```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

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

```
data.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	6.0
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	13.0

5 rows × 27 columns

```
# Shape of the dataset -
print("No. of rows: ", data.shape[0])
print("No. of columns: ", data.shape[1])
```

```
No. of rows:  12409
No. of columns:  27
```

```
# Checking the distribution of outcome labels -
data.loan_status.value_counts(normalize=True)*100
```

```
Fully Paid     80.820372
Charged Off    19.179628
Name: loan_status, dtype: float64
```

```
# Statistical summary of the dataset -
data.describe(include='all')
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...
count	12409.000000	12409	12409.000000	12409.000000	12409	12409	11705	11843	12409	1.240900e+04	...
unique	NaN	2	NaN	NaN	7	35	8411	11	5	NaN	...
top	NaN	36 months	NaN	NaN	B	B3	Teacher	10+ years	MORTGAGE	NaN	...
freq	NaN	9477	NaN	NaN	3677	834	135	4035	6164	NaN	...
mean	14159.247320	NaN	13.650211	433.505532	NaN	NaN	NaN	NaN	NaN	7.416127e+04	...
std	8336.864819	NaN	4.480178	249.152385	NaN	NaN	NaN	NaN	NaN	5.245257e+04	...
min	900.000000	NaN	5.320000	21.620000	NaN	NaN	NaN	NaN	NaN	2.500000e+03	...
25%	8000.000000	NaN	10.590000	256.230000	NaN	NaN	NaN	NaN	NaN	4.500000e+04	...
50%	12000.000000	NaN	13.330000	379.330000	NaN	NaN	NaN	NaN	NaN	6.400000e+04	...
75%	20000.000000	NaN	16.490000	568.640000	NaN	NaN	NaN	NaN	NaN	9.000000e+04	...
max	40000.000000	NaN	28.990000	1533.810000	NaN	NaN	NaN	NaN	NaN	2.500000e+06	...

11 rows × 27 columns

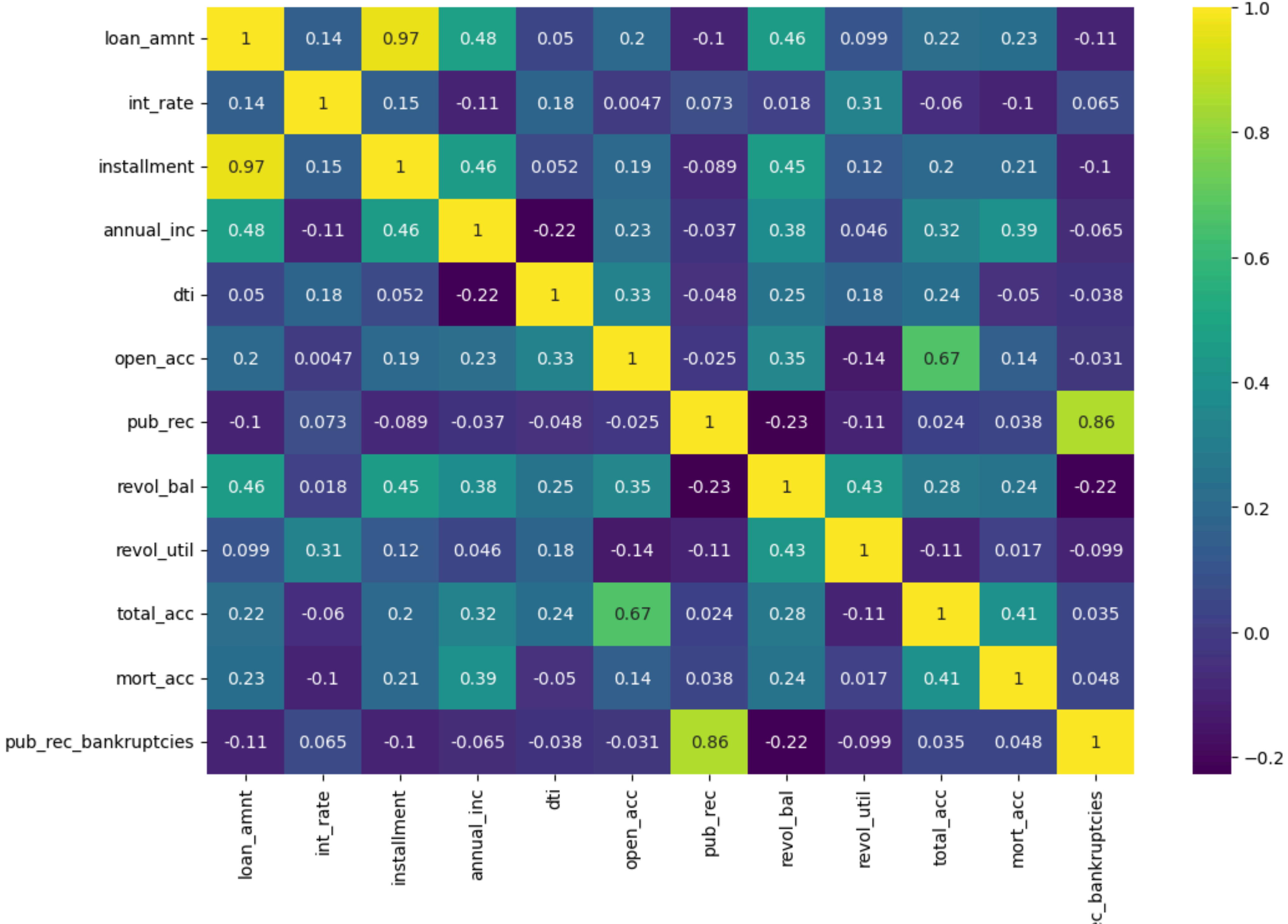
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12409 entries, 0 to 12408
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   loan_amnt             12409 non-null  float64
 1   term                  12409 non-null  object
 2   int_rate              12409 non-null  float64
 3   installment           12409 non-null  float64
 4   grade                 12409 non-null  object
 5   sub_grade             12409 non-null  object
 6   emp_title             11705 non-null  object
 7   emp_length            11843 non-null  object
 8   home_ownership        12409 non-null  object
```

```
 9   annual_inc            12409 non-null  float64
 10  verification_status   12409 non-null  object
 11  issue_d               12409 non-null  object
 12  loan_status           12409 non-null  object
 13  purpose               12409 non-null  object
 14  title                 12355 non-null  object
 15  dti                   12408 non-null  float64
 16  earliest_cr_line      12408 non-null  object
 17  open_acc              12408 non-null  float64
 18  pub_rec               12408 non-null  float64
 19  revol_bal             12408 non-null  float64
 20  revol_util            12401 non-null  float64
 21  total_acc             12408 non-null  float64
 22  initial_list_status   12408 non-null  object
 23  application_type      12408 non-null  object
 24  mort_acc              11287 non-null  float64
 25  pub_rec_bankruptcies  12393 non-null  float64
 26  address               12408 non-null  object
dtypes: float64(12), object(15)
memory usage: 2.6+ MB
```

```
# Heat Map
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(method='spearman'), annot=True, cmap='viridis')
plt.show()

# 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 t
# So, we can drop either one of those columns.
```

```
<ipython-input-10-a0844c95ef40>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fut
  sns.heatmap(data.corr(method='spearman'), annot=True, cmap='viridis')
```

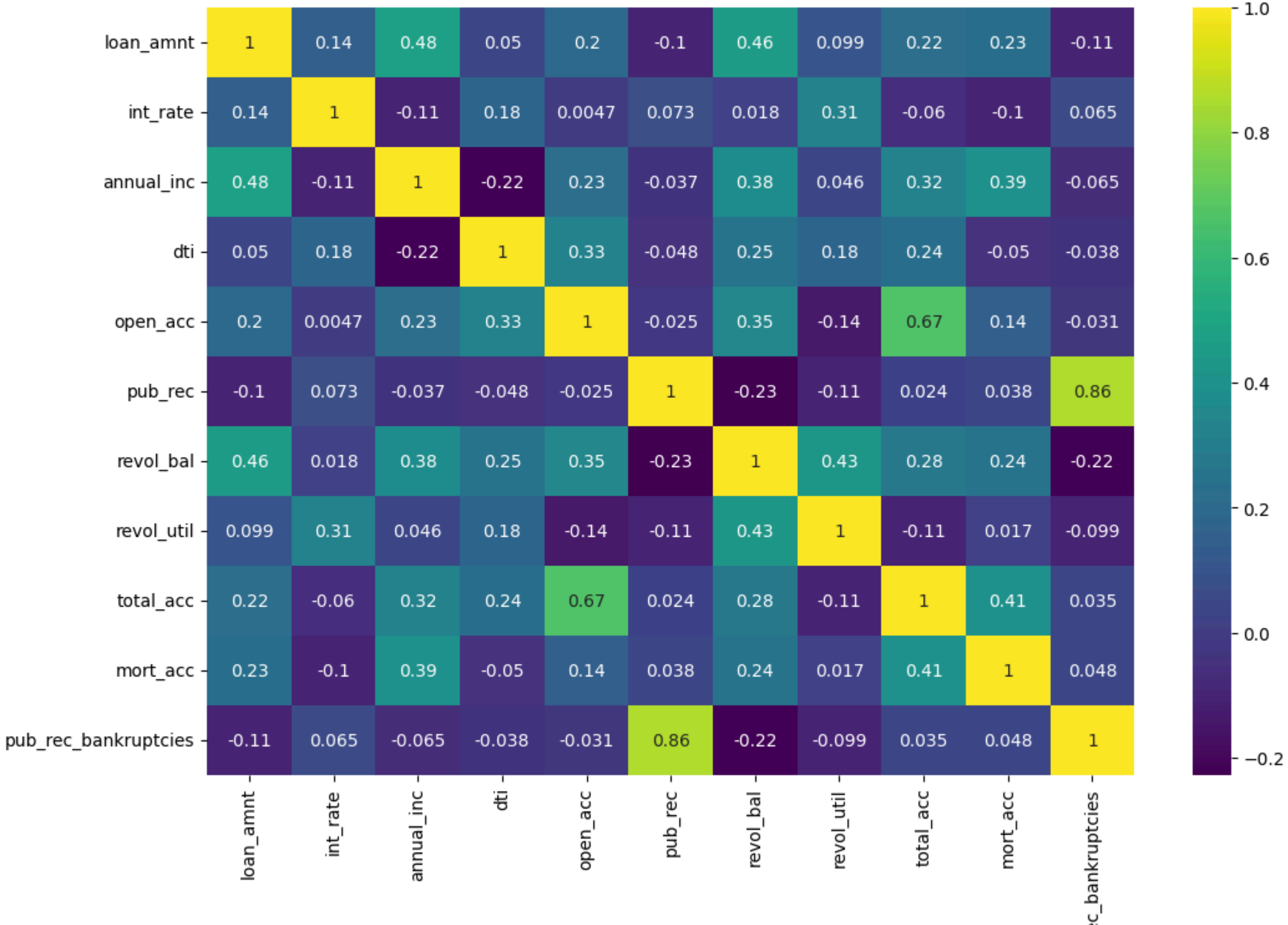


pub_re

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

```
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(method='spearman'), annot=True, cmap='viridis')
plt.show()
```

```
<ipython-input-12-f3907261aaff>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fut
  sns.heatmap(data.corr(method='spearman'), annot=True, cmap='viridis')
```



pub_re

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

	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	2380.0	14966.386555	8472.566619	1000.0	8381.25	13450.0	20000.0	39700.0
Fully Paid	10029.0	13967.703659	8293.237252	900.0	8000.00	12000.0	19275.0	40000.0

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

```
data['home_ownership'].value_counts()
# The majority of people have home ownership as Mortgage and Rent.
```

```
MORTGAGE    6164
RENT        5090
OWN         1152
OTHER          2
NONE           1
Name: home_ownership, dtype: int64
```

```
data.loc[(data.home_ownership == 'ANY') | (data.home_ownership == 'NONE'), 'home_ownership'] = 'OTHER'
data.home_ownership.value_counts()
# Combininging the minority classes as 'OTHER'.
```

```
MORTGAGE    6164
RENT        5090
OWN         1152
OTHER          3
Name: home_ownership, dtype: int64
```

```
# Checking the distribution of 'Other' -
data.loc[data['home_ownership']=='OTHER', 'loan_status'].value_counts()
```

```
Charged Off    2
Fully Paid     1
Name: loan_status, dtype: int64
```

```
data['issue_d'] = pd.to_datetime(data['issue_d'])
data['earliest_cr_line'] = pd.to_datetime(data['earliest_cr_line'])
# Coverting string to date-time format.
```

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

```
Debt consolidation           4780
Credit card refinancing      1687
Home improvement              478
Other                         423
Debt Consolidation            339
Major purchase                152
Consolidation                 117
debt consolidation            109
Debt Consolidation Loan        92
Business                       92
Medical expenses               69
Car financing                  68
Vacation                       59
Consolidation Loan             57
Moving and relocation          56
Credit Card Consolidation      51
consolidation                  50
Home Improvement               40
Home buying                    40
Credit Card Payoff             39
Name: title, dtype: int64
```

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

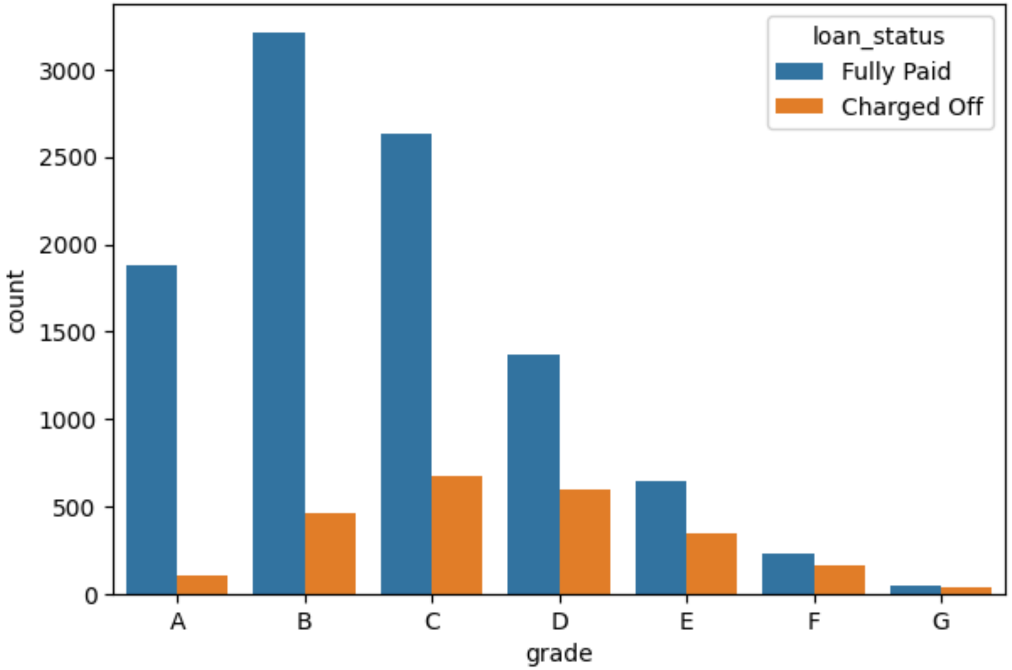
```
data.title.value_counts()[:10]
```

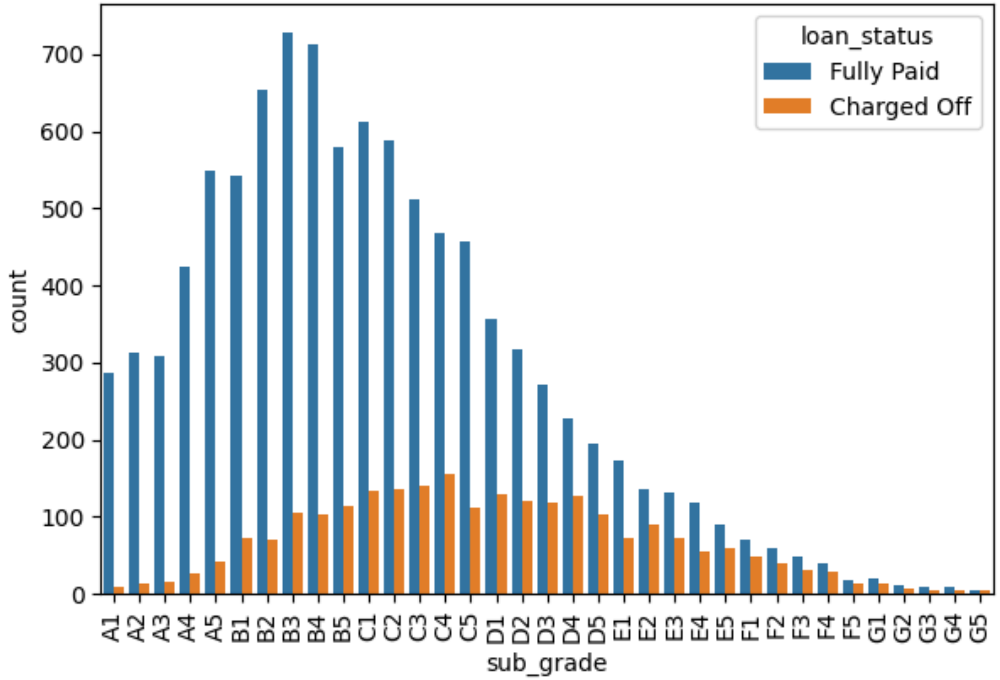
```
debt consolidation           5244
credit card refinancing      1696
```

```
home improvement              533
other                         424
consolidation                 169
major purchase                158
debt consolidation loan       104
business                       93
credit card consolidation      78
consolidation loan             75
Name: title, dtype: int64
```

```
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
grade = sorted(data.grade.unique().tolist())
sns.countplot(x='grade', data=data, hue='loan_status', order=grade)

plt.subplot(2, 2, 2)
sub_grade = sorted(data.sub_grade.unique().tolist())
g = sns.countplot(x='sub_grade', data=data, hue='loan_status', order=sub_grade)
g.set_xticklabels(g.get_xticklabels(), rotation=90);

# The grade of majority of people those who have fully paid the loan is 'B' and have subgrade 'B3'.

# So from where we can infer that people with grade 'B' and subgrade 'B3' are more likely to fully pay the loan.
```

```
<ipython-input-23-436f83565516>:10: UserWarning: FixedFormatter should only be used together with FixedLocator
  g.set_xticklabels(g.get_xticklabels(), rotation=90);
```



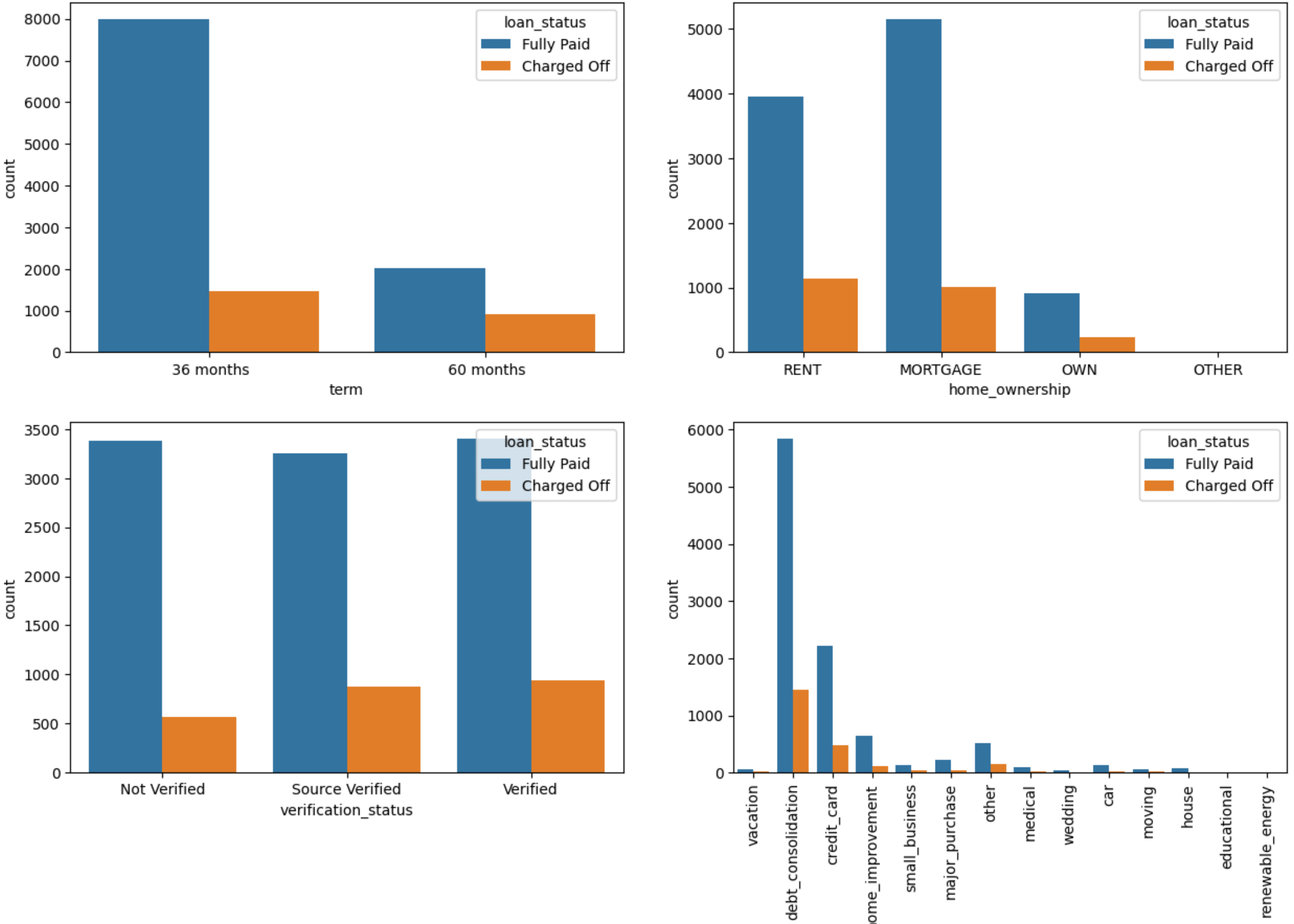



```
grade
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

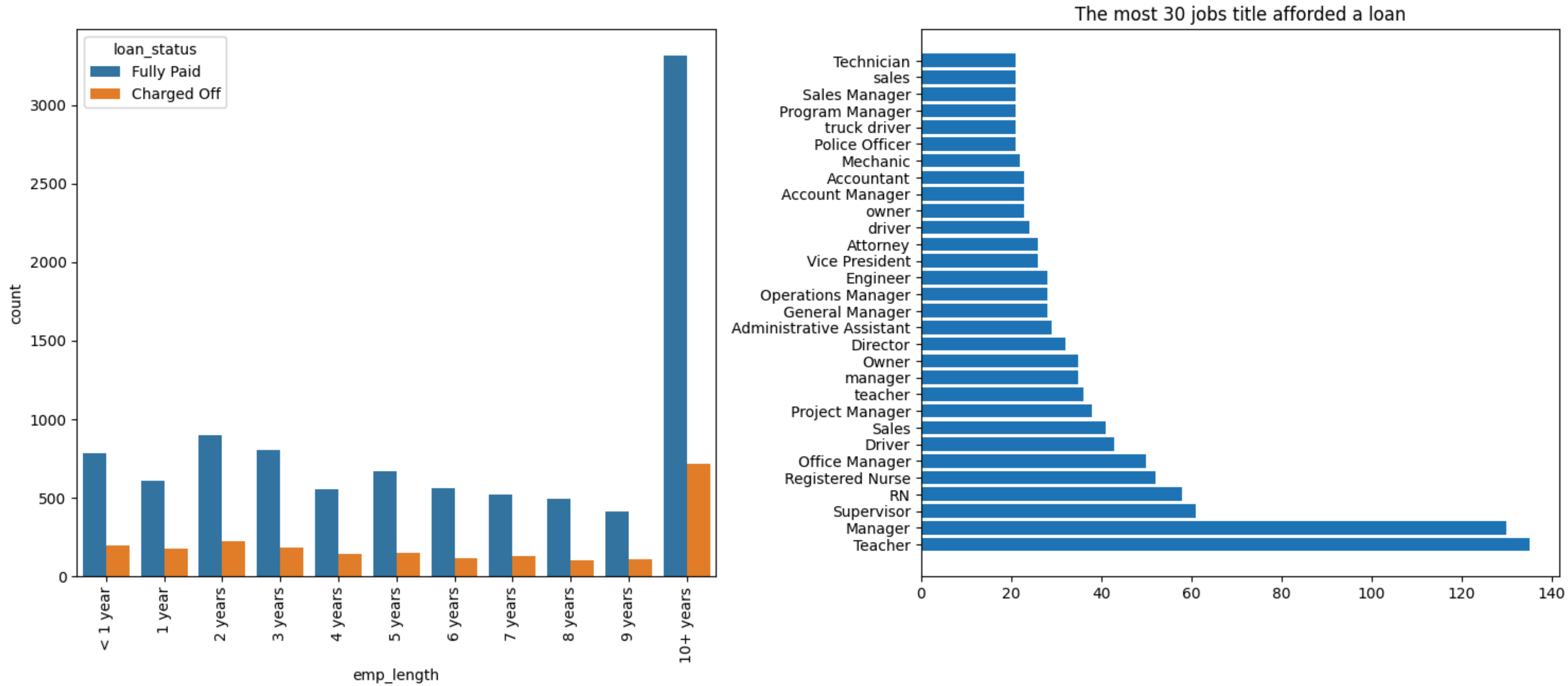
```
plt.figure(figsize=(15, 20))

plt.subplot(4, 2, 1)
sns.countplot(x='term', data=data, hue='loan_status')

plt.subplot(4, 2, 2)
sns.countplot(x='home_ownership', data=data, hue='loan_status')

plt.subplot(4, 2, 3)
sns.countplot(x='verification_status', data=data, hue='loan_status')

plt.subplot(4, 2, 4)
g = sns.countplot(x='purpose', data=data, hue='loan_status')
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```

```
<ipython-input-25-62191b24ebc8>:14: UserWarning: FixedFormatter should only be used together with FixedLocator
  g.set_xticklabels(g.get_xticklabels(), rotation=90);
```



```python
plt.figure(figsize=(15, 12))

plt.subplot(2, 2, 1)
order = ['< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years',
          '6 years', '7 years', '8 years', '9 years', '10+ years',]
g = sns.countplot(x='emp_length', data=data, hue='loan_status', order=order)
g.set_xticklabels(g.get_xticklabels(), rotation=90);

plt.subplot(2, 2, 2)
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.tight_layout()
# Manager and Teacher are the most afforded loan job titles.
```

```
<ipython-input-26-34ebc1631d63>:7: UserWarning: FixedFormatter should only be used together with FixedLocator
  g.set_xticklabels(g.get_xticklabels(), rotation=90);
```

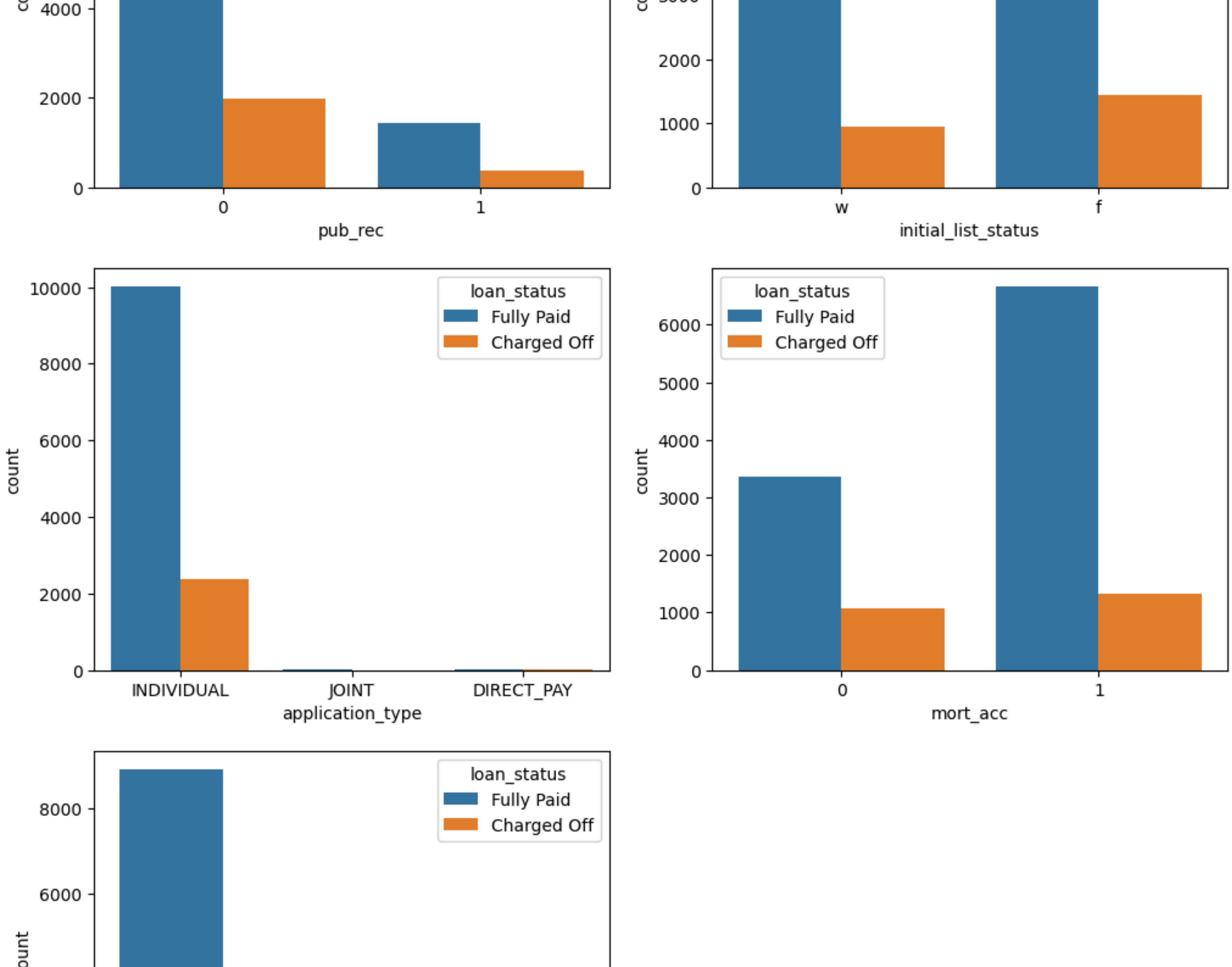


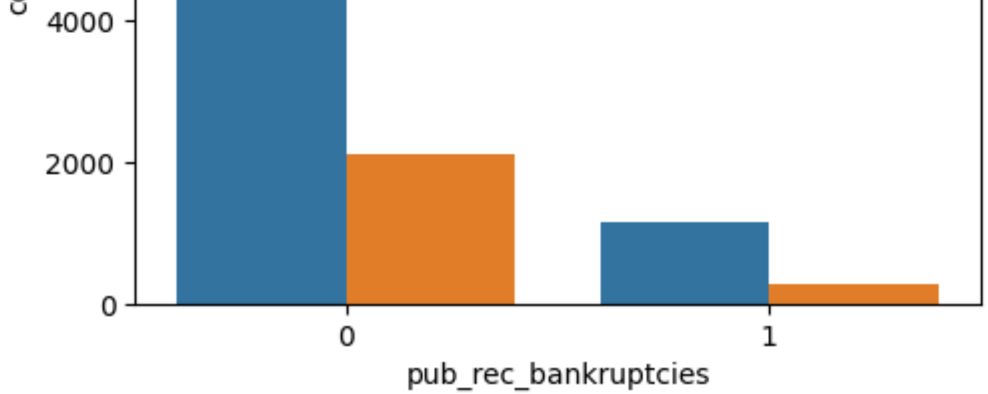
```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

def mort_acc(number):
    if number == 0.0:
        return 0
    else:
        return 1



def pub_rec_bankruptcies(number):
    if number == 0.0:
        return 0
    else:
        return 1
```

```
data['pub_rec'] = data.pub_rec.apply(pub_rec)
data['mort_acc'] = data.mort_acc.apply(mort_acc)
data['pub_rec_bankruptcies'] = data.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

```
plt.figure(figsize=(12, 30))

plt.subplot(6, 2, 1)
sns.countplot(x='pub_rec', data=data, hue='loan_status')

plt.subplot(6, 2, 2)
sns.countplot(x='initial_list_status', data=data, hue='loan_status')

plt.subplot(6, 2, 3)
sns.countplot(x='application_type', data=data, hue='loan_status')

plt.subplot(6, 2, 4)
sns.countplot(x='mort_acc', data=data, hue='loan_status')

plt.subplot(6, 2, 5)
sns.countplot(x='pub_rec_bankruptcies', data=data, hue='loan_status')
```

```
plt.show()
```


4000
2000
0
0
1
pub_rec
2000
1000
0
w
f
initial_list_status
10000
8000
6000
4000
2000
0
count
loan_status
Fully Paid
Charged Off
INDIVIDUAL
JOINT
DIRECT_PAY
application_type
6000
5000
4000
3000
2000
1000
0
count
loan_status
Fully Paid
Charged Off
0
1
mort_acc
8000
6000
loan_status
Fully Paid
Charged Off




```
# Mapping of target variable -
data['loan_status'] = data.loan_status.map({'Fully Paid':0, 'Charged Off':1})
```

```
data.isnull().sum()/len(data)*100
```

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.673302
emp_length              4.561206
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.435168
dti                     0.008059
```

```
earliest_cr_line        0.008059
open_acc                0.008059
pub_rec                 0.000000
revol_bal               0.008059
revol_util              0.064469
total_acc               0.008059
initial_list_status     0.008059
application_type        0.008059
mort_acc                0.000000
pub_rec_bankruptcies    0.000000
address                 0.008059
dtype: float64
```

```
data.groupby(by='total_acc').mean()
```

```
<ipython-input-34-05a6a8313bf2>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. I
  data.groupby(by='total_acc').mean()
```

	loan_amnt	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_util	mort_acc	pub_
total_acc											
2.0	5000.000000	7.430000	200000.000000	0.000000	0.280000	2.000000	0.000000	3164.000000	13.700000	1.000000	
3.0	6912.500000	16.468750	39503.000000	0.250000	6.830000	2.750000	0.000000	4517.500000	43.162500	0.375000	
4.0	7742.777778	14.153556	40245.622222	0.133333	7.424222	3.466667	0.022222	5310.155556	54.766667	0.488889	
5.0	8649.264706	14.584412	46484.482353	0.176471	10.165294	3.941176	0.073529	7163.352941	54.597059	0.367647	
6.0	9264.606742	15.448202	47844.917303	0.235955	11.712472	4.674157	0.067416	6418.674157	61.007865	0.280899	
...	...	...	...	...	...	...	...	...	...	...	
87.0	12000.000000	13.980000	75580.000000	0.000000	28.040000	35.000000	0.000000	12181.000000	31.300000	1.000000	
89.0	16000.000000	24.080000	103000.000000	1.000000	20.180000	18.000000	1.000000	23305.000000	68.500000	1.000000	
97.0	35000.000000	17.860000	485000.000000	0.000000	8.760000	21.000000	0.000000	32812.000000	23.900000	1.000000	
104.0	18825.000000	11.530000	78000.000000	0.000000	27.280000	10.000000	0.000000	19179.000000	66.800000	1.000000	
105.0	35000.000000	18.250000	130685.000000	0.000000	26.310000	27.000000	0.000000	17406.000000	53.100000	1.000000	

83 rows × 11 columns

```
total_acc_avg = data.groupby(by='total_acc').mean().mort_acc
# Saving mean of mort_acc according to total_acc_avg (you can pick any variable for your understanding)
```

```
<ipython-input-35-81314869079b>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. I
  total_acc_avg = data.groupby(by='total_acc').mean().mort_acc
```

```
def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
```

```
data['mort_acc'] = data.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']), axis=1)
```

```
data.isnull().sum()/len(data)*100
```

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.673302
emp_length              4.561206
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.435168
dti                     0.008059
earliest_cr_line        0.008059
open_acc                0.008059
pub_rec                 0.000000
revol_bal               0.008059
revol_util              0.064469
total_acc               0.008059
initial_list_status     0.008059
application_type        0.008059
mort_acc                0.000000
pub_rec_bankruptcies    0.000000
```

```
address                  0.008059
dtype: float64
```

```
# Current no. of rows -
data.shape
```

```
(12409, 26)
```

```
# Dropping rows with null values -
data.dropna(inplace=True)
```

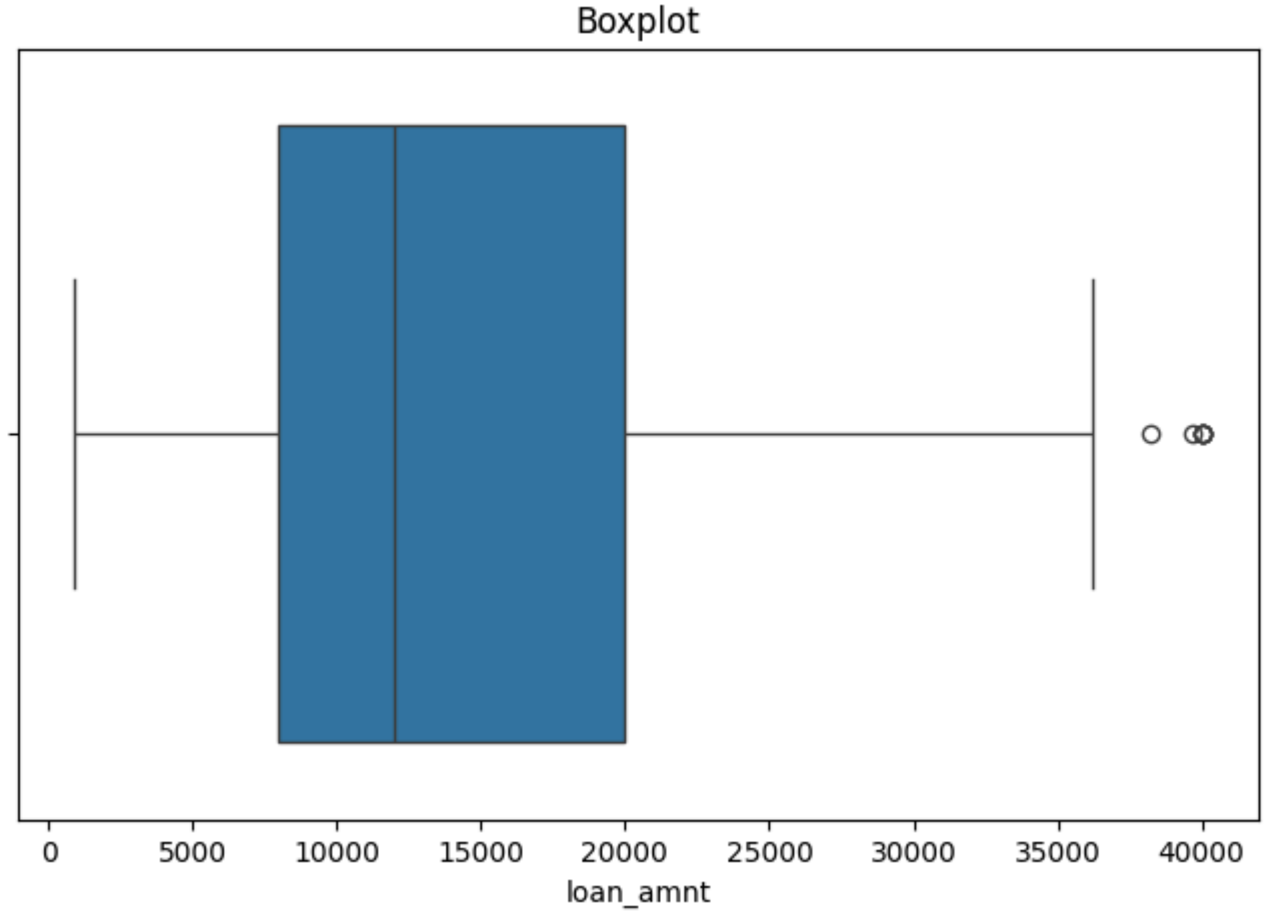
```
# Remaining no. of rows -
data.shape
```

```
(11645, 26)
```

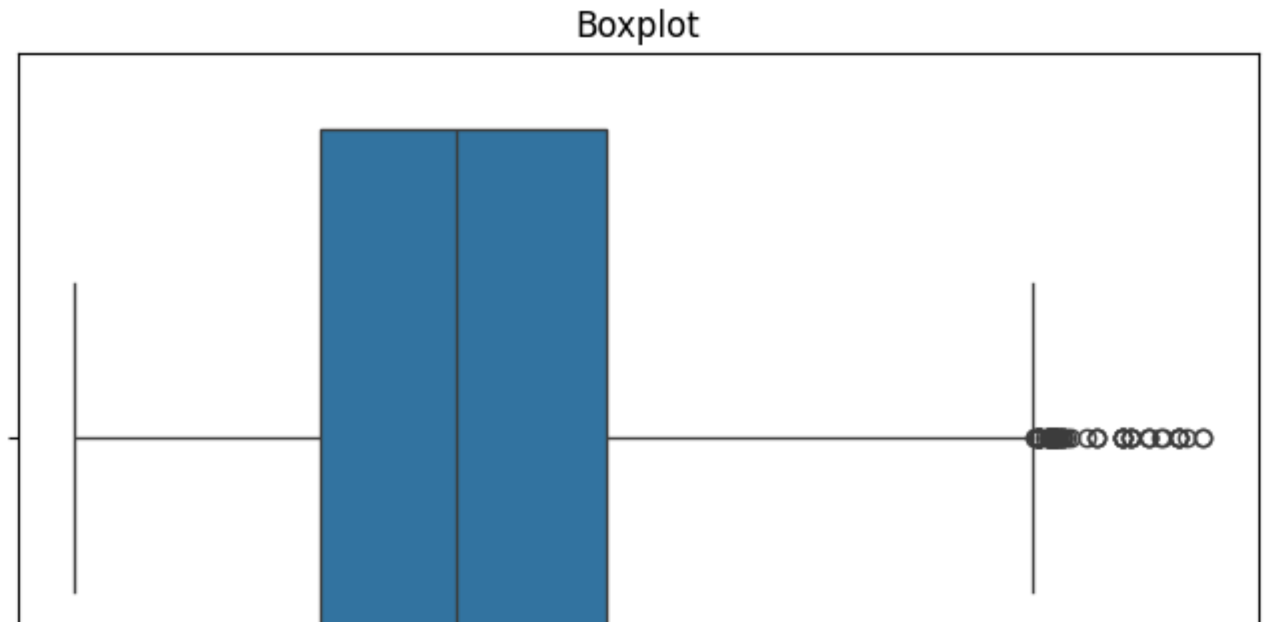
```
numerical_data = data.select_dtypes(include='number')
num_cols = numerical_data.columns
len(num_cols)
```

```
12
```

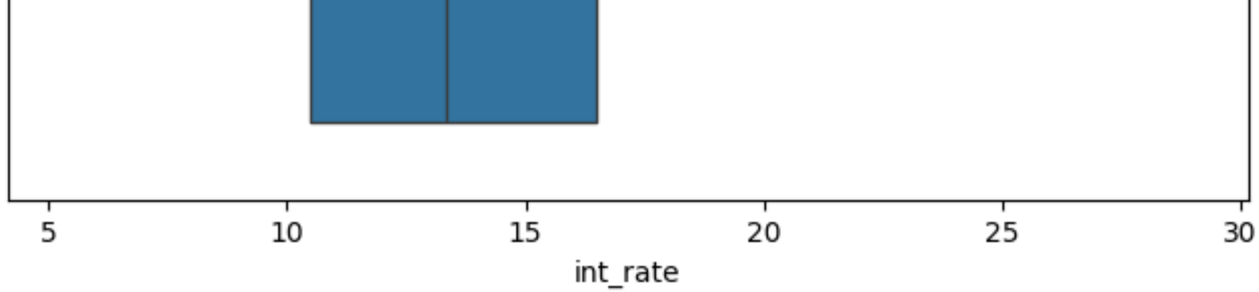
```
# outier detection
def box_plot(col):
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=data[col])
    plt.title('Boxplot')
    plt.show()

for col in num_cols:
    box_plot(col)
```



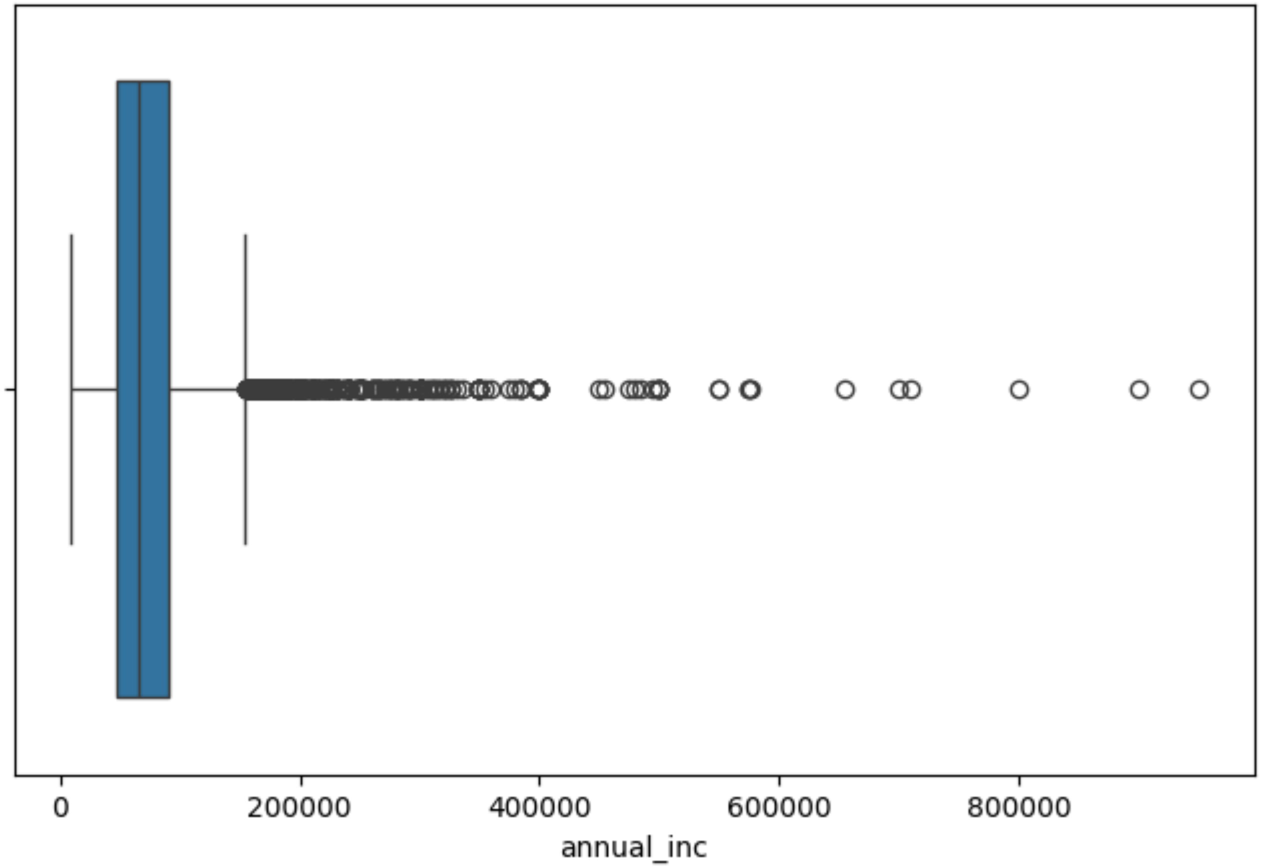
Boxplot
0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt



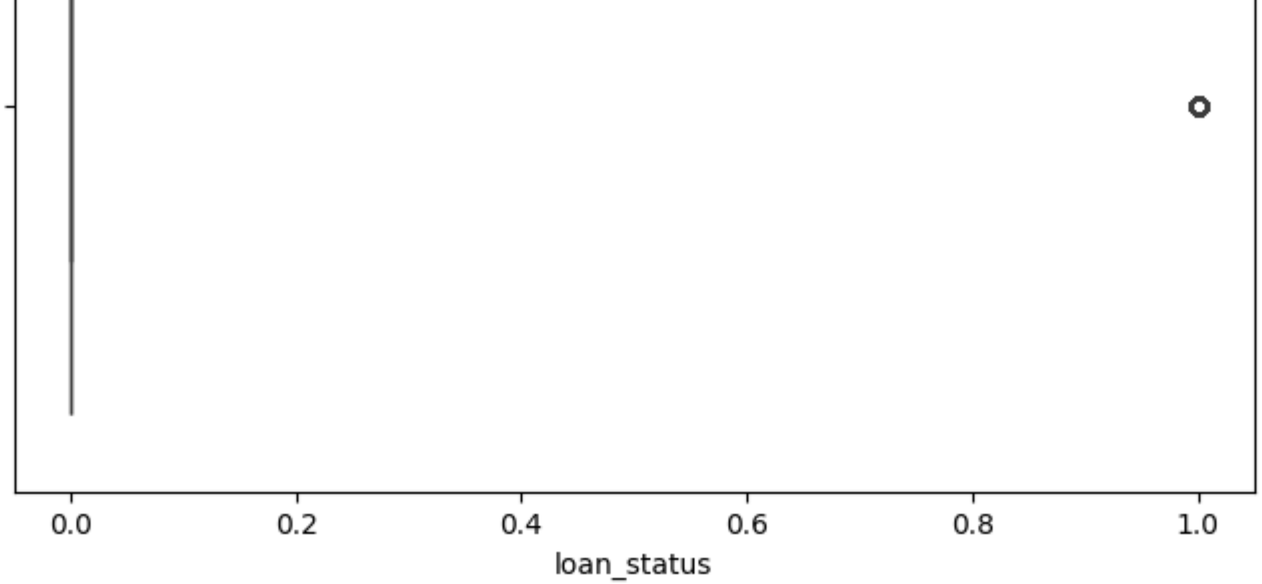
Boxplot

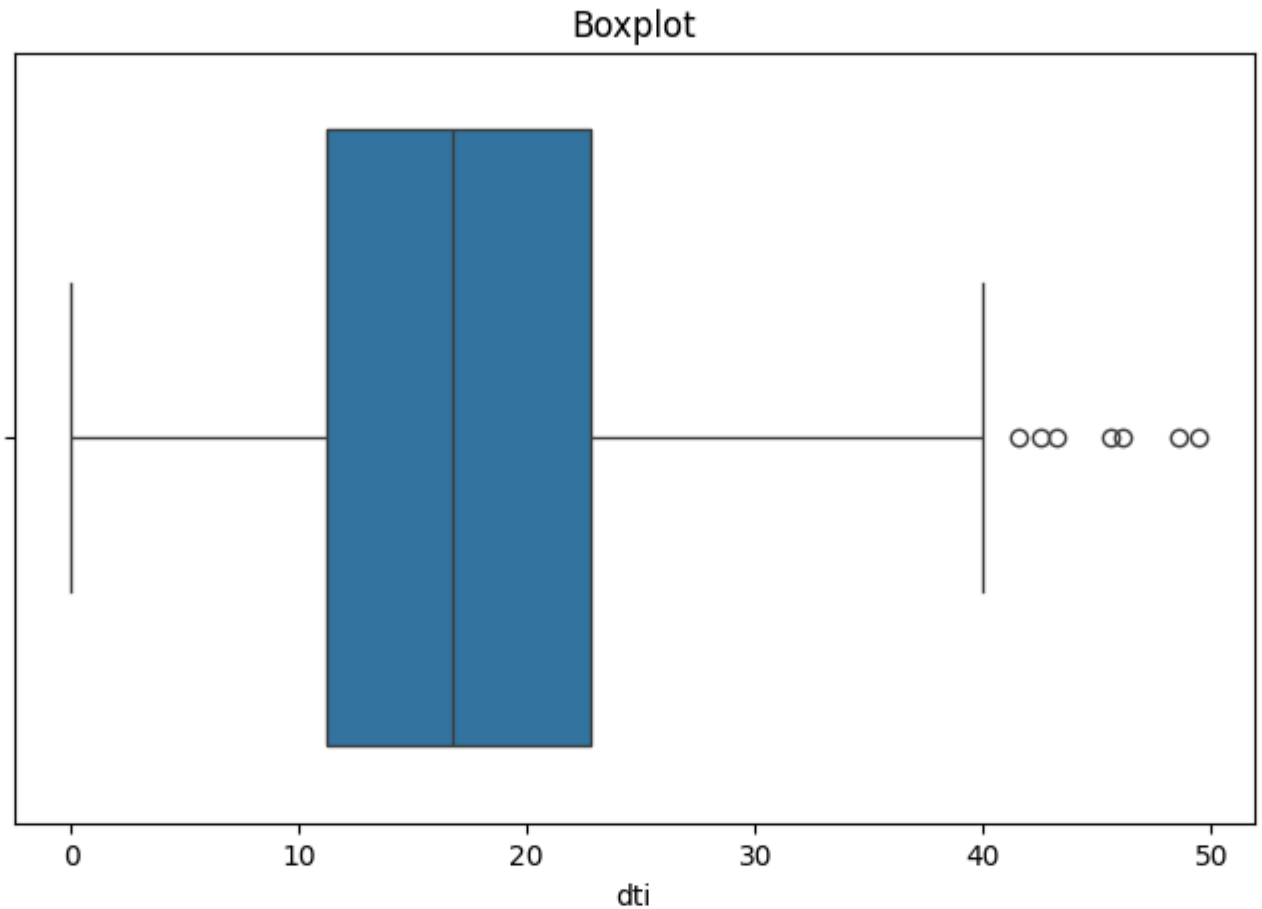


Boxplot

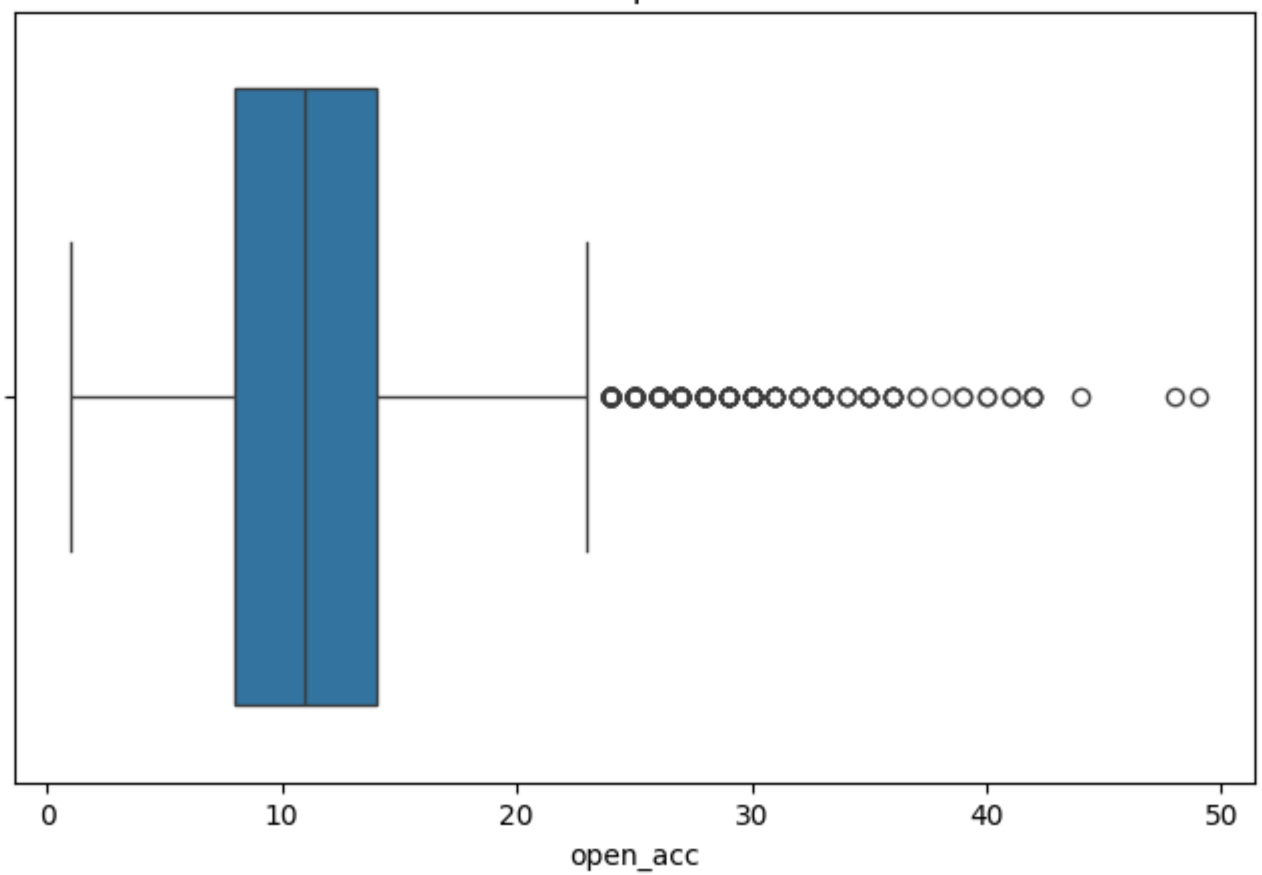


Boxplot

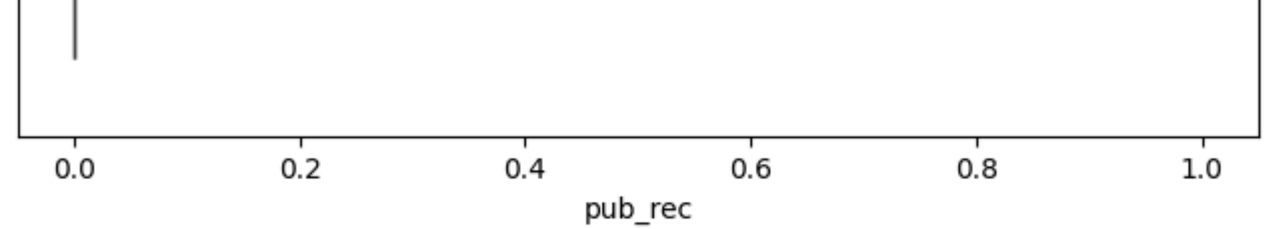




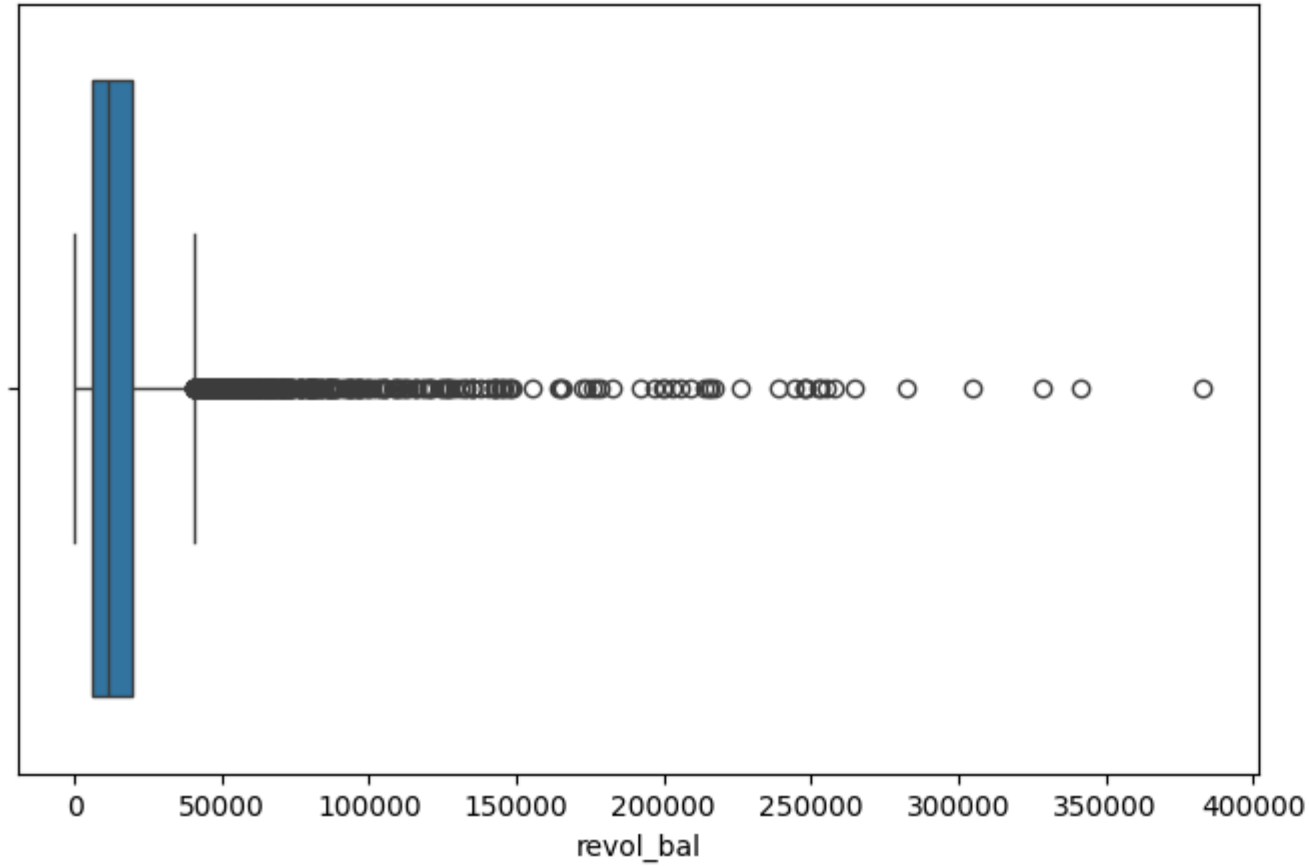
Boxplot



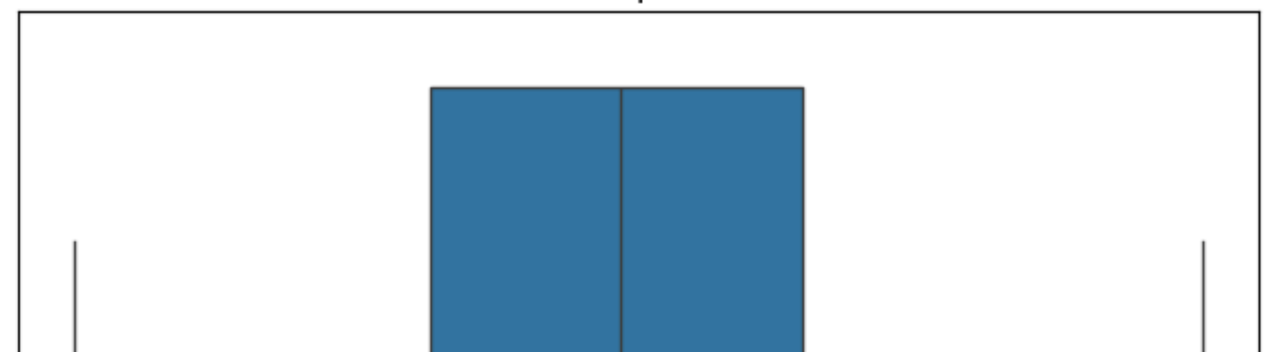
Boxplot

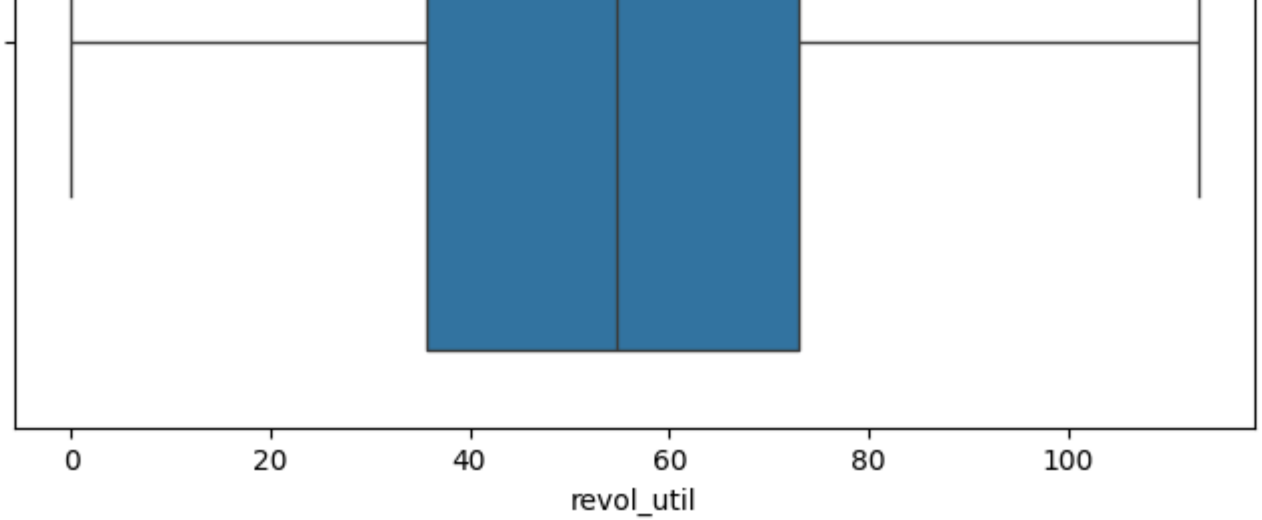


Boxplot

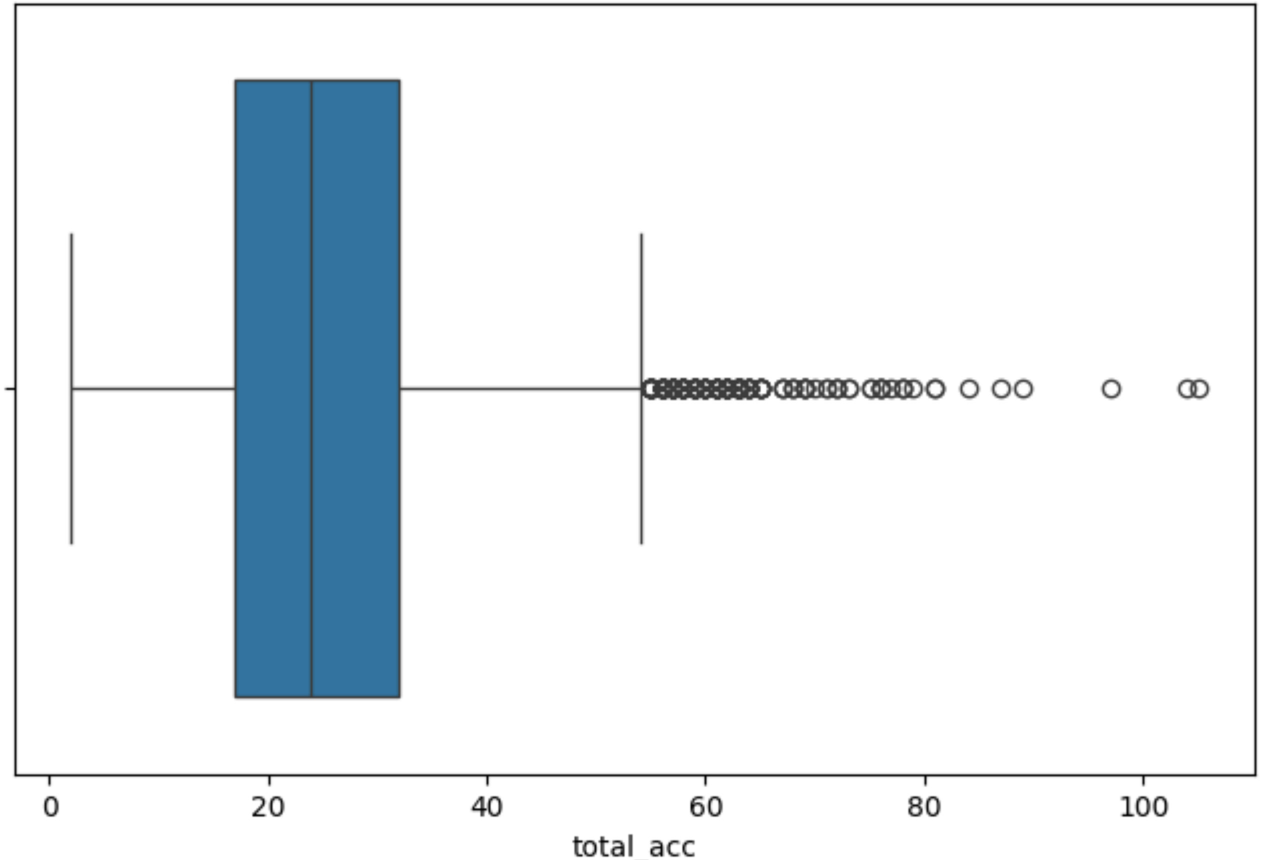


Boxplot

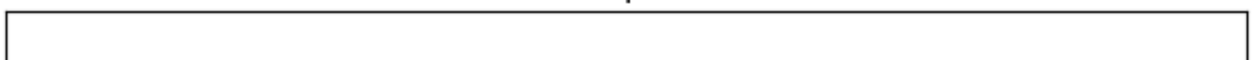


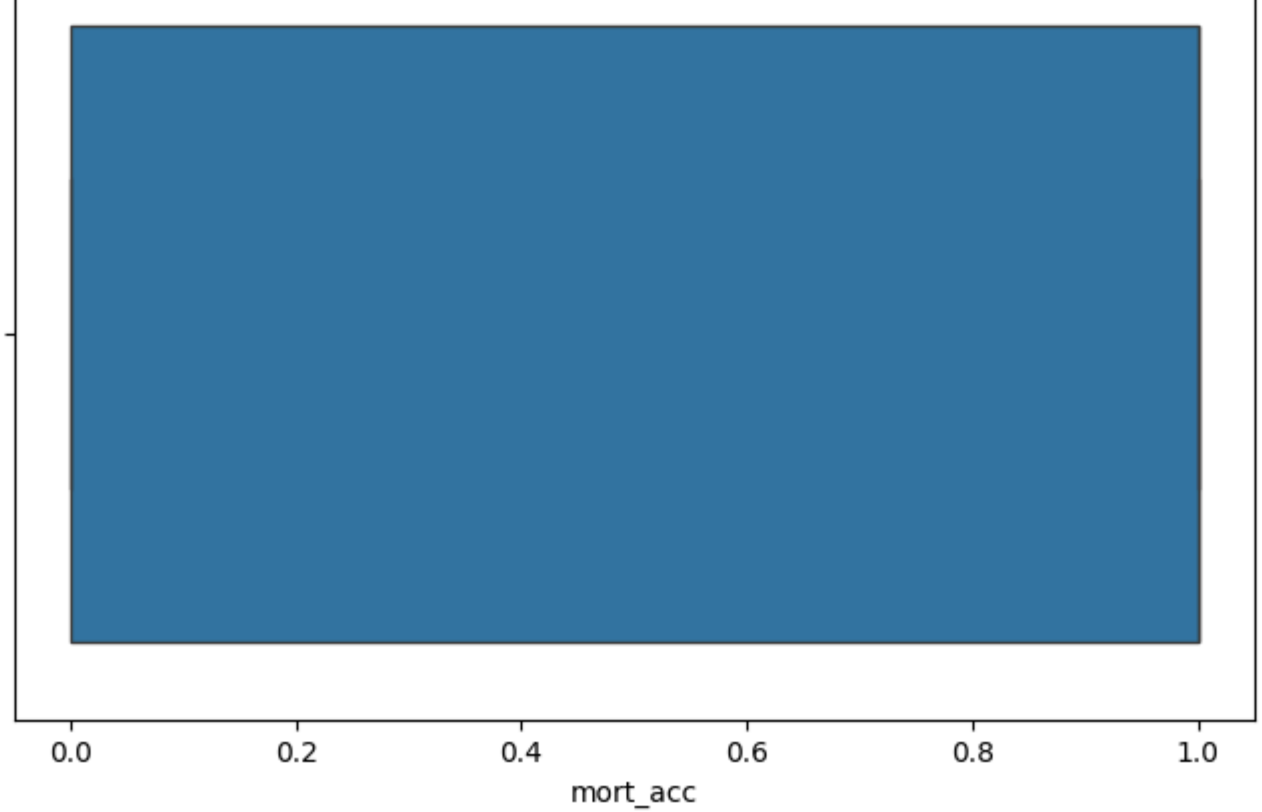


Boxplot

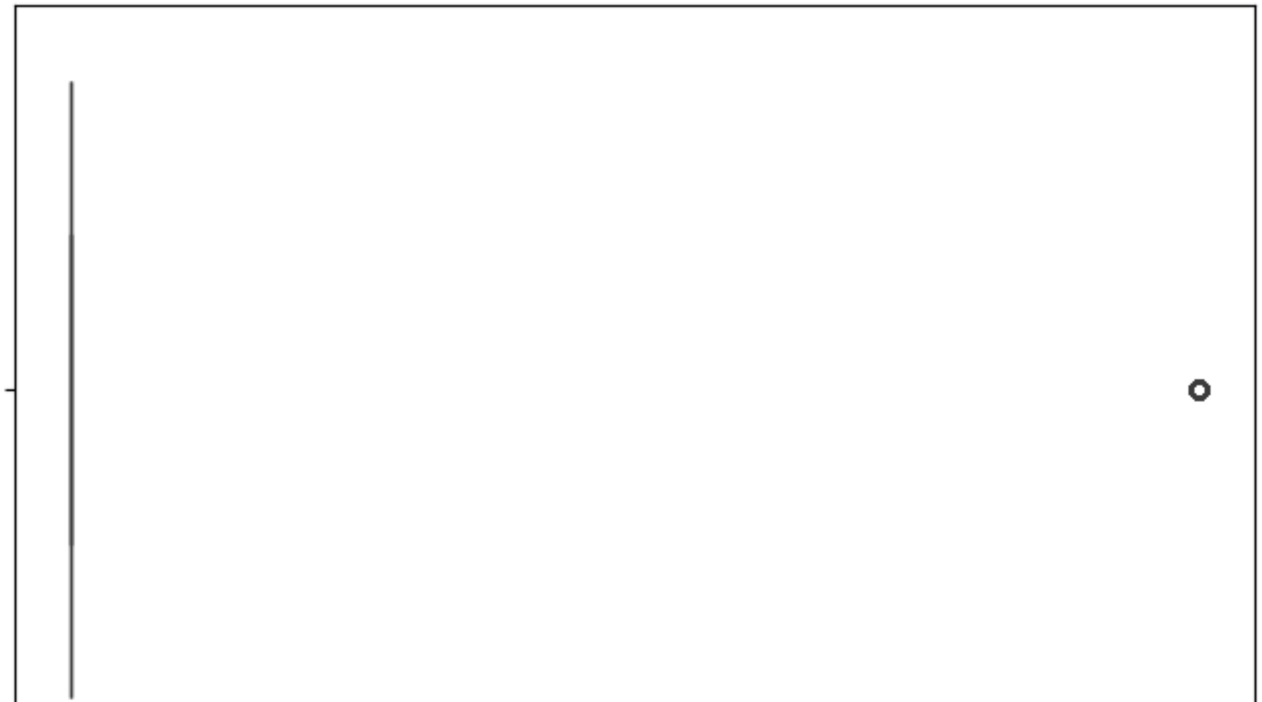


Boxplot





Boxplot




```
for col in num_cols:
    mean = data[col].mean()
    std = data[col].std()

    upper_limit = mean+3*std
    lower_limit = mean-3*std

    data = data[(data[col]<upper_limit) & (data[col]>lower_limit)]

data.shape
```

```
(11051, 26)
```

```
# Term -
data.term.unique()
```

```
array([' 36 months', ' 60 months'], dtype=object)
```

```
term_values = {' 36 months': 36, ' 60 months': 60}
data['term'] = data.term.map(term_values)
```

```
# Initial List Status -
data['initial_list_status'].unique()
```

```
array(['w', 'f'], dtype=object)
```

```
list_status = {'w': 0, 'f': 1}
data['initial_list_status'] = data.initial_list_status.map(list_status)
```

```
# Let's fetch ZIP from address and then drop the remaining details -
data['zip_code'] = data.address.apply(lambda x: x[-5:])
```

```
data['zip_code'].value_counts(normalize=True)*100
```

```
22690    14.894580
70466    14.487377
30723    13.998733
48052    13.781558
29597    12.007963
00813    11.501222
05113    11.401683
93700     2.687540
86630     2.642295
11650     2.597050
Name: zip_code, dtype: float64
```

```
# Dropping some variables which IMO we can let go for now -
data.drop(columns=['issue_d', 'emp_title', 'title', 'sub_grade',
                   'address', 'earliest_cr_line', 'emp_length'],
                   axis=1, inplace=True)
```

```
# One Hot Encoding
dummies = ['purpose', 'zip_code', 'grade', 'verification_status', 'application_type', 'home_ownership']
data = pd.get_dummies(data, columns=dummies, drop_first=True)
```

```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

data.head()
```

	loan_amnt	term	int_rate	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	initial_list_s
0	10000.0	36	11.44	117000.0	0	26.24	16.0	0	36369.0	41.8	25.0	
1	8000.0	36	11.99	65000.0	0	22.05	17.0	0	20131.0	53.3	27.0	
2	15600.0	36	10.49	43057.0	0	12.79	13.0	0	11987.0	92.2	26.0	
3	7200.0	36	6.49	54000.0	0	2.60	6.0	0	5472.0	21.5	13.0	
4	24375.0	60	17.27	55000.0	1	33.95	13.0	0	24584.0	69.8	43.0	

```
data.shape
```

```
(11051, 49)
```

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
                                                    stratify=y, random_state=42)
print(X_train.shape)
print(X_test.shape)
```

```
(7735, 48)
(3316, 48)
```

```
# Scaling - MinMax Scaling
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Logistic Regression
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred = logreg.predict(X_test)
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.format(logreg.score(X_test, y_test)))
```

```
Accuracy of Logistic Regression Classifier on test set: 0.896
```

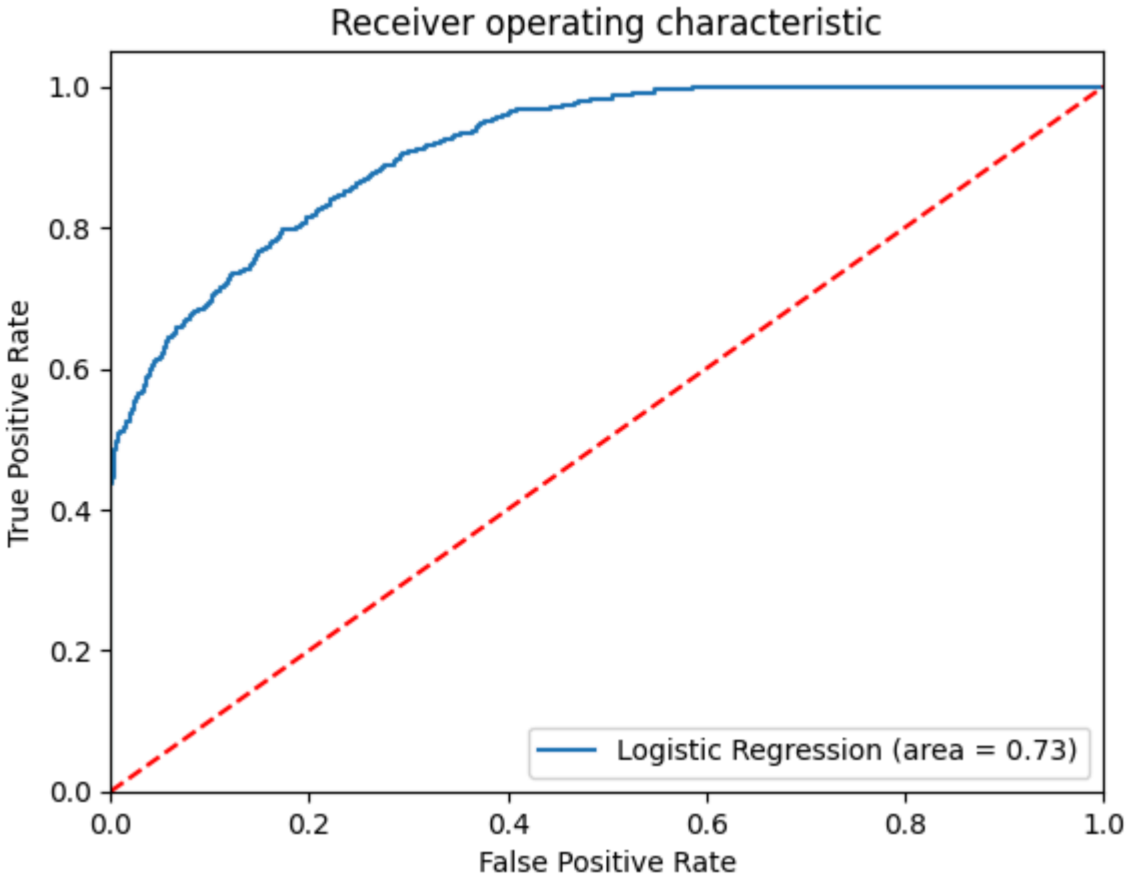
```
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[2678   10]
 [ 334  294]]
```

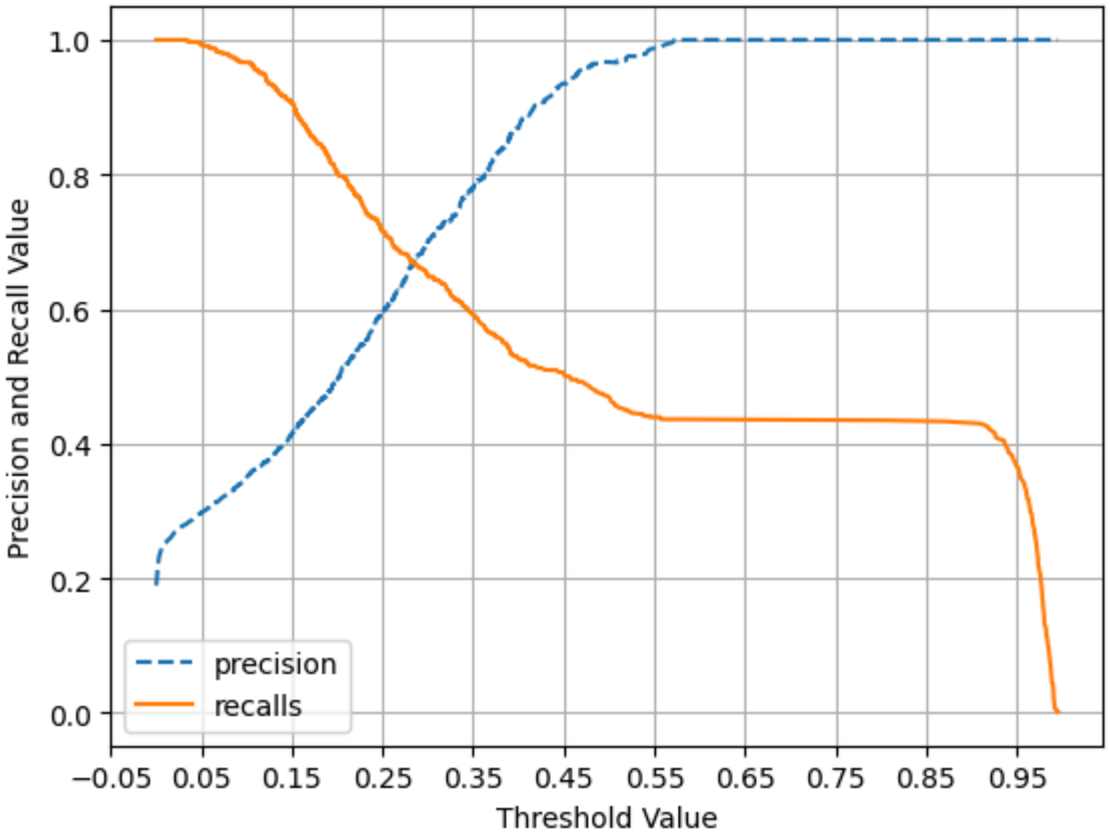
```
# classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      2688
           1       0.97      0.47      0.63       628

    accuracy                           0.90      3316
   macro avg       0.93      0.73      0.79      3316
weighted avg       0.90      0.90      0.88      3316
```

```
# ROC Curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (area = 0.73)

```
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, logreg.predict_proba(X_test)[:,1])
```



```
# Multi Colinearity check - VIF
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)[:5]
```

	Feature	VIF
43	application_type_INDIVIDUAL	159.34
2	int_rate	123.89
14	purpose_debt_consolidation	47.70
1	term	27.85
13	purpose_credit_card	18.13

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

	Feature	VIF
2	int_rate	102.84
14	purpose_debt_consolidation	26.48
1	term	24.89
5	open_acc	13.67
9	total_acc	12.35

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

	Feature	VIF
1	term	23.84
13	purpose_debt_consolidation	21.81
4	open_acc	13.51
8	total_acc	12.35
7	revol_util	9.17

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

	Feature	VIF
12	purpose_debt_consolidation	18.54
3	open_acc	13.51
7	total_acc	12.33
6	revol_util	9.16
1	annual_inc	8.76

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

	Feature	VIF
3	open_acc	12.96
7	total_acc	12.31
6	revol_util	8.41
1	annual_inc	8.39
2	dti	7.67

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

	Feature	VIF
1	annual_inc	8.28
5	revol_util	8.10
6	total_acc	8.07
2	dti	7.05
0	loan_amnt	6.68

```
X = scaler.fit_transform(X)

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

```
Cross Validation accuracy: 0.890
```

```
!pip install imbalanced-learn==0.8.0
```

```
Collecting imbalanced-learn==0.8.0
  Downloading imbalanced_learn-0.8.0-py3-none-any.whl (206 kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 206.5/206.5 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn==0.8.0) (1.25.
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn==0.8.0) (1.11.
Requirement already satisfied: scikit-learn>=0.24 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn==0.8.0) (
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn==0.8.0) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.24->imba
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.10.1
    Uninstalling imbalanced-learn-0.10.1:
      Successfully uninstalled imbalanced-learn-0.10.1
Successfully installed imbalanced-learn-0.8.0
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())
```

```
print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res == 0)))
```

```
After OverSampling, the shape of train_X: (12538, 48)
After OverSampling, the shape of train_y: (12538,)

After OverSampling, counts of label '1': 6269
After OverSampling, counts of label '0': 6269
```

```
lr1 = LogisticRegression(max_iter=1000)
lr1.fit(X_train_res, y_train_res)
predictions = lr1.predict(X_test)

# Classification Report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.79      0.86      2688
           1       0.48      0.82      0.60       628
```

```
    accuracy                           0.80      3316
   macro avg       0.71      0.81      0.73      3316
weighted avg       0.86      0.80      0.81      3316
```

```
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, lr1.predict_proba(X_test)[:,1])
```

```
# Observations have been written at the cell level
# From the values mentioned above it can be observed that
the model is performing as expected and no further
hypertuning can improve the preformance.
# The low precision value for class 0 can be due to the
imbalance of data for the same, if more real time data
for class 0 can be provided, the model can be trained
better and the performance might increase.
# Also since the data consists of a lot of categorical
columns a different ML model might prove better in
predicting the outcome than Logistic Regression.
# The model's precision value of 0.90 signifies that it
accurately predicts the likelihood of loan repayment in
90% of cases.
# The model's precision value of 0.38 for charged-off
loans indicates that, among the instances predicted as
charged off, only 38% were correctly classified,
emphasizing a lower accuracy in predicting this specific
class.
# The model's sensitivity value of 0.71 for loan
repayment signifies that it accurately identifies 71% of
the instances where loans are repaid, demonstrating its
ability to effectively capture a significant portion of
the actual loan repayment cases.
# The model's sensitivity value of 0.71 for charged-off
loans signifies that it correctly identifies 71% of the
actual charged-off instances, reflecting its ability to
capture a substantial portion of the relevant cases for
this class.
# The features that heavily affected the models outcome
are
# grade - LoanTap assigned loan grade (Risk ratings by
LoanTap)
# pub_rec - Negative records on borrower's public credit
profile.
# From the analysis performed it can also be observed
that the applicants for regions with pincodes('11650'm
'86630' and '93700') have not made any loan repayment. It
can be inferred that either
# The data is missing w.r.t. loan repayment for these
regions or
```

```
# The applicants from regions with pincodes('11650'm
'86630' and '93700') are highly unlikely to repay the
loan granted by LoanTap.
# LoanTap should carefully review the applicants
belonging to above regions.
```