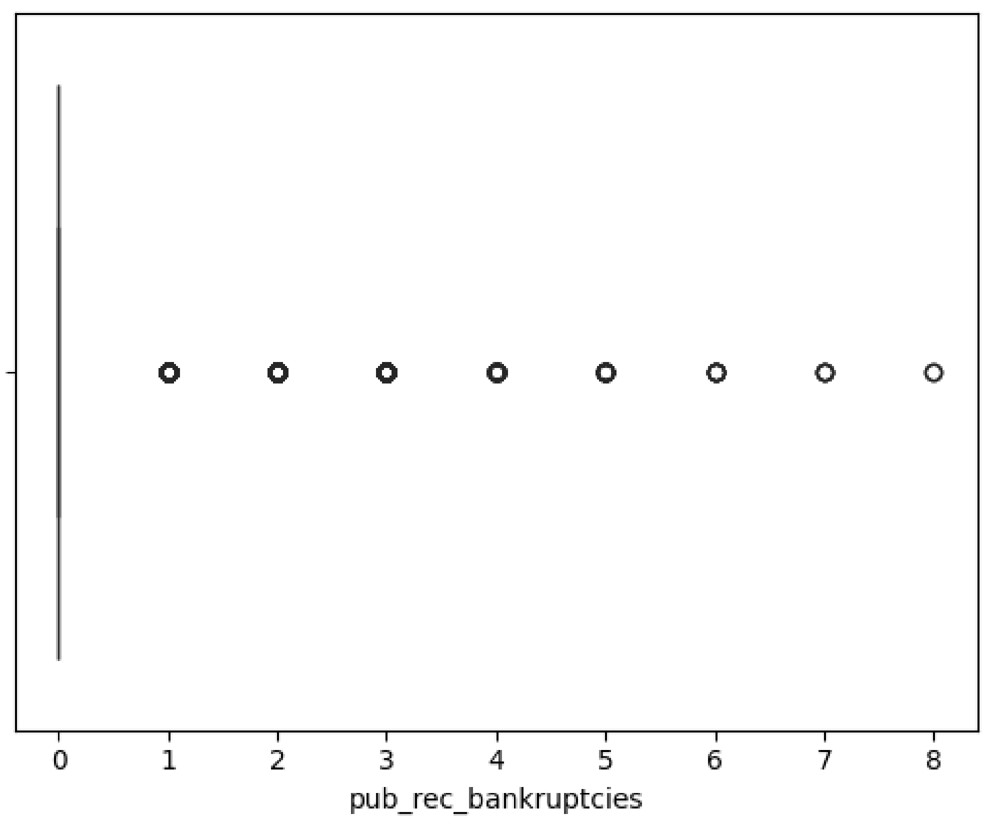

0
20
40
60
80
pub_rec


0
5000
10000
15000
20000
25000
30000
35000
40000
revol_bal


0
10
20
30
40
50
total_acc


0
5
10
15
20
25
30
35
mort_acc


0
20
40
60
80
100
120
revol_util


0
1
2
3
4
5
6
7
8
pub_rec_bankruptcies

```
data[cat_columns]
```

{"type":"dataframe"}

Feature Engineering

New Feature: is_defaulter

```
data.loc[data['loan_status'] == 'Charged Off', 'is_defaulter'] = 1
data.loc[data['loan_status'] == 'Fully Paid', 'is_defaulter'] = 0

data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data['is_defaulter'].value_counts()
```

```
is_defaulter
0.0    318357
1.0     77673
Name: count, dtype: int64
```

```
data['is_defaulter'].value_counts(normalize = True)
```

```
is_defaulter
0.0    0.803871
1.0    0.196129
Name: proportion, dtype: float64
```

We can see that 80.4% are non-defaulters and 19.6% are defaulters.

Convert emp_length to numerical

```
data['emp_length'].value_counts()
```

```
emp_length
10+ years    144342
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: count, dtype: int64
```

```
data['emp_length_num'] = ((((data['emp_length'].str.replace('year',
'')).str.replace('s', '')).str.strip()).replace('10+', 10)).replace('<
1', 0)).astype('float')
```

```
data[['emp_length', 'emp_length_num']]
```

{"type":"dataframe"}

```
data['emp_length_num'].value_counts()
```

```
emp_length_num
10.0    144342
2.0      35827
0.0      31725
3.0      31665
5.0      26495
1.0      25882
4.0      23952
6.0      20841
7.0      20819
8.0      19168
9.0      15314
Name: count, dtype: int64
```

```
data = data.drop('emp_length', axis = 1)
data = data.rename(columns = {'emp_length_num' : 'emp_length'})
data.head()
```

{"type":"dataframe","variable_name":"data"}

Extract month and year from issue_d and earliest_cr_line

```
data[['issue_d_month', 'issue_d_year']] =
data['issue_d'].str.split('-', expand = True)
data[['earliest_cr_line_month', 'earliest_cr_line_year']] =
data['earliest_cr_line'].str.split('-', expand = True)
```

```
data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data['issue_d_month'].value_counts()
```

```
issue_d_month
Oct    42130
Jul    39714
Jan    34682
Nov    34068
Apr    33223
Aug    32816
Mar    31919
```

```
May    31895
Jun    30140
Dec    29082
Feb    28742
Sep    27619
Name: count, dtype: int64
```

```
data['issue_d_year'].value_counts()
```

```
issue_d_year
2014    102860
2013     97662
2015     94264
2012     41202
2016     28088
2011     17435
2010      9258
2009      3826
2008      1240
2007       195
Name: count, dtype: int64
```

```
data['earliest_cr_line_month'].value_counts()
```

```
earliest_cr_line_month
Oct    38291
Sep    37673
Aug    37349
Nov    35583
Dec    33687
Jul    31972
Mar    31617
Jan    30694
Jun    30445
May    30445
Apr    29231
Feb    29043
Name: count, dtype: int64
```

```
data['earliest_cr_line_year'].value_counts()
```

```
earliest_cr_line_year
2000    29366
2001    29083
1999    26491
2002    25901
2003    23657
        ...
1951        3
1950        3
1953        2
```

```
1944        1
1948        1
Name: count, Length: 65, dtype: int64
```

1. We can see that we have max loan applications from month of Oct and the year 2014.
2. First Credit line is mostly from year 2000.

```
data_dict = {'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6,
'Jul':7, 'Aug':8, 'Sep':9, 'Oct':10, 'Nov':11, 'Dec':12}

data['issue_d_month_no'] = data['issue_d_month'].map(data_dict)
data['earliest_cr_line_month_no'] =
data['earliest_cr_line_month'].map(data_dict)

data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data['issue_d_month_no'].value_counts()
```

```
issue_d_month_no
10    42130
7     39714
1     34682
11    34068
4     33223
8     32816
3     31919
5     31895
6     30140
12    29082
2     28742
9     27619
Name: count, dtype: int64
```

```
data['earliest_cr_line_month_no'].value_counts()
```

```
earliest_cr_line_month_no
10    38291
9     37673
8     37349
11    35583
12    33687
7     31972
3     31617
1     30694
6     30445
5     30445
4     29231
2     29043
Name: count, dtype: int64
```

Extract state code and zip code from address

```
data[['address']]
```

{"type":"dataframe"}

```
data['address'].apply(lambda x: x[-8:-6:].strip())
```

```
0         OK
1         SD
2         WV
3         MA
4         VA
          ..
396025    DC
396026    LA
396027    NY
396028    FL
396029    AR
Name: address, Length: 396030, dtype: object
```

```
data['state'] = data['address'].apply(lambda x: x[-8:-6:].strip())
```

```
data['address'].apply(lambda x: x[-5::].strip())
```

```
0         22690
1         05113
2         05113
3         00813
4         11650
          ...
396025    30723
396026    05113
396027    70466
396028    29597
396029    48052
Name: address, Length: 396030, dtype: object
```

```
data['zip_code'] = data['address'].apply(lambda x: x[-5::].strip())
```

```
data.head()
```

{"type":"dataframe","variable_name":"data"}

Distribution of loan status among state code and zip code

```
data['state'].value_counts()
```

```
state
AP    14308
AE    14157
AA    13919
```

NJ	7091
WI	7081
LA	7068
NV	7038
AK	7034
MA	7022
VA	7022
VT	7005
NY	7004
MS	7003
TX	7000
SC	6973
ME	6972
AR	6969
OH	6969
GA	6967
ID	6958
IN	6958
KS	6945
WV	6944
RI	6940
MO	6939
IL	6934
WY	6933
NE	6927
HI	6927
IA	6926
FL	6921
AZ	6918
CO	6914
OK	6911
CT	6904
MN	6904
NC	6901
OR	6898
CA	6898
AL	6898
MD	6896
WA	6895
UT	6887
SD	6887
MT	6883
DE	6874
TN	6869
ND	6858
MI	6854
DC	6842
NM	6842
PA	6825

```
NH      6818
KY      6800
Name: count, dtype: int64
```

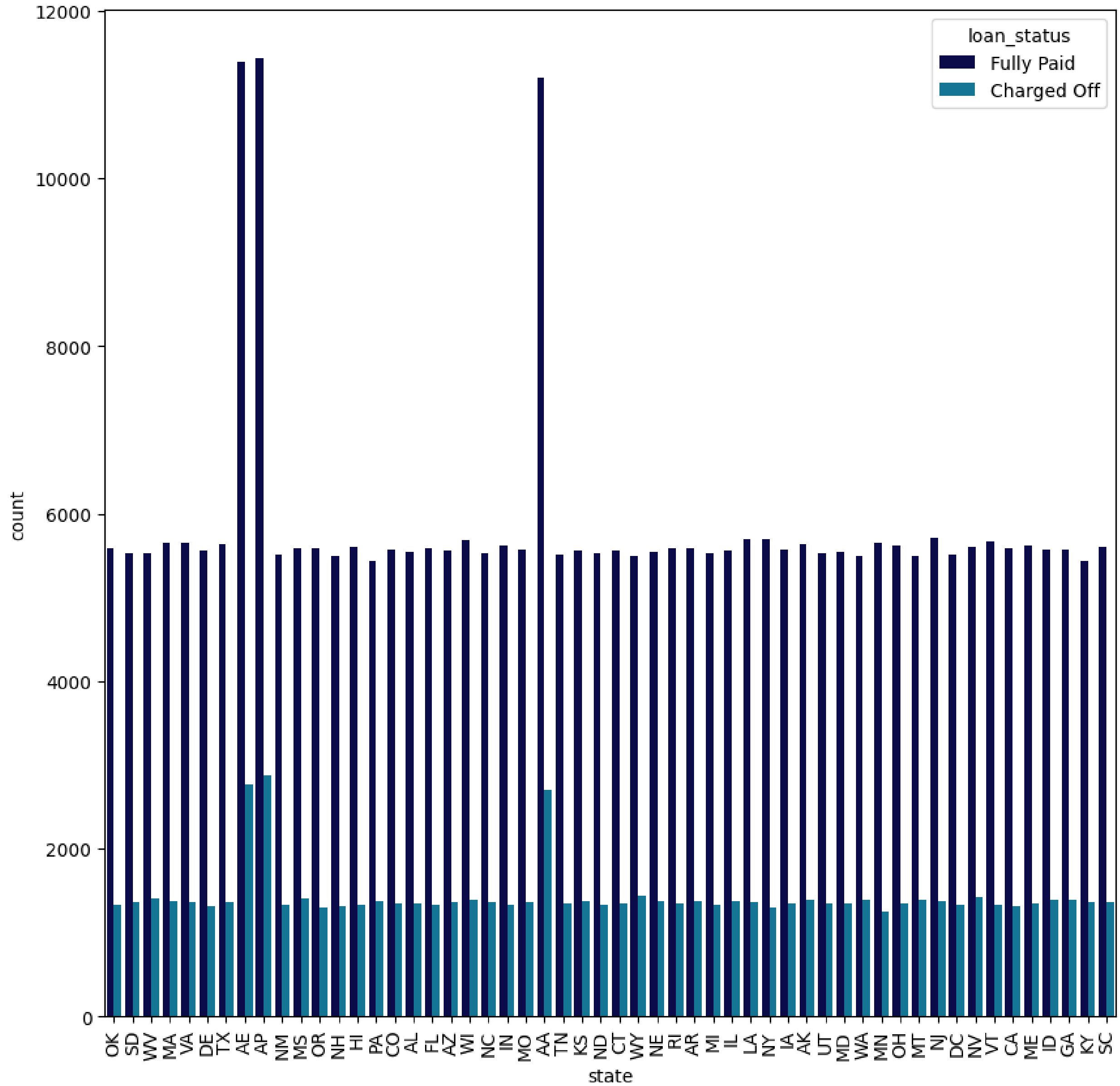
```
data['zip_code'].value_counts()
```

```
zip_code
70466    56985
30723    56546
22690    56527
48052    55917
00813    45824
29597    45471
05113    45402
11650    11226
93700    11151
86630    10981
Name: count, dtype: int64
```

```
data['zip_code'].nunique()
```

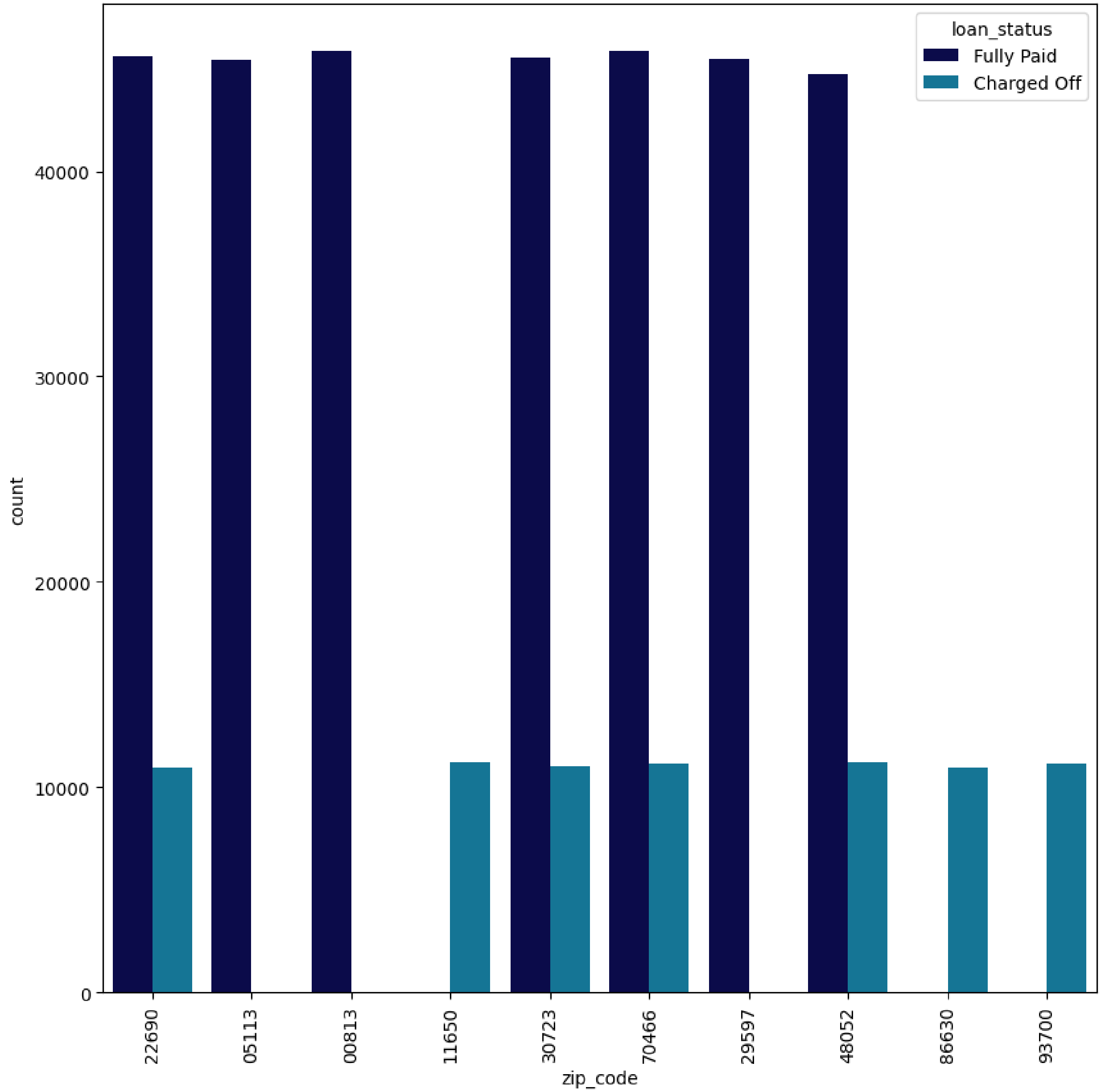
```
10
```

```
fig, ax = plt.subplots(figsize=(10,10))
sns.countplot(data = data, x = 'state', hue = 'loan_status', palette =
'ocean', ax = ax)
plt.xticks(rotation = 90)
plt.show()
```



1. State codes 'AP', 'AE' and 'AA' are the top 3 states from which loan applications have been received (in same order).
2. For all other states, it's almost similar.
3. Thus, distribution is different across states.

```
fig, ax = plt.subplots(figsize=(10,10))
sns.countplot(data = data, x = 'zip_code', hue = 'loan_status',
palette = 'ocean', ax = ax)
plt.xticks(rotation = 90)
plt.show()
```

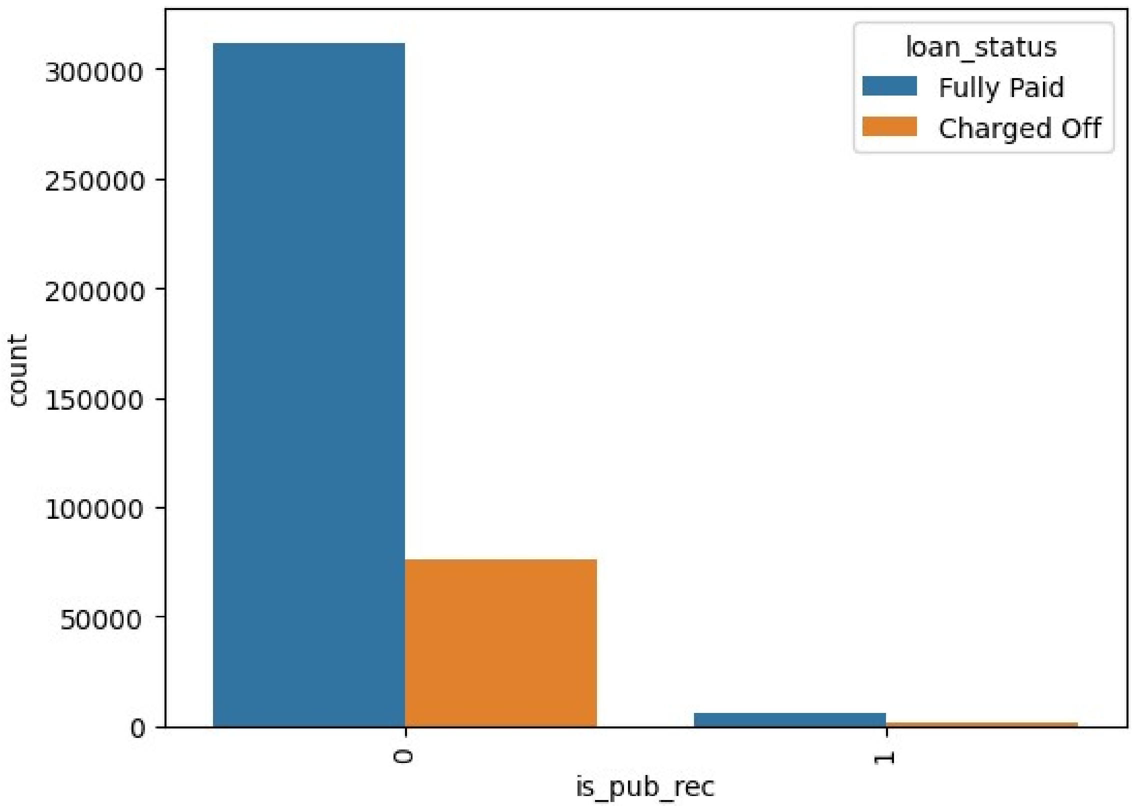


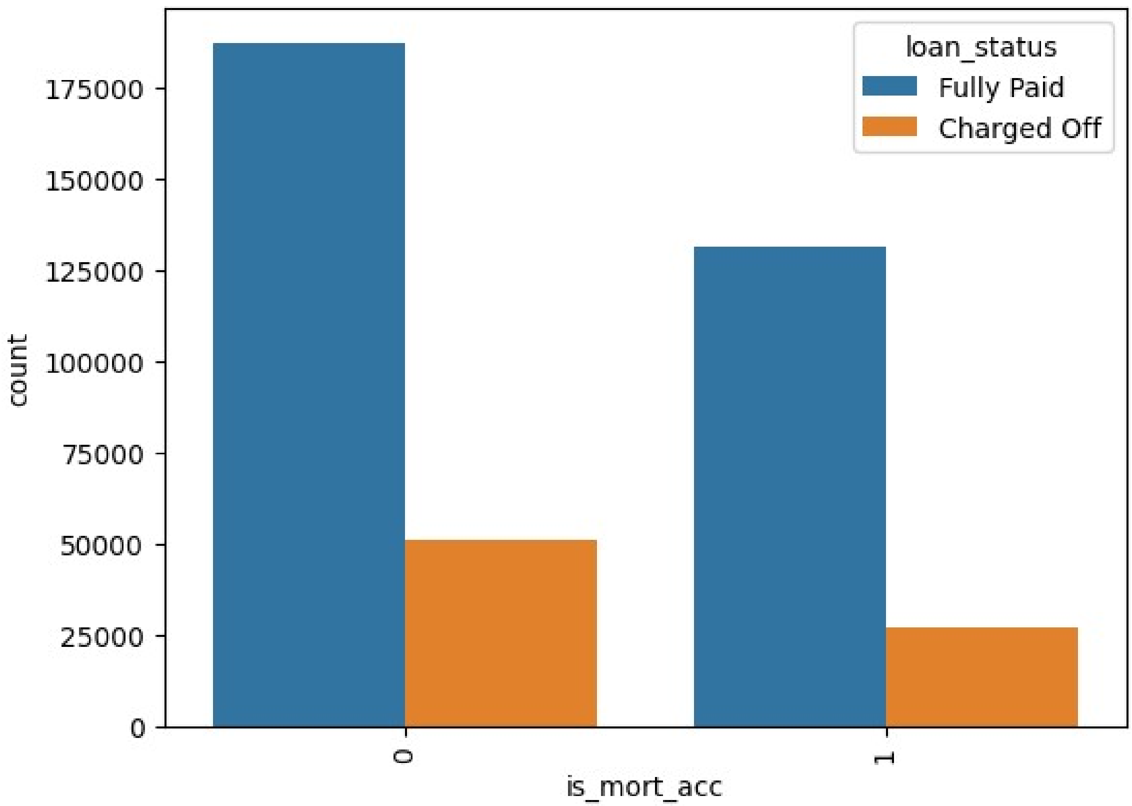
1. We can see that the distribution of borrowers w.r.t. their loan status is significantly different as per the zip codes.
2. Zip codes: 05113, 00813, 29597 are having only non-defaulters where as the zip codes: 11650, 86630, 93700 are having only defaulters.

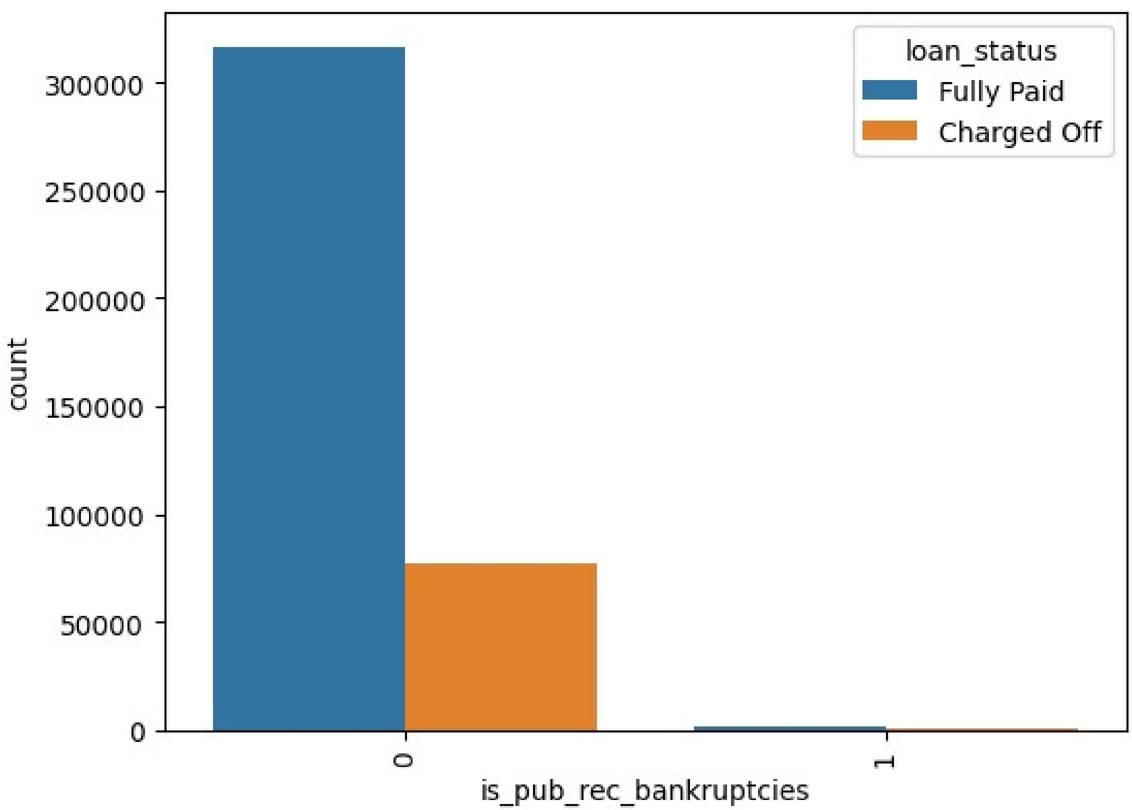
Creation of flags for Pub_rec, Mort_acc and Pub_rec_bankruptcies

```
data['is_pub_rec'] = np.where(data['pub_rec'] > 1.0, 1, 0)

data['is_mort_acc'] = np.where(data['mort_acc'] > 1.0, 1, 0)

data['is_pub_rec_bankruptcies'] =
np.where(data['pub_rec_bankruptcies'] > 1.0, 1, 0)
```

```
data.head()

{"type":"dataframe","variable_name":"data"}

cols = ['is_pub_rec', 'is_mort_acc', 'is_pub_rec_bankruptcies']

for col in cols:
  sns.countplot(data = data, x = col, fill = 'True', hue =
'loan_status')
  plt.xticks(rotation = 90)
  plt.show()
```




loan_status
Fully Paid
Charged Off
count
175000
150000
125000
100000
75000
50000
25000
0
0
1
is_mort_acc


loan_status
Fully Paid
Charged Off
count
300000
250000
200000
150000
100000
50000
0
0
1
is_pub_rec_bankruptcies

We can see that when either negative records on borrower's public credit profile are present or bankruptcy records are available for borrower, then there are more chances of defaulting.

Data preparation for modeling

```
data.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'grade', 'sub_grade',
'home_ownership',
       'annual_inc', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'revol_bal', 'total_acc', 'initial_list_status',
'application_type',
       'address', 'emp_title', 'title', 'mort_acc', 'revol_util',
       'pub_rec_bankruptcies', 'is_defaulter', 'emp_length',
'issue_d_month',
       'issue_d_year', 'earliest_cr_line_month',
'earliest_cr_line_year',
       'issue_d_month_no', 'earliest_cr_line_month_no', 'state',
'zip_code',
       'is_pub_rec', 'is_mort_acc', 'is_pub_rec_bankruptcies'],
      dtype='object')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 38 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   loan_amnt                  396030 non-null  float64
 1   term                       396030 non-null  object
 2   int_rate                   396030 non-null  float64
 3   grade                      396030 non-null  object
 4   sub_grade                  396030 non-null  object
 5   home_ownership             396030 non-null  object
 6   annual_inc                 396030 non-null  float64
 7   verification_status        396030 non-null  object
 8   issue_d                    396030 non-null  object
 9   loan_status                396030 non-null  object
 10  purpose                    396030 non-null  object
 11  dti                        396030 non-null  float64
 12  earliest_cr_line           396030 non-null  object
 13  open_acc                   396030 non-null  float64
 14  pub_rec                    396030 non-null  float64
 15  revol_bal                  396030 non-null  float64
 16  total_acc                  396030 non-null  float64
 17  initial_list_status        396030 non-null  object
 18  application_type           396030 non-null  object
 19  address                    396030 non-null  object
```

```
 20  emp_title                    396030 non-null  object
 21  title                        396030 non-null  object
 22  mort_acc                     396030 non-null  float64
 23  revol_util                   396030 non-null  float64
 24  pub_rec_bankruptcies         396030 non-null  float64
 25  is_defaulter                 396030 non-null  float64
 26  emp_length                   396030 non-null  float64
 27  issue_d_month                396030 non-null  object
 28  issue_d_year                 396030 non-null  object
 29  earliest_cr_line_month       396030 non-null  object
 30  earliest_cr_line_year        396030 non-null  object
 31  issue_d_month_no             396030 non-null  int64
 32  earliest_cr_line_month_no    396030 non-null  int64
 33  state                        396030 non-null  object
 34  zip_code                     396030 non-null  object
 35  is_pub_rec                   396030 non-null  int64
 36  is_mort_acc                  396030 non-null  int64
 37  is_pub_rec_bankruptcies      396030 non-null  int64
dtypes: float64(13), int64(5), object(20)
memory usage: 114.8+ MB
```

Dropping extra unnecessary columns

```
data.drop(columns = ['pub_rec', 'mort_acc', 'pub_rec_bankruptcies',
'issue_d_month', 'earliest_cr_line_month', 'issue_d', 'loan_status',
'earliest_cr_line', 'address' ], axis = 1, inplace = True)

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 29 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   loan_amnt                    396030 non-null  float64
 1   term                         396030 non-null  object
 2   int_rate                     396030 non-null  float64
 3   grade                        396030 non-null  object
 4   sub_grade                    396030 non-null  object
 5   home_ownership               396030 non-null  object
 6   annual_inc                   396030 non-null  float64
 7   verification_status          396030 non-null  object
 8   purpose                      396030 non-null  object
 9   dti                          396030 non-null  float64
 10  open_acc                     396030 non-null  float64
 11  revol_bal                    396030 non-null  float64
 12  total_acc                    396030 non-null  float64
 13  initial_list_status          396030 non-null  object
 14  application_type             396030 non-null  object
```

```
 15  emp_title                   396030 non-null  object
 16  title                       396030 non-null  object
 17  revol_util                  396030 non-null  float64
 18  is_defaulter                396030 non-null  float64
 19  emp_length                  396030 non-null  float64
 20  issue_d_year                396030 non-null  object
 21  earliest_cr_line_year       396030 non-null  object
 22  issue_d_month_no            396030 non-null  int64
 23  earliest_cr_line_month_no   396030 non-null  int64
 24  state                       396030 non-null  object
 25  zip_code                    396030 non-null  object
 26  is_pub_rec                  396030 non-null  int64
 27  is_mort_acc                 396030 non-null  int64
 28  is_pub_rec_bankruptcies     396030 non-null  int64
dtypes: float64(10), int64(5), object(14)
memory usage: 87.6+ MB
```

```
data = data.rename(columns = {'is_pub_rec' : 'pub_rec',
'is_mort_acc' : 'mort_acc', 'is_pub_rec_bankruptcies' :
'pub_rec_bankruptcies'})
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 29 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   loan_amnt                   396030 non-null  float64
 1   term                        396030 non-null  object
 2   int_rate                    396030 non-null  float64
 3   grade                       396030 non-null  object
 4   sub_grade                   396030 non-null  object
 5   home_ownership              396030 non-null  object
 6   annual_inc                  396030 non-null  float64
 7   verification_status         396030 non-null  object
 8   purpose                     396030 non-null  object
 9   dti                         396030 non-null  float64
 10  open_acc                    396030 non-null  float64
 11  revol_bal                   396030 non-null  float64
 12  total_acc                   396030 non-null  float64
 13  initial_list_status         396030 non-null  object
 14  application_type            396030 non-null  object
 15  emp_title                   396030 non-null  object
 16  title                       396030 non-null  object
 17  revol_util                  396030 non-null  float64
 18  is_defaulter                396030 non-null  float64
 19  emp_length                  396030 non-null  float64
 20  issue_d_year                396030 non-null  object
 21  earliest_cr_line_year       396030 non-null  object
```

```
 22   issue_d_month_no            396030 non-null  int64
 23   earliest_cr_line_month_no   396030 non-null  int64
 24   state                       396030 non-null  object
 25   zip_code                    396030 non-null  object
 26   pub_rec                     396030 non-null  int64
 27   mort_acc                    396030 non-null  int64
 28   pub_rec_bankruptcies        396030 non-null  int64
dtypes: float64(10), int64(5), object(14)
memory usage: 87.6+ MB
```

Converting necessary columns to category.

```
for col in data.columns:
  if data[col].dtype == 'object':
    data[col] = data[col].astype('category')

for col in data.columns:
  if data[col].dtype == 'category':
    print(col)
```

```
term
grade
sub_grade
home_ownership
verification_status
purpose
initial_list_status
application_type
emp_title
title
issue_d_year
earliest_cr_line_year
state
zip_code
```

```
for col in data.columns:
  if data[col].dtype != 'category':
    print(col)
```

```
loan_amnt
int_rate
annual_inc
dti
open_acc
revol_bal
total_acc
revol_util
is_defaulter
emp_length
issue_d_month_no
```

```
earliest_cr_line_month_no
pub_rec
mort_acc
pub_rec_bankruptcies
```

```python
columns_to_cat =
['emp_length','issue_d_month_no','earliest_cr_line_month_no','pub_rec'
,'mort_acc','pub_rec_bankruptcies']

for col in columns_to_cat:
  data[col] = data[col].astype('category')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 29 columns):
 #   Column                     Non-Null Count   Dtype   
---  ------                     --------------   -----   
 0   loan_amnt                  396030 non-null  float64 
 1   term                       396030 non-null  category
 2   int_rate                   396030 non-null  float64 
 3   grade                      396030 non-null  category
 4   sub_grade                  396030 non-null  category
 5   home_ownership             396030 non-null  category
 6   annual_inc                 396030 non-null  float64 
 7   verification_status        396030 non-null  category
 8   purpose                    396030 non-null  category
 9   dti                        396030 non-null  float64 
 10  open_acc                   396030 non-null  float64 
 11  revol_bal                  396030 non-null  float64 
 12  total_acc                  396030 non-null  float64 
 13  initial_list_status        396030 non-null  category
 14  application_type           396030 non-null  category
 15  emp_title                  396030 non-null  category
 16  title                      396030 non-null  category
 17  revol_util                 396030 non-null  float64 
 18  is_defaulter               396030 non-null  float64 
 19  emp_length                 396030 non-null  category
 20  issue_d_year               396030 non-null  category
 21  earliest_cr_line_year      396030 non-null  category
 22  issue_d_month_no           396030 non-null  category
 23  earliest_cr_line_month_no  396030 non-null  category
 24  state                      396030 non-null  category
 25  zip_code                   396030 non-null  category
 26  pub_rec                    396030 non-null  category
 27  mort_acc                   396030 non-null  category
 28  pub_rec_bankruptcies       396030 non-null  category
dtypes: category(20), float64(9)
memory usage: 43.7 MB
```

```
target_encoding_cols = ['term', 'grade', 'home_ownership',
'verification_status', 'initial_list_status', 'application_type',
'sub_grade', 'purpose', 'emp_title', 'title', 'state',
'issue_d_month_no', 'earliest_cr_line_month_no', 'issue_d_year',
'earliest_cr_line_year', 'zip_code',
'pub_rec','mort_acc','pub_rec_bankruptcies', 'emp_length']

for col in data.columns:
  if data[col].dtype == 'category' and col not in
target_encoding_cols:
    print(col)

for col in data.columns:
  if data[col].dtype != 'category' and col in target_encoding_cols:
    print(col)
```

Target Encoding

We will apply target encoding for all the category columns as one hot encoding for columns with 2 categories might add multi-collinearity. So, it will replace category with mean value of target column i.e. is_defaulter with that category value.

```
target_encoder = ce.TargetEncoder()

data[['term', 'grade', 'home_ownership', 'verification_status',
'initial_list_status', 'application_type', 'sub_grade', 'purpose',
'emp_title', 'title', 'state', 'issue_d_month_no',
'earliest_cr_line_month_no', 'issue_d_year', 'earliest_cr_line_year',
'zip_code', 'pub_rec','mort_acc','pub_rec_bankruptcies',
'emp_length']] = target_encoder.fit_transform(data[['term', 'grade',
'home_ownership', 'verification_status', 'initial_list_status',
'application_type', 'sub_grade', 'purpose', 'emp_title', 'title',
'state', 'issue_d_month_no', 'earliest_cr_line_month_no',
'issue_d_year', 'earliest_cr_line_year', 'zip_code',
'pub_rec','mort_acc','pub_rec_bankruptcies', 'emp_length']],
data['is_defaulter'])

data
```

{"type":"dataframe","variable_name":"data"}

#Model Building

Split training and testing data

```
data['is_defaulter']
```

```
0          0.0
1          0.0
2          0.0
```

```
3         0.0
4         1.0
         ...
396025    0.0
396026    0.0
396027    0.0
396028    0.0
396029    0.0
Name: is_defaulter, Length: 396030, dtype: float64
```

```
X = data.drop('is_defaulter', axis = 1)
y = data['is_defaulter']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3, random_state = 42)
```

```
X_train.shape
```

```
(277221, 28)
```

```
X_test.shape
```

```
(118809, 28)
```

```
y_train.shape
```

```
(277221,)
```

```
y_test.shape
```

```
(118809,)
```

```
X_train.head()
```

{"type":"dataframe","variable_name":"X_train"}

```
y_train.head()
```

```
3412      0.0
134032    0.0
19526     0.0
61015     0.0
2896      0.0
Name: is_defaulter, dtype: float64
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 277221 entries, 3412 to 121958
Data columns (total 28 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   loan_amnt                  277221 non-null  float64
```

```
 1   term                        277221 non-null  float64
 2   int_rate                    277221 non-null  float64
 3   grade                       277221 non-null  float64
 4   sub_grade                   277221 non-null  float64
 5   home_ownership              277221 non-null  float64
 6   annual_inc                  277221 non-null  float64
 7   verification_status         277221 non-null  float64
 8   purpose                     277221 non-null  float64
 9   dti                         277221 non-null  float64
 10  open_acc                    277221 non-null  float64
 11  revol_bal                   277221 non-null  float64
 12  total_acc                   277221 non-null  float64
 13  initial_list_status         277221 non-null  float64
 14  application_type            277221 non-null  float64
 15  emp_title                   277221 non-null  float64
 16  title                       277221 non-null  float64
 17  revol_util                  277221 non-null  float64
 18  emp_length                  277221 non-null  float64
 19  issue_d_year                277221 non-null  float64
 20  earliest_cr_line_year       277221 non-null  float64
 21  issue_d_month_no            277221 non-null  float64
 22  earliest_cr_line_month_no   277221 non-null  float64
 23  state                       277221 non-null  float64
 24  zip_code                    277221 non-null  float64
 25  pub_rec                     277221 non-null  float64
 26  mort_acc                    277221 non-null  float64
 27  pub_rec_bankruptcies        277221 non-null  float64
dtypes: float64(28)
memory usage: 61.3 MB
```

```
y_train.info()
```

```
<class 'pandas.core.series.Series'>
Index: 277221 entries, 3412 to 121958
Series name: is_defaulter
Non-Null Count   Dtype
--------------   -----
277221 non-null  float64
dtypes: float64(1)
memory usage: 4.2 MB
```

```
y_train
```

```
3412      0.0
134032    0.0
19526     0.0
61015     0.0
2896      0.0
         ...
259178    0.0
```

```
365838    1.0
131932    0.0
146867    0.0
121958    1.0
Name: is_defaulter, Length: 277221, dtype: float64
```

Feature Scaling

Using Standard Scaler as scaling method and note that fit_transform should be done only on training data.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression using sklearn

```
model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

LogisticRegression(max_iter=1000)

pred_y_train = model.predict(X_train)
pred_y_test = model.predict(X_test)

print('Accuracy of Logistic Regression Classifier on test set:
{:.3f}'.format(model.score(X_test, y_test)))

Accuracy of Logistic Regression Classifier on test set: 0.920
```

Model coef and intercept

```
coefs = model.coef_.reshape(-1,1)
lst = list(X.columns)

#w1, w2...w27
coefs

array([[ 1.38398980e-01],
       [ 2.29202670e-01],
       [ 1.77839868e-01],
       [-2.33215218e-02],
       [ 2.11443813e-01],
       [ 1.46470923e-01],
       [-6.77768341e-03],
       [ 3.35492924e-02],
       [-1.97819527e-01],
       [ 1.47518410e-01],
       [ 1.73290930e-01],
```

```
       [-7.15317429e-02],
       [-7.00802116e-02],
       [-1.17981489e-01],
       [ 4.84440015e-03],
       [ 1.31665904e+00],
       [ 4.92844290e-01],
       [ 2.04307342e-01],
       [ 2.57313199e-02],
       [ 5.44637475e-02],
       [-4.45658436e-03],
       [ 2.77710460e-02],
       [-4.90437871e-03],
       [ 3.15780681e-02],
       [ 9.28179390e+00],
       [-1.16193516e-02],
       [ 8.29470126e-02],
       [-4.73083570e-03]])
```

```
for i in range(len(lst)):
  print('Column: ', lst[i], ' Coef: ', coefs[i])
```

```
Column:  loan_amnt  Coef:  [0.13839898]
Column:  term  Coef:  [0.22920267]
Column:  int_rate  Coef:  [0.17783987]
Column:  grade  Coef:  [-0.02332152]
Column:  sub_grade  Coef:  [0.21144381]
Column:  home_ownership  Coef:  [0.14647092]
Column:  annual_inc  Coef:  [-0.00677768]
Column:  verification_status  Coef:  [0.03354929]
Column:  purpose  Coef:  [-0.19781953]
Column:  dti  Coef:  [0.14751841]
Column:  open_acc  Coef:  [0.17329093]
Column:  revol_bal  Coef:  [-0.07153174]
Column:  total_acc  Coef:  [-0.07008021]
Column:  initial_list_status  Coef:  [-0.11798149]
Column:  application_type  Coef:  [0.0048444]
Column:  emp_title  Coef:  [1.31665904]
Column:  title  Coef:  [0.49284429]
Column:  revol_util  Coef:  [0.20430734]
Column:  emp_length  Coef:  [0.02573132]
Column:  issue_d_year  Coef:  [0.05446375]
Column:  earliest_cr_line_year  Coef:  [-0.00445658]
Column:  issue_d_month_no  Coef:  [0.02777105]
Column:  earliest_cr_line_month_no  Coef:  [-0.00490438]
Column:  state  Coef:  [0.03157807]
Column:  zip_code  Coef:  [9.2817939]
Column:  pub_rec  Coef:  [-0.01161935]
Column:  mort_acc  Coef:  [0.08294701]
Column:  pub_rec_bankruptcies  Coef:  [-0.00473084]
```

```
coef_df = pd.DataFrame()
coef_df['column_name'] = lst
coef_df['coef_value'] = coefs
coef_df.sort_values(by = 'coef_value', ascending = False)
```

{"summary":"{\n \"name\": \"coef_df\",\n \"rows\": 28,\n \"fields\": [\n   {\n   \"column\": \"column_name\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 28,\n   \"samples\": [\n   \"home_ownership\",\n   \"revol_bal\",\n   \"dti\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"coef_value\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.754294768944616,\n   \"min\": -0.1978195274163468,\n   \"max\": 9.28179390488985,\n   \"num_unique_values\": 28,\n   \"samples\": [\n   0.14647092257489297,\n   -0.07153174292788539,\n   0.14751841013769154\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

We can see that zip_code and emp_title are having top 2 highest weights, which is quite surprising.

```
#w0
model.intercept_
```

```
array([-2.11207663])
```

```
model.score(X_test, y_test)
```

```
0.9204689880396266
```

Metric evaluation

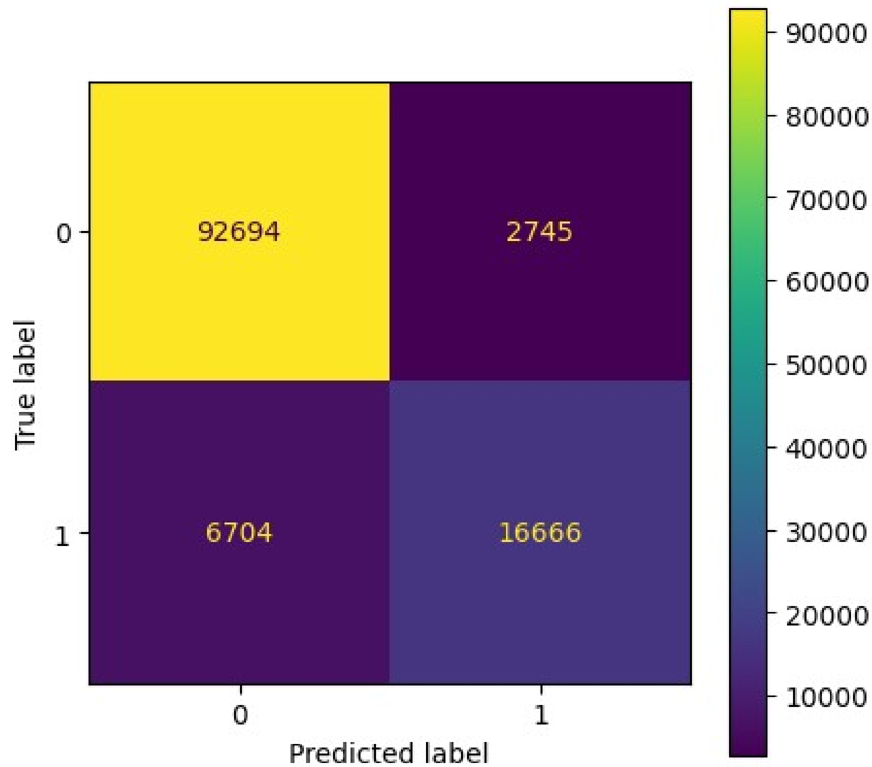
Confusion Matrix

```
conf_matrix = confusion_matrix(y_test, pred_y_test)
conf_matrix
```

```
array([[92694,  2745],
       [ 6704, 16666]])
```

```
fig, ax = plt.subplots(figsize= (5,5))
ConfusionMatrixDisplay(conf_matrix).plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7bd14fadbd30>
```



Accuracy

```
accuracy = np.diag(conf_matrix).sum() / conf_matrix.sum()
accuracy
```

```
0.9204689880396266
```

```
accuracy_score(y_test, pred_y_test)
```

```
0.9204689880396266
```

Precision

```
precision_score(y_test, pred_y_test)
```

```
0.8585853382102931
```

Recall

```
recall_score(y_test, pred_y_test)
```

```
0.7131364997860505
```

Precision is higher than recall which means that FN is higher than FP.

F1 Score

Testing F1-score

```
f1_score(y_test, pred_y_test)
```

```
0.7791309226058298
```

Training F1-score

```
f1_score(y_train, pred_y_train)
```

```
0.7793166011490779
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_y_test))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.97      0.95     95439
         1.0       0.86      0.71      0.78     23370

    accuracy                           0.92    118809
   macro avg       0.90      0.84      0.87    118809
weighted avg       0.92      0.92      0.92    118809
```

AUC-ROC Curve

```
probability = model.predict_proba(X_test)
```

```
probability
```

```
array([[3.22092916e-01, 6.77907084e-01],
       [9.99385798e-01, 6.14202397e-04],
       [9.32259580e-01, 6.77404198e-02],
       ...,
       [9.63454985e-01, 3.65450145e-02],
       [9.99344302e-01, 6.55697673e-04],
       [9.99937408e-01, 6.25923859e-05]])
```

```
probabilities = probability[:, 1]
probabilities
```

```
array([6.77907084e-01, 6.14202397e-04, 6.77404198e-02, ...,
       3.65450145e-02, 6.55697673e-04, 6.25923859e-05])
```

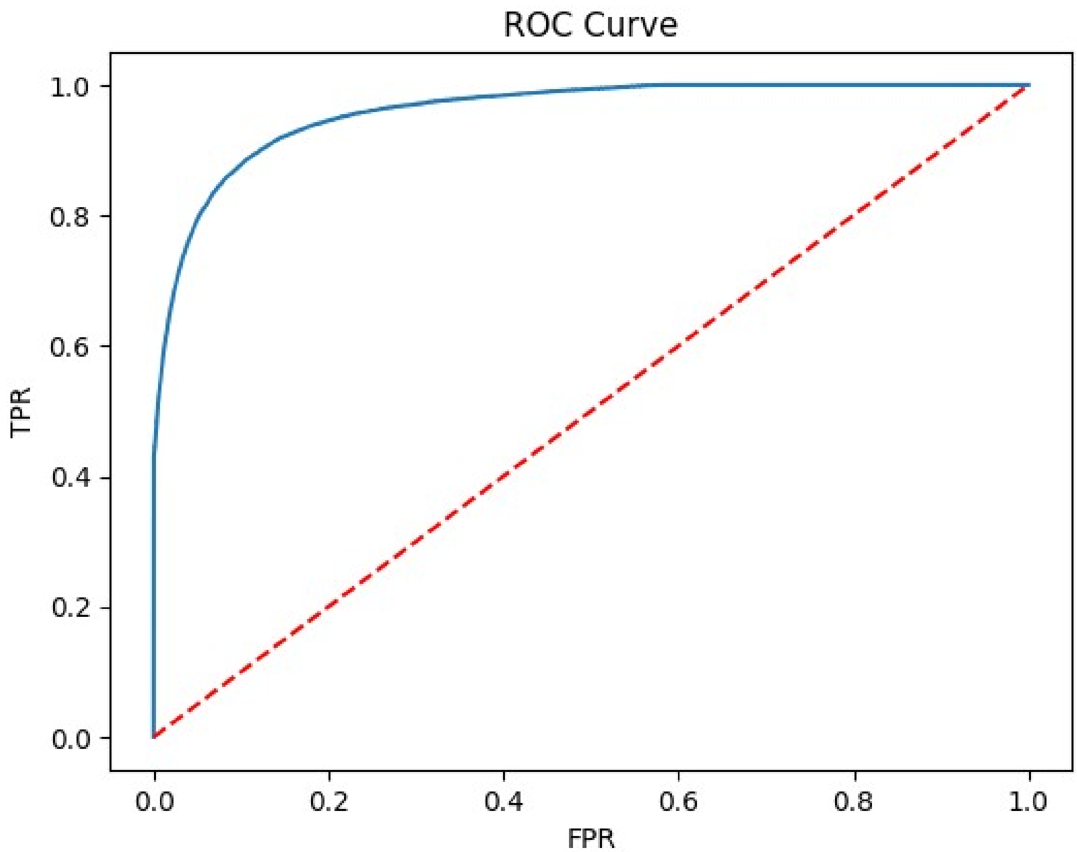
```
fpr, tpr, thr = roc_curve(y_test, probabilities)
```

```
plt.plot(fpr, tpr)
```

```
#random model
```

```
plt.plot(fpr, fpr, '--', color = 'red')
```

```
plt.title('ROC Curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

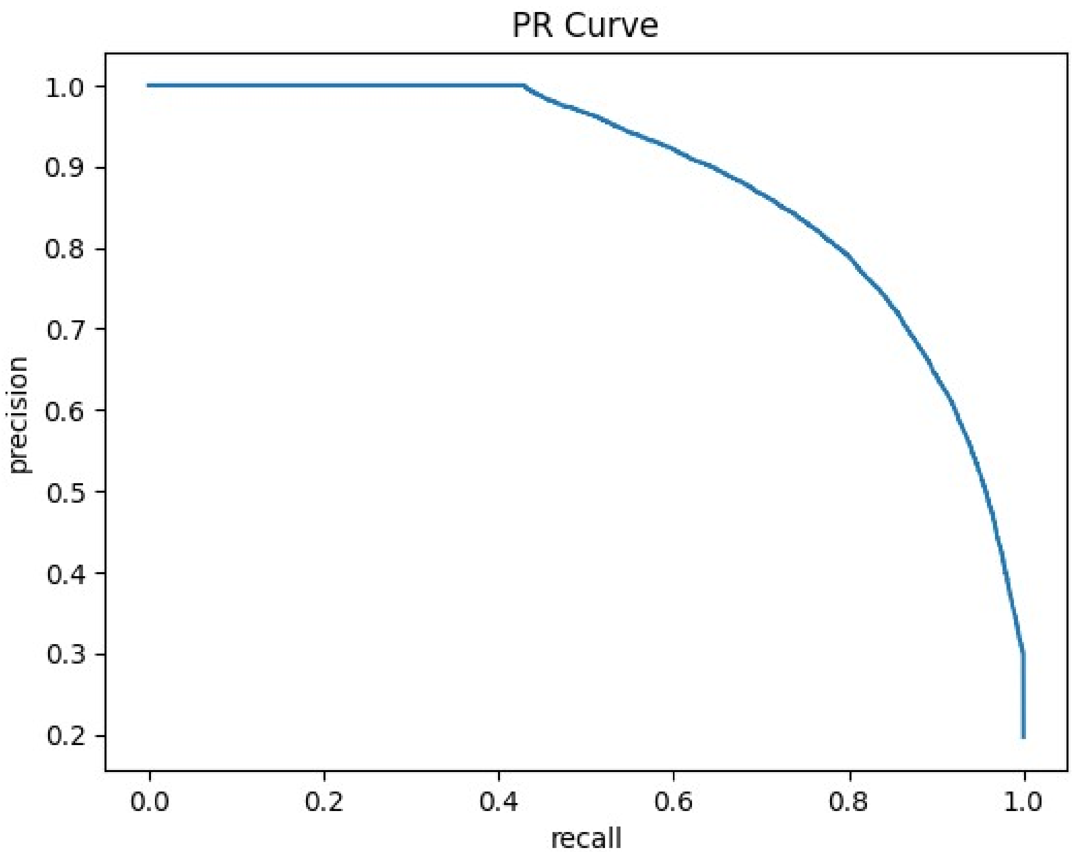



```
roc_auc_score(y_test, probabilities)
```

```
0.9597543106693195
```

PR Curve

```
precision, recall, thr = precision_recall_curve(y_test, probabilities)

plt.plot(recall, precision)

plt.xlabel('recall')
plt.ylabel('precision')
plt.title('PR Curve')
plt.show()
```

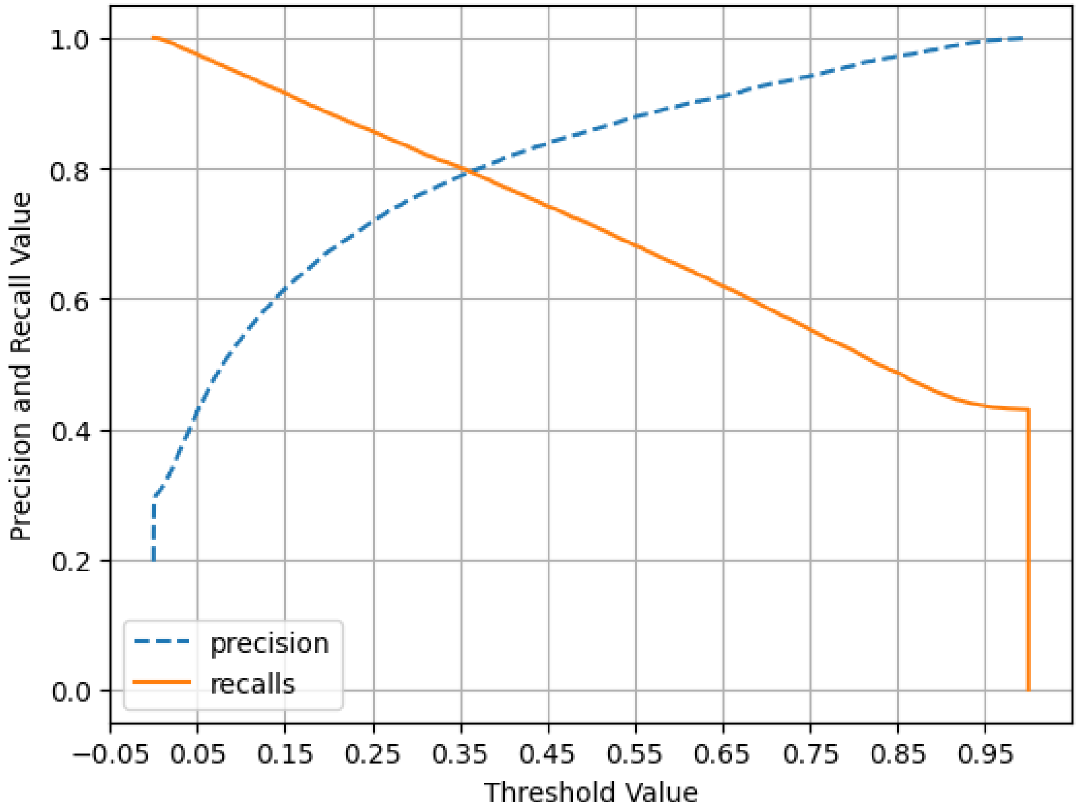



```
auc(recall, precision)
```

```
0.8881669306885851
```

```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test,
pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary],
linestyle='--', label='precision')
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary],
label='recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall
Value')
    plt.legend(); plt.grid()
    plt.show()
```

```
precision_recall_curve_plot(y_test, model.predict_proba(X_test)[:,1])
```



Threshold value here comes out to be almost 0.35

Multicollinearity check using Variance Inflation Factor (VIF) -

Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. Multicollinearity can be a problem in a regression model because we would not be able to distinguish between the individual effects of the independent variables on the dependent variable.

Multicollinearity can be detected via various methods. One such method is Variance Inflation Factor aka VIF. In VIF method, we pick each independent feature and regress it against all of the other independent features. VIF score of an independent variable represents how well the variable is explained by other independent variables.

VIF = 1/1-R2

```
def calc_vif(X):
    # Calculating the VIF
    vif = pd.DataFrame()
```

```
    vif['Feature'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by='VIF', ascending = False)
    return vif

calc_vif(X)
```


```
{"summary":"{\n  \"name\": \"calc_vif(X)\",\n  \"rows\": 28,\n
\"fields\": [\n    {\n      \"column\": \"Feature\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 28,\n        \"samples\": [\n
\"sub_grade\",\n          \"loan_amnt\",\n          \"int_rate\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"VIF\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 1445.469758322807,\n
\"min\": 1.81,\n        \"max\": 5966.65,\n
\"num_unique_values\": 28,\n        \"samples\": [\n          187.51,\
n          7.17,\n          212.17\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
X.drop(columns=['pub_rec_bankruptcies'], axis=1, inplace=True)
calc_vif(X)[:5]
```


```
{"summary":"{\n  \"name\": \"calc_vif(X)[:5]\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Feature\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"initial_list_status\",\n          \"application_type\",\n
\"emp_length\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"VIF\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 879.0135631604327,\n        \"min\": 1475.18,\n
\"max\": 3703.96,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          2533.0,\n          1475.18,\n
2006.58\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
X.drop(columns=['earliest_cr_line_month_no'], axis=1, inplace=True)
calc_vif(X)[:5]
```


```
{"summary":"{\n  \"name\": \"calc_vif(X)[:5]\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Feature\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"emp_length\",\n          \"pub_rec\",\n          \"state\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"VIF\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
```

\"std\": 400.4819925165176,\n   \"min\": 1250.02,\n
\"max\": 2267.53,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   1849.24,\n   1250.02,\n
1636.76\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['initial_list_status'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"state\",\n   \"issue_d_month_no\",\n
\"application_type\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 490.04208693743846,\n   \"min\": 409.24,\n
\"max\": 1674.54,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   1513.27,\n   409.24,\n
1313.33\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['emp_length'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"application_type\",\n   \"int_rate\",\n   \"pub_rec\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 499.38102283727204,\n   \"min\": 210.15,\n
\"max\": 1319.1,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   1187.17,\n   210.15,\n
1082.78\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['state'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"pub_rec\",\n   \"sub_grade\",\n
\"issue_d_month_no\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 377.087264210819,\n   \"min\": 185.42,\n   \"max\":
961.89,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
904.44,\n   185.42,\n   387.45\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['application_type'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"issue_d_month_no\",\n   \"mort_acc\",\n
\"int_rate\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 167.35722467225608,\n   \"min\": 123.39,\n
\"max\": 539.96,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   352.78,\n   123.39,\n
206.63\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['pub_rec'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"int_rate\",\n   \"grade\",\n   \"sub_grade\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 50.7330841759103,\n   \"min\": 105.41,\n   \"max\":
221.72,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
196.92,\n   105.41,\n   178.17\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['issue_d_month_no'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"sub_grade\",\n   \"earliest_cr_line_year\",\n
\"mort_acc\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 42.96233664501967,\n   \"min\": 87.23,\n   \"max\":
182.73,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
169.56,\n   87.23,\n   105.46\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['int_rate'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"mort_acc\",\n   \"purpose\",\n   \"sub_grade\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"VIF\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 12.236969396055542,\
n   \"min\": 80.92,\n   \"max\": 105.26,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   104.74,\n
80.92,\n   104.27\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

```
X.drop(columns=['grade'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"earliest_cr_line_year\",\n   \"title\",\n
\"purpose\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 21.565842900290267,\n   \"min\": 46.07,\n
\"max\": 104.73,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   84.09,\n   46.07,\n   80.91\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['mort_acc'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"earliest_cr_line_year\",\n   \"verification_status\",\n
\"home_ownership\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 17.590588392660436,\n   \"min\": 36.52,\n
\"max\": 77.25,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   72.56,\n   36.52,\n   50.26\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['purpose'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"home_ownership\",\n   \"issue_d_year\",\n
\"verification_status\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"VIF\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 16.727715624077305,\n   \"min\":
23.3,\n   \"max\": 67.31,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   47.5,\n   23.3,\n   35.6\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['earliest_cr_line_year'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"verification_status\",\n   \"emp_title\",\n
\"title\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 7.828088527859148,\n   \"min\": 16.75,\n   \"max\":
33.7,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
33.68,\n   16.75,\n   33.05\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['home_ownership'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"verification_status\",\n   \"open_acc\",\n
\"issue_d_year\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 7.455894983166005,\n   \"min\": 13.83,\n   \"max\":
30.42,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
29.09,\n   13.83,\n   22.16\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['title'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"issue_d_year\",\n   \"term\",\n   \"emp_title\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"VIF\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 5.671773972929458,\n
\"min\": 13.05,\n   \"max\": 26.91,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   18.17,\n
13.05,\n   15.03\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

```
X.drop(columns=['verification_status'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"open_acc\",\n   \"total_acc\",\n   \"emp_title\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"VIF\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 1.821930295044242,\n
\"min\": 11.78,\n   \"max\": 16.58,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   13.71,\n
11.78,\n   13.37\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

```
X.drop(columns=['issue_d_year'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"term\",\n   \"annual_inc\",\n   \"total_acc\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"VIF\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 1.9949887217726319,\

n   \"min\": 8.17,\n   \"max\": 13.51,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   12.13,\n
8.17,\n   11.76\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['open_acc'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"emp_title\",\n   \"revol_util\",\n   \"annual_inc\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"VIF\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 2.129143020090478,\n
\"min\": 7.3,\n   \"max\": 12.13,\n   \"num_unique_values\":
5,\n   \"samples\": [\n   10.47,\n   7.3,\n
8.06\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
X.drop(columns=['term'], axis=1, inplace=True)
calc_vif(X)[:5]
```

{"summary":"{\n \"name\": \"calc_vif(X)[:5]\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n
\"annual_inc\",\n   \"dti\",\n   \"total_acc\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"VIF\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1.0531476629608976,\n   \"min\": 7.1,\n   \"max\":
9.66,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
7.98,\n   7.1,\n   7.32\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
X.head()
```

{"type":"dataframe","variable_name":"X"}

```
X = scaler.fit_transform(X)
```

```
kfold = KFold(n_splits=5)
accuracy = np.mean(cross_val_score(model, X, y, cv=kfold,
scoring='accuracy', n_jobs=-1))
print("Cross Validation accuracy: {:.3f}".format(accuracy))
```

```
Cross Validation accuracy: 0.919
```

SMOTE

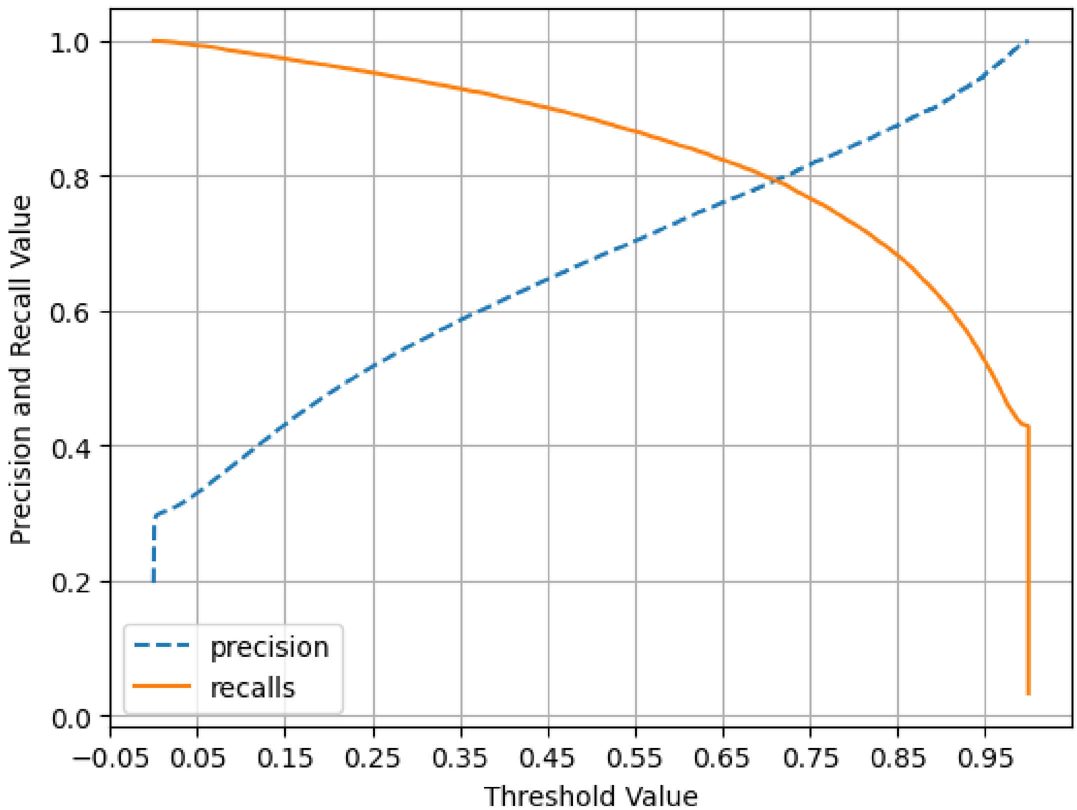
Oversampling be creating synthetic samples for minority class which is class 1 here i.e. defaulters to make the no of samples for class 1 same as class 0

```
smt = SMOTE()

print('Before SMOTE')
y_train.value_counts()

Before SMOTE

is_defaulter
0.0    222918
1.0     54303
Name: count, dtype: int64

X_sm, y_sm = smt.fit_resample(X_train, y_train)

print('After SMOTE')
y_sm.value_counts()

After SMOTE

is_defaulter
0.0    222918
1.0    222918
Name: count, dtype: int64

model_smote= LogisticRegression(max_iter=1000)

def training(model, X_train, X_test, y_train, y_test):
  model.fit(X_train, y_train)

  train_y_pred = model.predict(X_train)
  test_y_pred = model.predict(X_test)

  train_score = f1_score(y_train, train_y_pred)
  test_score = f1_score(y_test, test_y_pred)

  print(classification_report(y_test, test_y_pred))

  return train_score, test_score

f1_train, f1_test = training(model_smote, X_sm, X_test, y_sm, y_test)

print(f'Training F1 score: {f1_train}, Testing F1 score: {f1_test}')

              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370
```

```
    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Training F1 score: 0.8950805398345668, Testing F1 score:
0.7658241921138453
```

We can see that training F1-score after SMOTE has significantly inceased from 0.779 to 0.895 and the testing score has decreased from 0.779 to 0.765.

```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test,
pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    # plot precision
    plt.plot(thresholds, precisions[0:threshold_boundary],
linestyle='--', label='precision')
    # plot recall
    plt.plot(thresholds, recalls[0:threshold_boundary],
label='recalls')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))

    plt.xlabel('Threshold Value'); plt.ylabel('Precision and Recall
Value')
    plt.legend(); plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, model_smote.predict_proba(X_test)
[:,1])
```



Threshold value has increased after SMOTE from 0.35 to 0.7

Hyperparameter Tuning

L2

```
model_l2 = LogisticRegression(C = 5, penalty = 'l2', solver = 
'liblinear')

f1_train, f1_test = training(model_l2, X_sm, X_test, y_sm, y_test)

print(f'Training F1 score: {f1_train}, Testing F1 score: {f1_test}')
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Training F1 score: 0.8951170091260161, Testing F1 score: 
0.7657390595471893
```

Tuning lambda values (C) with penalty L2

```
lambda_values = [0.1, 0.01, 0.001, 0.0001, 10, 100]

for lambda_val in lambda_values:
  model_l2 = LogisticRegression(C = lambda_val, penalty = 'l2', solver
= 'liblinear')
  f1_train, f1_test = training(model_l2, X_sm, X_test, y_sm, y_test)
  print('Lambda value: ', lambda_val)
  print(f'Training F1 score: {f1_train}, Testing F1 score: {f1_test}')
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Lambda value:  0.1
Training F1 score: 0.8950678523812303, Testing F1 score:
0.766201987246033
              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Lambda value:  0.01
Training F1 score: 0.8946107407590543, Testing F1 score:
0.7670606601248885
              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.87      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Lambda value:  0.001
Training F1 score: 0.8913456563342349, Testing F1 score:
0.7658952496954933
              precision    recall  f1-score   support

         0.0       0.97      0.89      0.93     95439
         1.0       0.66      0.87      0.75     23370
```

```
    accuracy                           0.89    118809
   macro avg       0.81      0.88      0.84    118809
weighted avg       0.90      0.89      0.89    118809

Lambda value:  0.0001
Training F1 score: 0.8833002984427699, Testing F1 score:
0.7491695578356832
              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Lambda value:  10
Training F1 score: 0.895126329488848, Testing F1 score:
0.7657390595471893
              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809

Lambda value:  100
Training F1 score: 0.8951099163706618, Testing F1 score:
0.7657390595471893
```

The F1-scores for train and test are almost same for all the values though highest for 0.1

L1

```
model_l1 = LogisticRegression(C = 5, penalty = 'l1', solver =
'liblinear')

f1_train, f1_test = training(model_l1, X_sm, X_test, y_sm, y_test)

print(f'Training F1 score: {f1_train}, Testing F1 score: {f1_test}')

              precision    recall  f1-score   support

         0.0       0.97      0.90      0.93     95439
         1.0       0.68      0.88      0.77     23370

    accuracy                           0.89    118809
   macro avg       0.82      0.89      0.85    118809
weighted avg       0.91      0.89      0.90    118809
```

```
Training F1 score: 0.8951174798216542, Testing F1 score:
0.7658076588176446
```

L1 and L2 both are giving almost same F1-scores for train and test data.

Class weight

Using class weight algorithm to use the weight calculated as (no of samples in majority class / no of samples in minority class) for minority class and 1 for majority class

```
wt = y_train.value_counts()[0]/y_train.value_counts()[1]

wt

4.105077067565328

model = LogisticRegression(class_weight={0: 1, 1: wt})

model.fit(X_train, y_train)

LogisticRegression(class_weight={0: 1, 1: 4.105077067565328})

pred_y_test = model.predict(X_test)
```

Testing F1-score

```
f1_score(y_test, pred_y_test)

0.7629366641526094
```

Training F1-score

```
f1_score(y_train, pred_y_train)

0.7793166011490779
```

Both train and test F1-scores are poor as compared to other options that we tried

Thus, we can see that simple logistic regression model is giving higher F1-score of 0.779 for testing but SMOTE has significantly better taining F1-score of ~0.9 and testing F1-score of 0.766

#Actionable insights and recommendations

- Around 80% of customers have fully paid their Loan amount. The defaulters are ~20%. From Personal loan business perspective, this ratio is high. These 20% will contribute in NPAs of LoanTap. To reduce the risk of NPAs: ** LoanTap should add slightly stringent rules to bring down this ratio to 5% to 6%. ** LoanTap should provide loans at slightly higher rate than other banks. This will offset the risks of defaulters and maintain the profitability of the business.

- The loan term 60 months has negative coefficient which means more chances of unlikely to pay. Which means LoanTap should focus more on loans for shorter duration (i.e. 36 months). Their social media campaign and marketing strategy should be based on this consideration.
- Overall statistics of the model [Classification Metrics]: ** Accuracy --> 92% ** Precision --> 86% ** Recall --> 71% ** F1-score --> 78%
- Precision is higher than recall which means that false positives is lesser than false negatives.That means LoanTap will not lose the potential clients that much with this model but might struggle in identifying NPAs.
- Features which have significant impact on outcome are as follow:

1. int_rate: Interest Rate
2. sub_grade: loan subgrade
3. term : number of payments on the loan
4. application_type
5. zip code (from address)
6. emp_title: job title supplied by the Borrower

- The sub_grade and grade logic to classify person by LoanTap is well created. From the model pov, it is considered to be significant. ** For the loan ratings 'A', 'B', 'C' and 'D', there's a huge difference between the no of defaulters and non-defaulters and no of non-defaulters is high which means more likely loan will be paid but for grades 'E', 'F' ad 'G', these nos are almost same that means high risk ratings. ** Thus, we can say that grades 'A', 'B' and 'C' are low risk, grades 'D' and 'E' are moderate risk and 'F' and 'G' are high risk. ** Similar pattern observed for the sub-grades with 1 being low risk in that grade and 5 being high risk. ** So overall, A1 is lowest risk and G5 is the highest risk.

So it's recommended to avoid approving the loan for highest risk customers.

- Distribution of loan status across States: State codes 'AP', 'AE' and 'AA' are the top 3 states from which loan applications have been received (in same order). For all other states, it's almost similar. Thus, distribution is different across states. So, it's recommended that LoanTap should focus more on getting more customers from these 3 states as they are also more likely to repay the loan.

- Distribution of loan status across zip codes: ** We can see that the distribution of borrowers w.r.t. their loan status is significantly different as per the zip codes. Zip codes: 05113, 00813, 29597 are having only non-defaulters where as the zip codes: 11650, 86630, 93700 are having only defaulters. So, it's recommended that LoanTap should focus more on getting customers from zip codes 05113, 00813, 29597.

- Among the borrowers with initial list status as 'w' i.e. whole the difference between defaulters and non-defaulters is lesser as compared to 'f' which means that more chance of defaulting when entire amount is approved.

So, it's recommended to approve only fractional amount in the beginning.

- Employee tenure, income verification status, purpose of loan, home ownership status and loan application type do not make much impact on defaulting.

- Chances of defaulting is the least for home_ownership as 'Mortgage' so recommended to gather more borrowers having this criteria met

- We can see that when either negative records on borrower's public credit profile are present or bankruptcy records are available for borrower, then there are more chances of defaulting.

So, it's recommended to avoid approving the loan when negative records are found or bankruptcy is found.

```
'''
Questionnaire:

1. What percentage of customers have fully paid their Loan Amount?
Ans--> We can see that 80.4% are non-defaulters and 19.6% are
defaulters.

2. Comment about the correlation between Loan Amount and Installment
features.
Ans--> The correlation coefficient is the highest for loan amount and
installment features which is +0.97 which tells us that greater the
loan amount, greater will be the monthly installment amount.

3. The majority of people have home ownership as _______.
Ans--> 'Mortgage'

4. People with grades ‘A’ are more likely to fully pay their loan.
(T/F)
Ans--> True

5. Name the top 2 afforded job titles.
Ans--> 'Teacher' followed by 'Manager'

6. Thinking from a bank's perspective, which metric should our primary
focus be on..
ROC AUC
Precision
Recall
F1 Score
Ans--> The best metric to consider is : F1-score
As we need to give equal importance to both precision and recall.
We don't want to miss potential customers and at the same time we also
don't want to give loan to defaulters


7. How does the gap in precision and recall affect the bank?
Ans-->
Precision 86% and Recall 71%
Precision is higher than recall which means that false positives is
lesser than false negatives.
```

```
That means LoanTap will not lose the potential clients that much with
this model might struggle in identifying NPAs.
From confusion matrix, % of misclassified points in each class:
class 0: 2.88%
class 1: 28.69%
If Recall value is low (i.e. FN is high), it means Bank's NPA
(defaulters) may increase.
If Precision value is low (i.e. FP is high), t means Bank is losing in
opportunity cost.


8. Which were the features that heavily affected the outcome?
Ans--> Features with higher posititive coefficients:
zip_code
emp_title
title
term
sub_grade
revol_util
int_rate
open_acc
dti
home_ownership
loan_amnt


9. Will the results be affected by geographical location? (Yes/No)
Ans--> Yes
Distribution across States:
State codes 'AP', 'AE' and 'AA' are the top 3 states from which loan
applications have been received (in same order).
For all other states, it's almost similar. Thus, distribution is
different across states.

Distribution across zip codes:
We can see that the distribution of borrowers w.r.t. their loan status
is significantly different as per the zip codes.
Zip codes: 05113, 00813, 29597 are having only non-defaulters where as
the zip codes: 11650, 86630, 93700 are having only defaulters.

'''
```