In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder, LabelBinarizer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.feature_selection._base import SelectorMixin
from sklearn.feature_extraction.text import _VectorizerMixin
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, classification_report, precision_recall_curve, plot_prec
from sklearn.metrics import PrecisionRecallDisplay, precision_score

from imblearn.over_sampling import SMOTE
```

In [2]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_colwidth', None)
```

In [3]:
```
df = pd.read_csv("D:ScalerData/loan_tap.csv")
```

EDA

In [4]:
```
df.shape
```

Out[4]:
```
(396030, 27)
```

In [5]:
```
description_dict = {
"loan_amnt" : "The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces
,"term" : "The number of payments on the loan. Values are in months and can be either 36 or 60."
,"int_rate" : "Interest Rate on the loan"
,"installment" : "The monthly payment owed by the borrower if the loan originates."
,"grade" : "LoanTap assigned loan grade"
,"sub_grade" : "LoanTap assigned loan subgrade"
,"emp_title" : "The job title supplied by the Borrower when applying for the loan."
,"emp_length" : "Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means t
```

```
,"home_ownership" : "The home ownership status provided by the borrower during registration or obtained from the credit report."
,"annual_inc" : "The self-reported annual income provided by the borrower during registration."
,"verification_status" : "Indicates if income was verified by LoanTap, not verified, or if the income source was verified"
,"issue_d" : "The month which the loan was funded"
,"loan_status" : "Current status of the loan - Target Variable"
,"purpose" : "A category provided by the borrower for the loan request."
,"title" : "The loan title provided by the borrower"
,"dti" : "A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage a
,"earliest_cr_line" : "The month the borrower's earliest reported credit line was opened"
,"open_acc" : "The number of open credit lines in the borrower's credit file."
,"pub_rec" : "Number of derogatory public records"
,"revol_bal" : "Total credit revolving balance"
,"revol_util" : "Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolvi
,"total_acc" : "The total number of credit lines currently in the borrower's credit file"
,"initial_list_status" : "The initial listing status of the loan. Possible values are – W, F"
,"application_type" : "Indicates whether the loan is an individual application or a joint application with two co-borrowers"
,"mort_acc" : "Number of mortgage accounts."
,"pub_rec_bankruptcies" : "Number of public record bankruptcies"
,"address": "Address of the individual"
}
```

In [6]:
```
desc_df = df.describe(include="all").transpose()
```

In [7]:
```
pd.concat([desc_df, pd.DataFrame(list(description_dict.items())).rename(columns={0: "columns", 1:"description"}).set_index("colu
```

Out[7]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	description
loan_amnt	396030.0	NaN	NaN	NaN	14113.888089	8357.441341	500.0	8000.0	12000.0	20000.0	40000.0	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
term	396030	2	36 months	302005	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The number of payments on the loan Values are in months and can be either 36 or 60
int_rate	396030.0	NaN	NaN	NaN	13.6394	4.472157	5.32	10.49	13.33	16.49	30.99	Interest Rate on the loan
installment	396030.0	NaN	NaN	NaN	431.849698	250.72779	16.08	250.33	375.43	567.3	1533.81	The monthly payment owed by the borrower if the loan originates
grade	396030	7	B	116018	NaN	NaN	NaN	NaN	NaN	NaN	NaN	LoanTap assigned loan grade

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	description
sub_grade	396030	35	B3	26655	NaN	NaN	NaN	NaN	NaN	NaN	NaN	LoanTap assigned loan subgrade
emp_title	373103	173105	Teacher	4389	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The job title supplied by the Borrower wher applying for the loan
emp_length	377729	11	10+ years	126041	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Employment length in years Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years
home_ownership	396030	6	MORTGAGE	198348	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The home ownership status provided by the borrower during registration or obtained from the credit report

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	description
annual_inc	396030.0	NaN	NaN	NaN	74203.175798	61637.621158	0.0	45000.0	64000.0	90000.0	8706582.0	The self-reported annual income provided by the borrower during registration
verification_status	396030	3	Verified	139563	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Indicates if income was verified by LoanTap, not verified, or if the income source was verified
issue_d	396030	115	Oct-2014	14846	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The month which the loan was funded
loan_status	396030	2	Fully Paid	318357	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Current status of the loan - Target Variable
purpose	396030	14	debt_consolidation	234507	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A category provided by the borrower for the loan request
title	394275	48817	Debt consolidation	152472	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The loan title provided by the borrower

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	description
dti	396030.0	NaN	NaN	NaN	17.379514	18.019092	0.0	11.28	16.91	22.98	9999.0	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations excluding mortgage and the requested LoanTap loan, divided by the borrower's self-reported monthly income
earliest_cr_line	396030	684	Oct-2000	3017	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The month the borrower's earliest reported credit line was opened
open_acc	396030.0	NaN	NaN	NaN	11.311153	5.137649	0.0	8.0	10.0	14.0	90.0	The number of open credit lines in the borrower's credit file
pub_rec	396030.0	NaN	NaN	NaN	0.178191	0.530671	0.0	0.0	0.0	0.0	86.0	Number of derogatory public records

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	description
revol_bal	396030.0	NaN	NaN	NaN	15844.539853	20591.836109	0.0	6025.0	11181.0	19620.0	1743266.0	Total credit revolving balance
revol_util	395754.0	NaN	NaN	NaN	53.791749	24.452193	0.0	35.8	54.8	72.9	892.3	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit
total_acc	396030.0	NaN	NaN	NaN	25.414744	11.886991	2.0	17.0	24.0	32.0	151.0	The total number of credit lines currently in the borrower's credit file
initial_list_status	396030	2	f	238066	NaN	NaN	NaN	NaN	NaN	NaN	NaN	The initial listing status of the loan Possible values are - W, F
application_type	396030	3	INDIVIDUAL	395319	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Indicates whether the loan is an individual application or a joint application with two co-borrowers

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	description
mort_acc	358235.0	NaN	NaN	NaN	1.813991	2.14793	0.0	0.0	1.0	3.0	34.0	Number of mortgage accounts
pub_rec_bankruptcies	395495.0	NaN	NaN	NaN	0.121648	0.356174	0.0	0.0	0.0	0.0	8.0	Number of public record bankruptcies
address	396030	393700	USCGC Smith\r\nFPO AE 70466	8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Address of the individua

In [8]:
```
df.sample(10)
```

Out[8]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issu
227730	10000.0	36 months	12.99	336.90	C	C1	Owner/Certified Appraiser	10+ years	MORTGAGE	32000.00	Source Verified	C 2
47296	9500.0	36 months	7.62	296.04	A	A3	Officer assistant manager	10+ years	RENT	50000.00	Not Verified	J 2
278429	14000.0	36 months	7.90	438.07	A	A4	INX International Ink Co.	10+ years	MORTGAGE	113318.00	Source Verified	S 2
41302	24000.0	60 months	13.98	558.19	C	C3	Product Manager	7 years	MORTGAGE	80400.00	Source Verified	2
241649	2700.0	36 months	9.91	87.01	B	B1	Allied Trade Group Inc.	2 years	RENT	45000.00	Not Verified	D 2
14575	8875.0	36 months	13.53	301.31	B	B5	registered dental assistant	5 years	RENT	28560.00	Source Verified	D 2
142603	12150.0	60 months	16.99	301.90	D	D3	NaN	NaN	RENT	75000.00	Verified	C 2
82770	20500.0	60 months	18.25	523.36	D	D3	Desktop Support Analyst	8 years	MORTGAGE	78000.00	Source Verified	J 2
341359	14000.0	36 months	12.99	471.65	C	C1	Healthy Aging Association	1 year	RENT	66000.00	Not Verified	2
183260	25000.0	60 months	22.40	696.18	E	E3	Chronic Care Program Manager	10+ years	MORTGAGE	90095.98	Source Verified	C 2

In [9]:
```
df.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                 394275 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
```

Target Value Analysis

In [10]:
```
np.round(100*df["loan_status"].value_counts()/df.shape[0],2)
```

Out[10]:
```
Fully Paid     80.39
Charged Off    19.61
Name: loan_status, dtype: float64
```

In [11]:
```
df["loan_status"].value_counts()
```

Out[11]:
```
Fully Paid     318357
Charged Off     77673
Name: loan_status, dtype: int64
```

In [12]:
```
df["loan_status"].replace("Fully Paid", 0, inplace=True)
df["loan_status"].replace("Charged Off", 1, inplace=True)
df["loan_status"].value_counts(normalize=True)*100
```

Out[12]:
```
0    80.387092
1    19.612908
Name: loan_status, dtype: float64
```

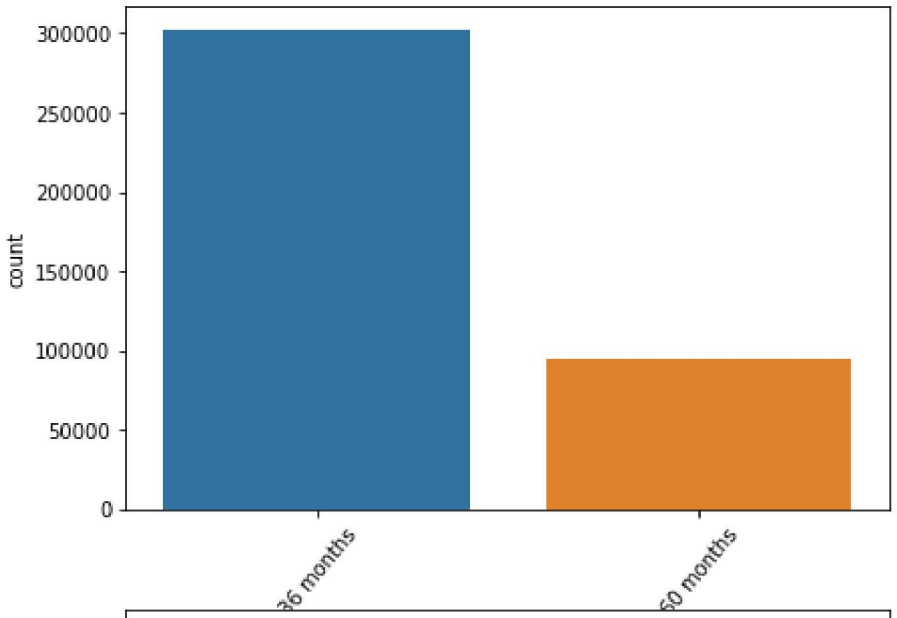
Segregating columns based on types

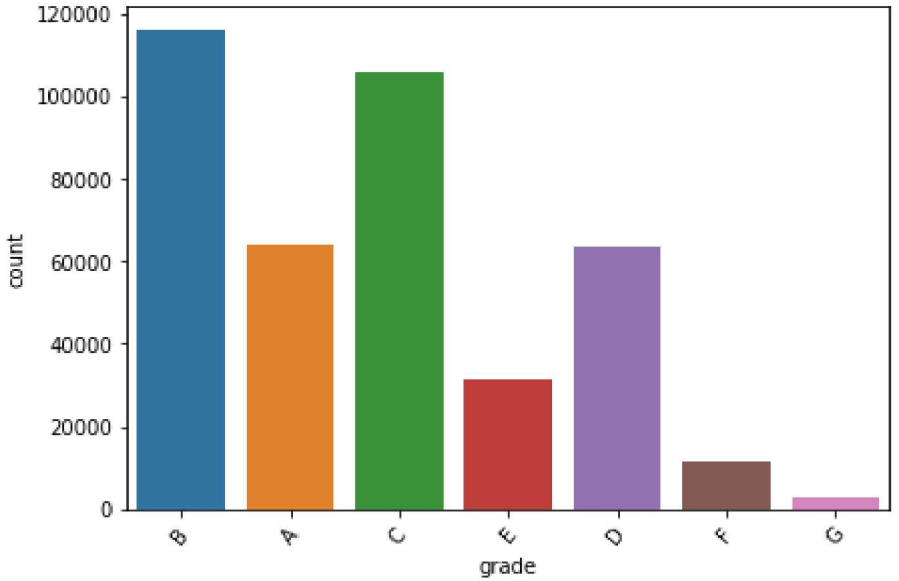
In [13]:
```
num_cols = [
    "loan_amnt"
    ,"int_rate"
    ,"installment"
    ,"annual_inc"
    ,"dti"
    ,"open_acc"
    ,"pub_rec"
    ,"revol_bal"
    ,"revol_util"
    ,"total_acc"
]

cat_cols = [
    "term"
    ,"grade"
    ,"sub_grade"
    ,"emp_length"
    ,"home_ownership"
    ,"verification_status"
    ,"purpose"
    ,"initial_list_status"
    ,"application_type"
```

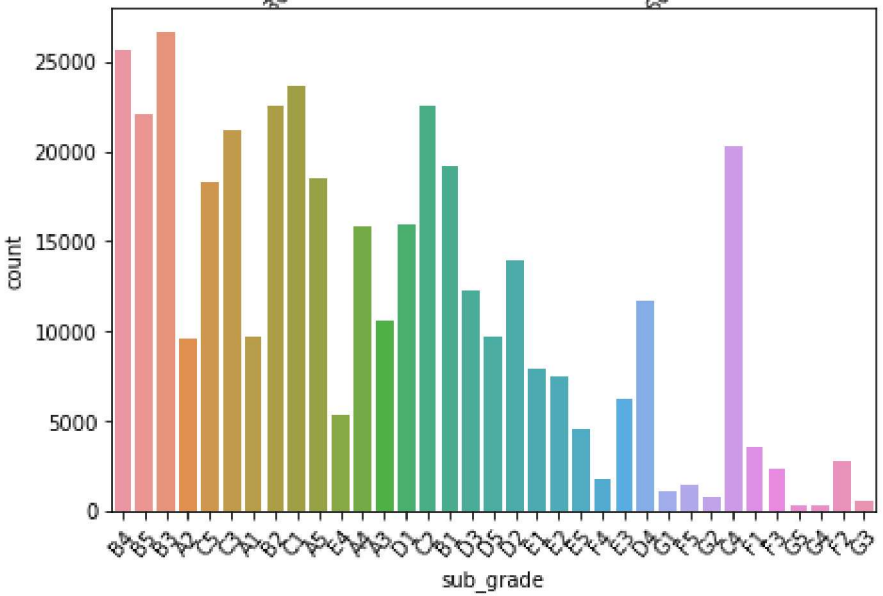
```
    ,"mort_acc"
    ,"pub_rec_bankruptcies"

]

datetime_cols = [
    "issue_d"
    ,"earliest_cr_line"

]

str_cols = [
    "emp_title"
    ,"title"
    ,"address"
]
```

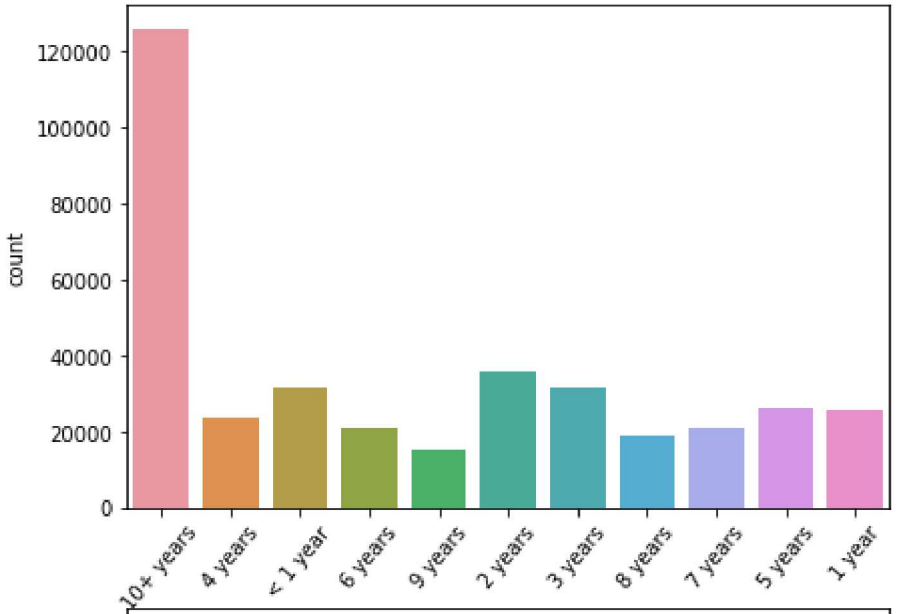
In [14]:

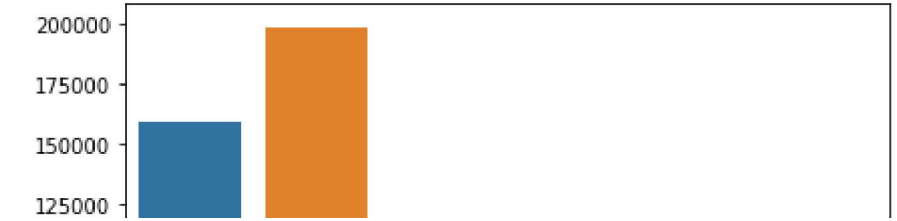
```
plt.figure(figsize=(15, 10*np.ceil(len(cat_cols)/2)))

for ind, i in enumerate(cat_cols):

    plt.subplot(len(cat_cols), 2, ind+1)
    sns.countplot(data=df, x=i)
    plt.xticks(rotation=50)


plt.show()
```


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

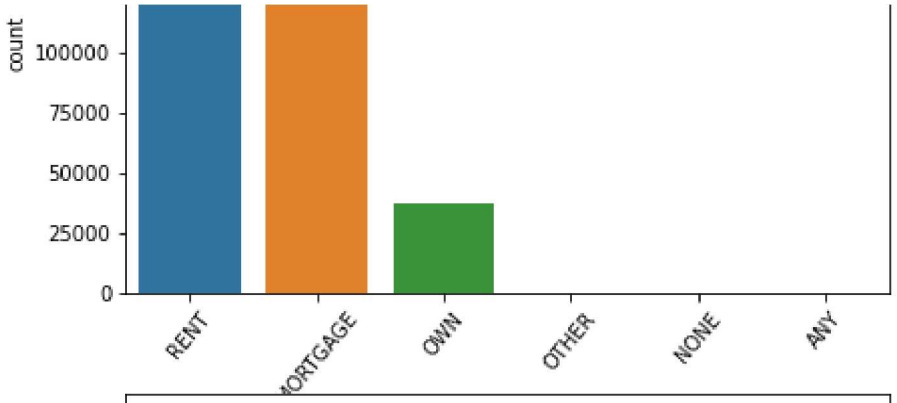

count
120000
100000
80000
60000
40000
20000
0
B
A
C
E
D
F
G
grade

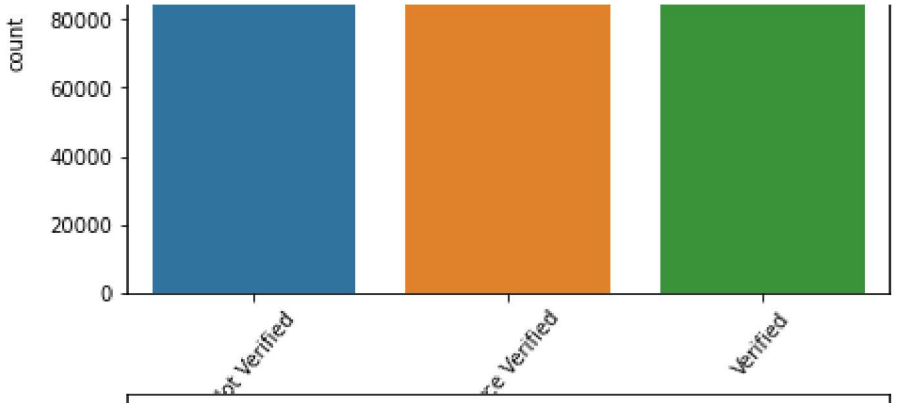

count
25000
20000
15000
10000
5000
0
B4 B5 B3 A2 C5 C3 A1 B2 C1 A5 E4 A4 A3 D1 C2 B1 D3 D5 D2 E1 E2 E5 F4 E3 D4 G1 F5 G2 C4 F1 F3 G5 G4 F2 G3
sub_grade

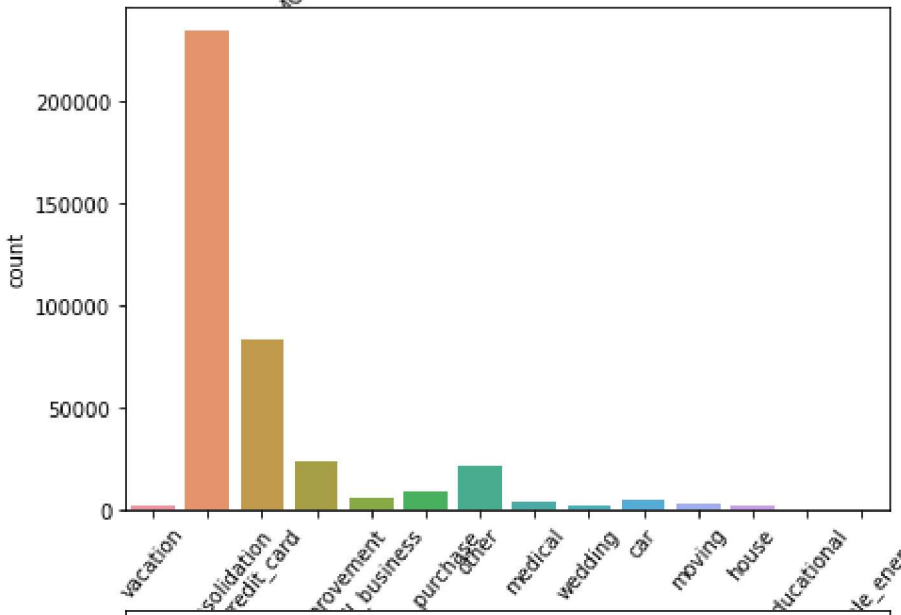

count
120000
100000
80000
60000
40000
20000
0
10+ years
4 years
< 1 year
6 years
9 years
2 years
3 years
8 years
7 years
5 years
1 year

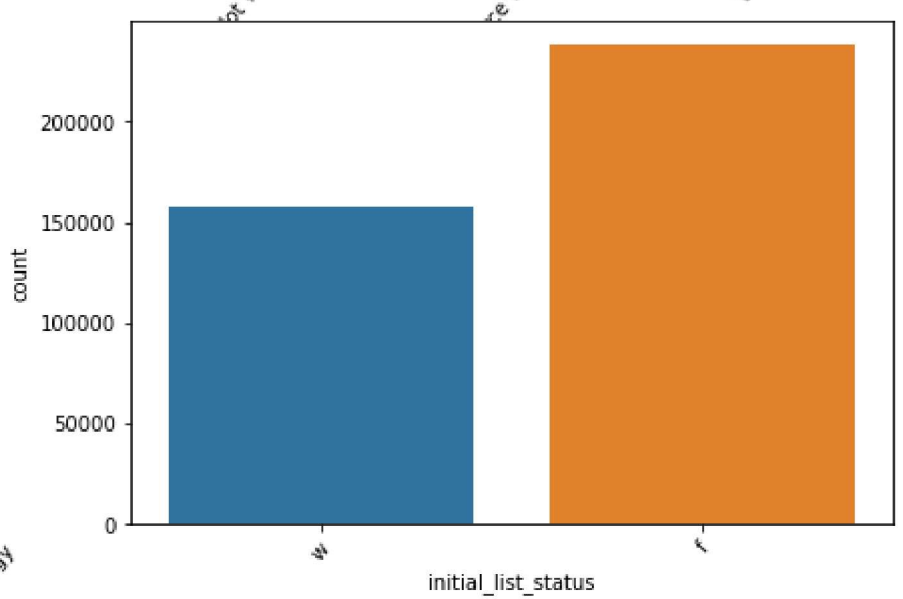

200000
175000
150000
125000

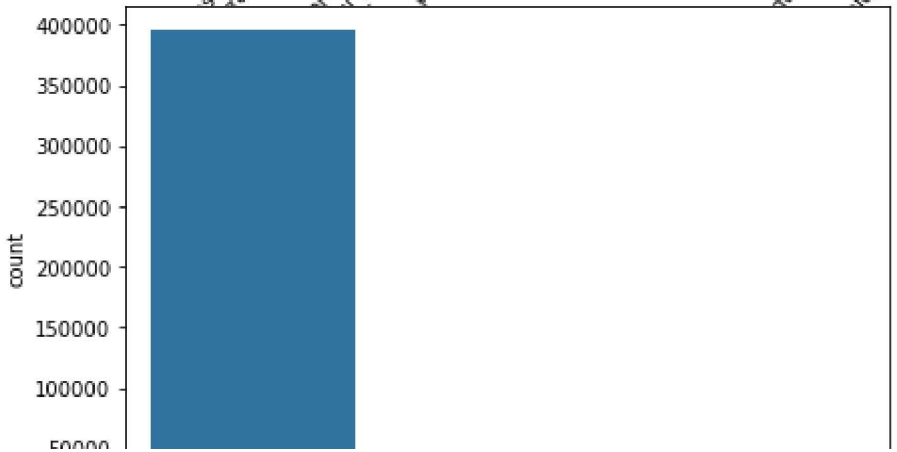

140000
120000
100000


count
100000
75000
50000
25000
0
RENT
MORTGAGE
OWN
OTHER
NONE
ANY

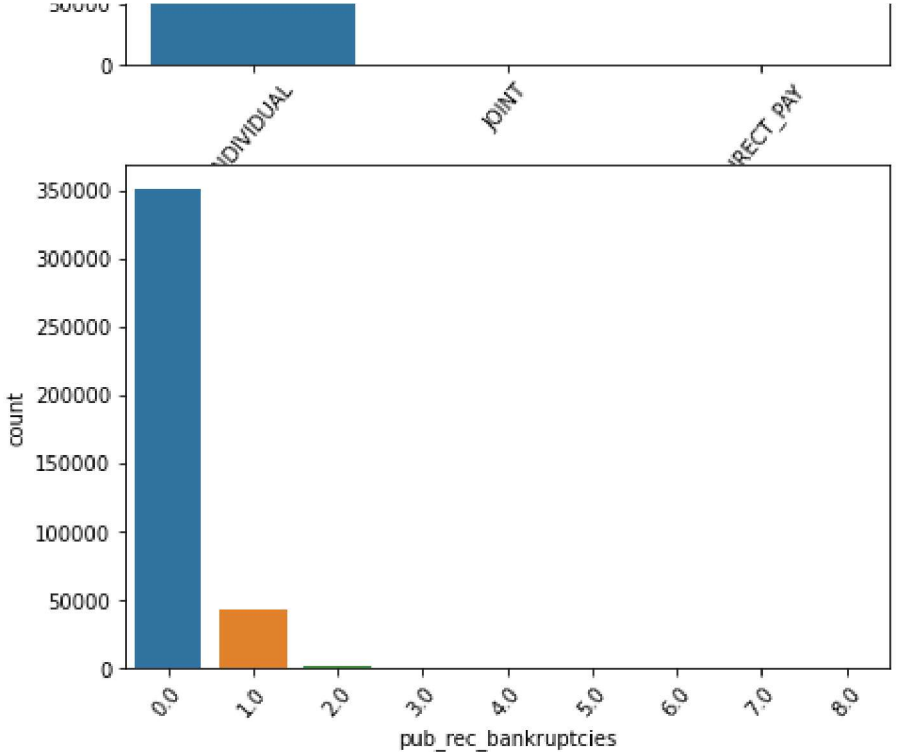

count
80000
60000
40000
20000
0
Verified

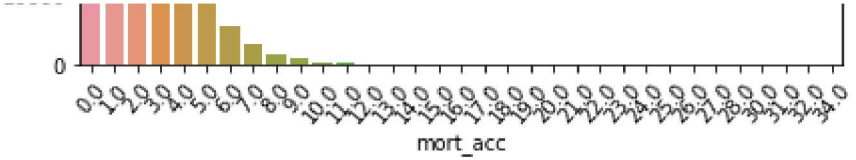

count
200000
150000
100000
50000
0
vacation
car
moving
house


count
200000
150000
100000
50000
0
w
f
initial_list_status


count
400000
350000
300000
250000
200000
150000
100000


count
140000
120000
100000
80000
60000
40000
20000





Observations:

- Outlier treatment required for:
  - mort_acc
  - pub_rec_bankruptcies

Outlier Treatment: Categorical features

In [15]:
```
df["mort_acc"].value_counts()
```

Out[15]:
```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
32.0         2
31.0         2
23.0         2
26.0         2
28.0         1
30.0         1
34.0         1
Name: mort_acc, dtype: int64
```

Let us cap the values greater than or equal to 10 to 10

In [16]:
```
for ind, row in df[["mort_acc"]].iterrows():
    if row["mort_acc"] >= 10:
        df.loc[ind, "mort_acc"] = 10.0
```

In [17]:
```
df["mort_acc"].value_counts()
```

Out[17]:
```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
10.0      2066
9.0       1656
Name: mort_acc, dtype: int64
```

In [18]:
```
df["pub_rec_bankruptcies"].value_counts()
```

Out[18]:
```
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

Let us cap the values greater than or equal to 2 to 2

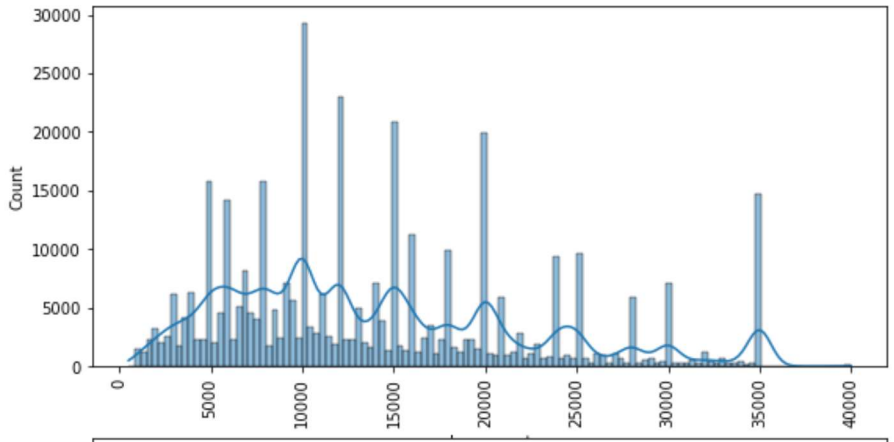
In [19]:
```
for ind, row in df[["pub_rec_bankruptcies"]].iterrows():
    if row["pub_rec_bankruptcies"] >= 2:
        df.loc[ind, "pub_rec_bankruptcies"] = 2
```

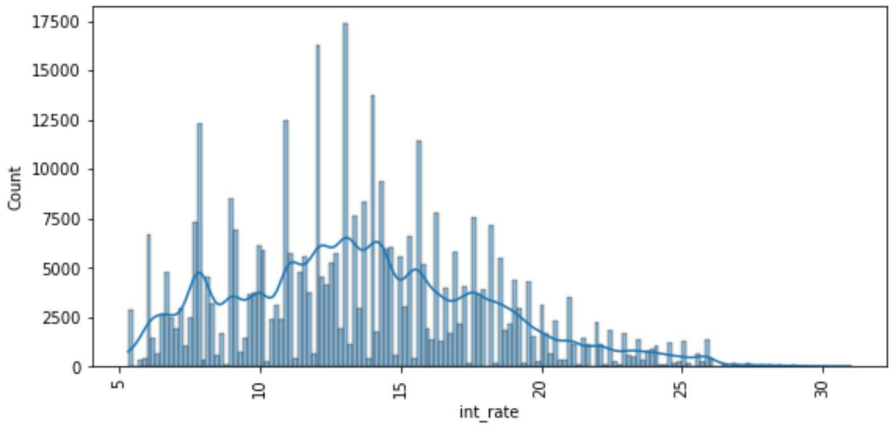
In [20]:
```
df["pub_rec_bankruptcies"].value_counts()
```

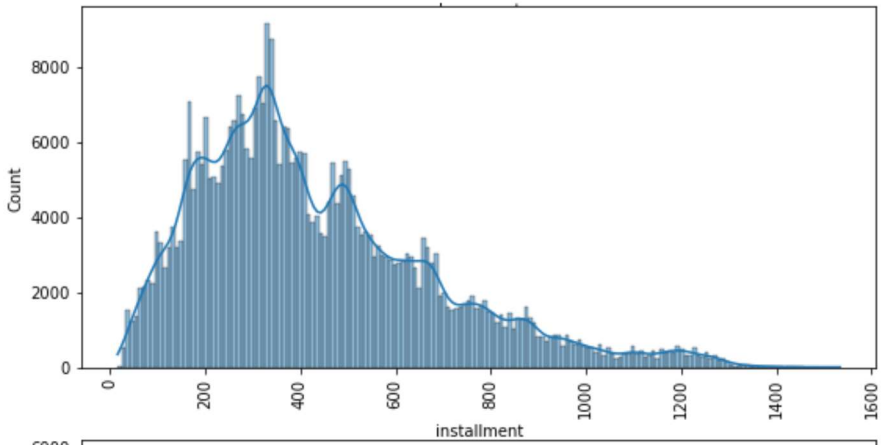
Out[20]:
```
0.0    350380
1.0     42790
2.0      2325
Name: pub_rec_bankruptcies, dtype: int64
```

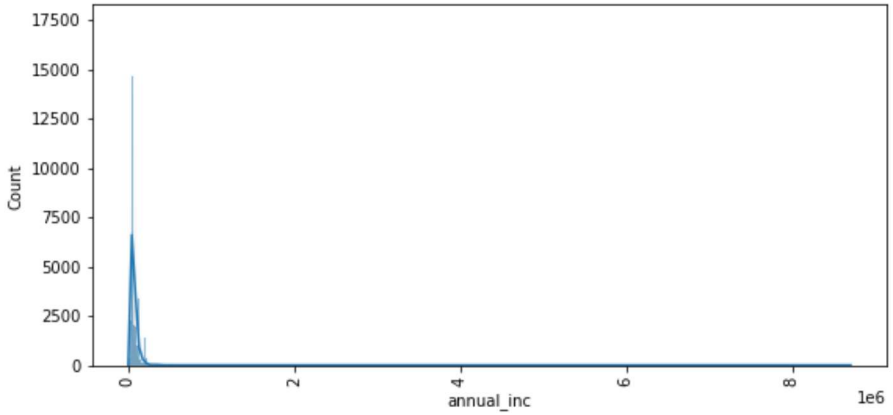
In [21]:
```
plt.figure(figsize=(20, 10*np.ceil(len(num_cols)/2)))

for ind, i in enumerate(num_cols):
```

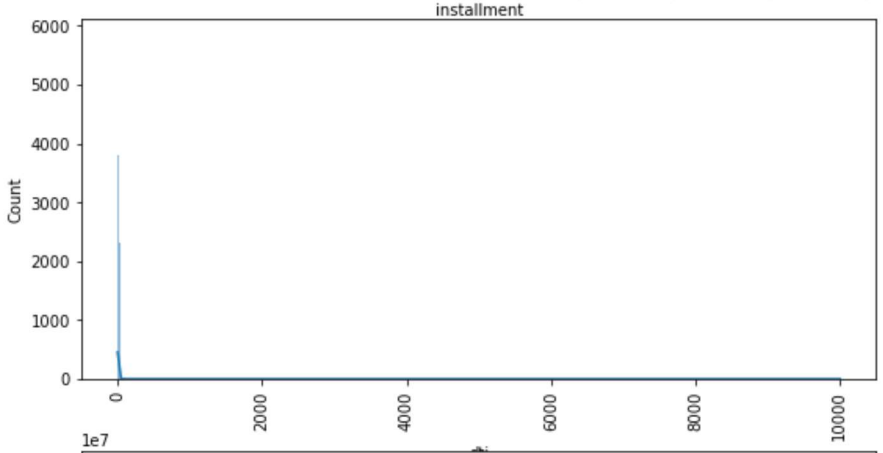
```
    plt.subplot(len(num_cols), 2, ind+1)
    sns.histplot(data=df, x=i, kde=True)
    plt.xticks(rotation=90)


plt.show()
```

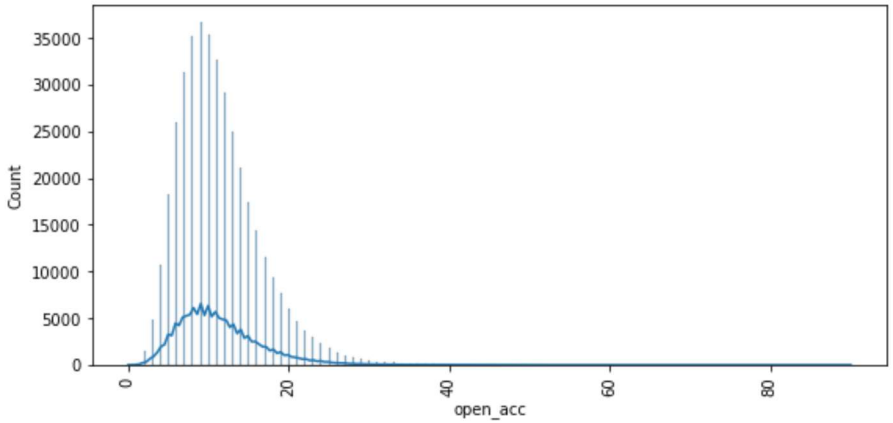

Count
0
5000
10000
15000
20000
25000
30000
0
5000
10000
15000
20000
25000
30000
35000
40000


Count
0
2500
5000
7500
10000
12500
15000
17500
5
10
15
20
25
30
int_rate


Count
0
2000
4000
6000
8000
0
200
400
600
800
1000
1200
1400
1600
installment

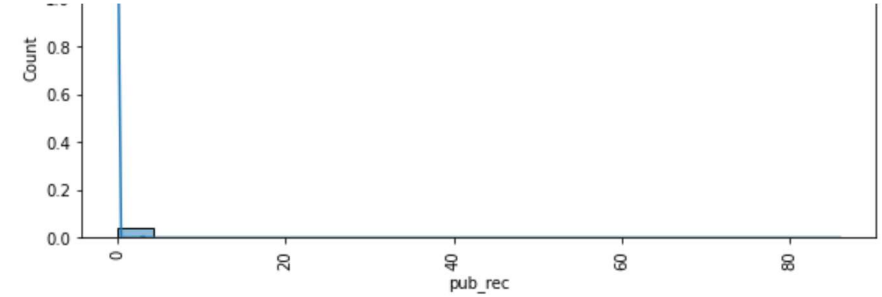

Count
0
2500
5000
7500
10000
12500
15000
17500
0
2
4
6
8
1e6
annual_inc

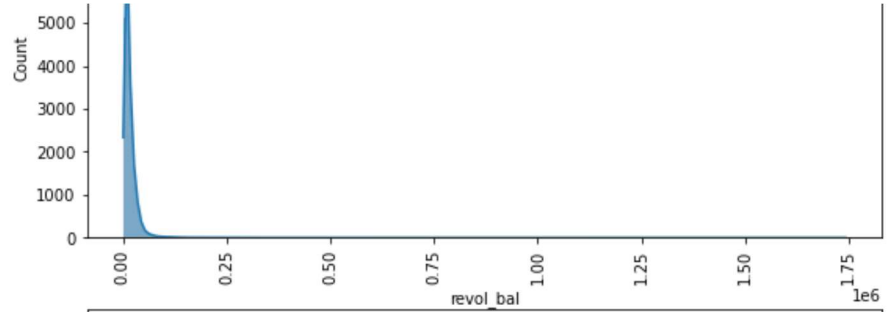

Count
0
1000
2000
3000
4000
5000
6000
0
2000
4000
6000
8000
10000

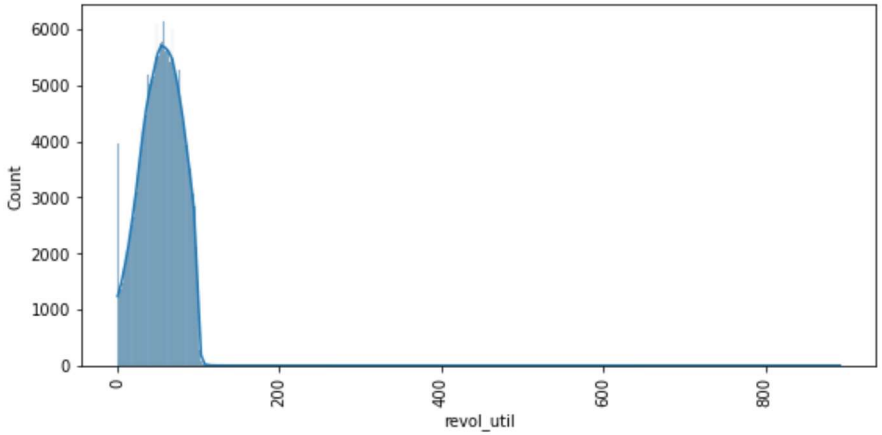

Count
0
5000
10000
15000
20000
25000
30000
35000
0
20
40
60
80
open_acc

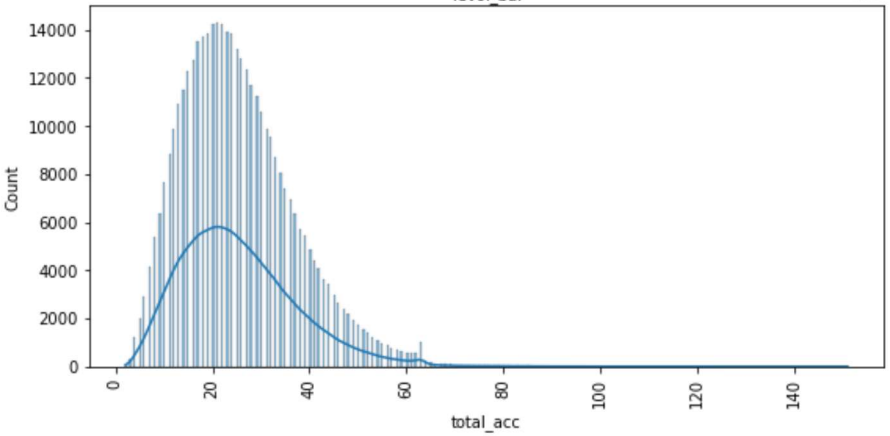

1e7
1.0
1.2
1.4


6000
7000
8000









Observations

- Outlier treatement required for the below features:
  - annual_inc
  - dti
  - open_acc
  - pub_rec
  - revol_bal
  - revol_util
  - total_acc

Outlier Treatment: Categorical features

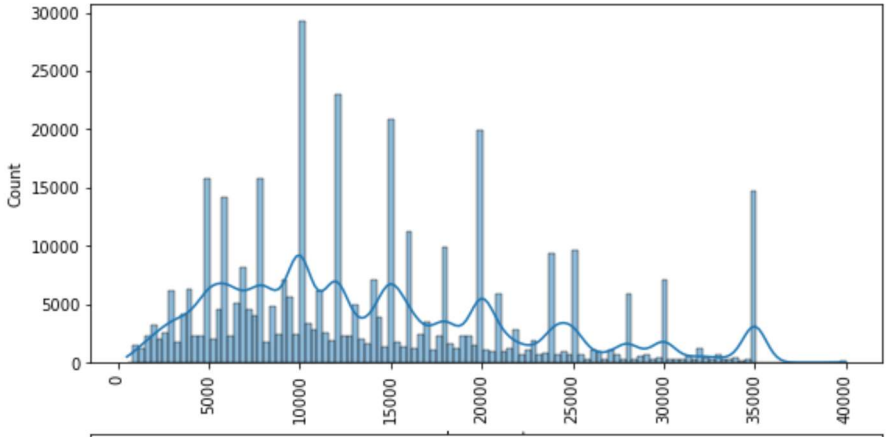
In [22]:
```
def iqr_treatment(df, feature, strategy="median"):
    data = sorted(df[feature])
```

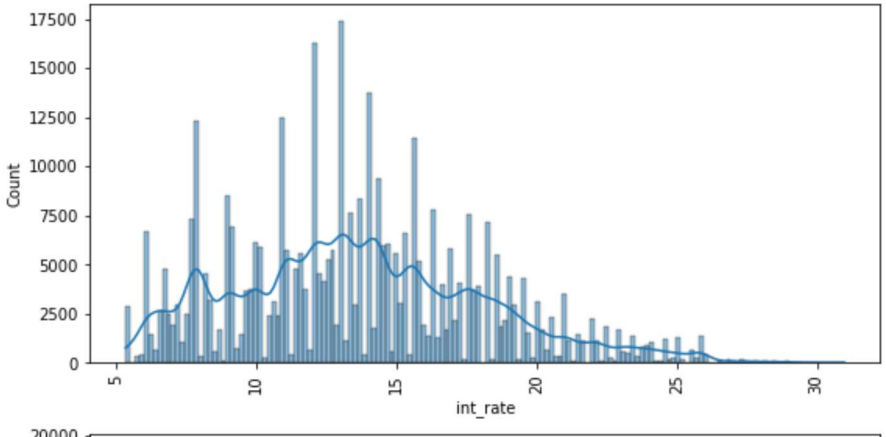
```
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)

    median = np.percentile(data, 50)

    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)

    for ind, row in df[[feature]].iterrows():
        if (row[feature] < lwr_bound or row[feature] > upr_bound):
            df.loc[ind, feature] = median
```

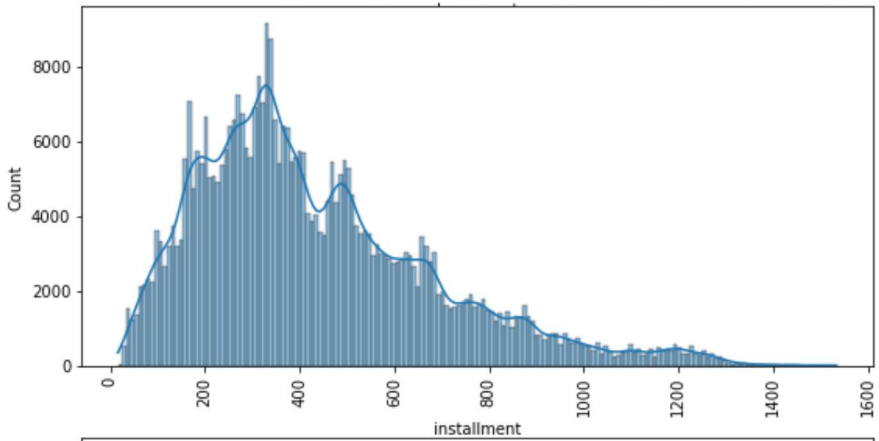
In [23]:
```
cols_for_outlier_treatment= ["annual_inc" ,"dti" ,"open_acc" ,"pub_rec" ,"revol_bal" ,"revol_util" ,"total_acc"]
```

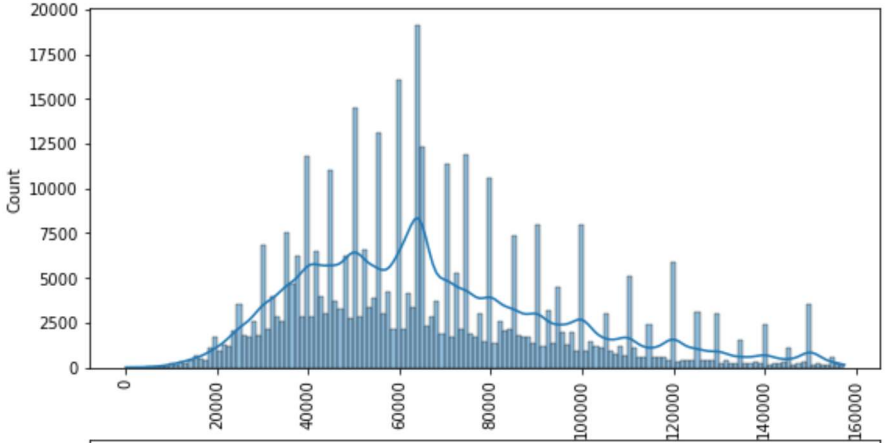
In [24]:
```
for i in cols_for_outlier_treatment:
    iqr_treatment(df, i)
```

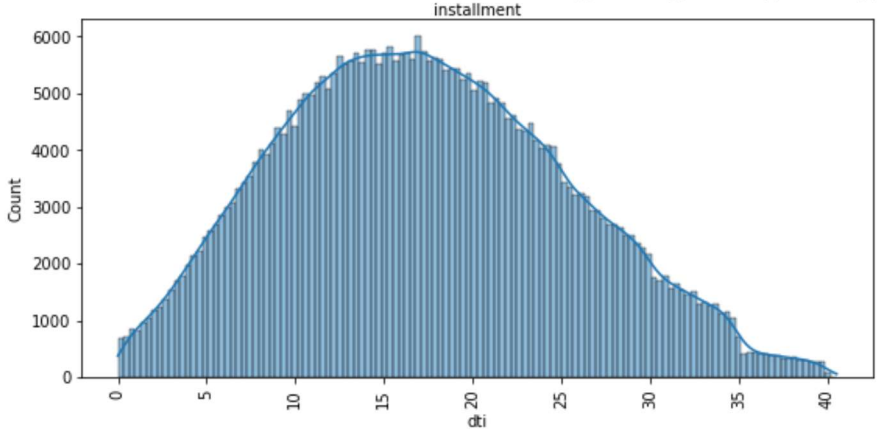
In [25]:
```
plt.figure(figsize=(20, 10*np.ceil(len(num_cols)/2)))

for ind, i in enumerate(num_cols):

    plt.subplot(len(num_cols), 2, ind+1)
    sns.histplot(data=df, x=i, kde=True)
    plt.xticks(rotation=90)


plt.show()
```

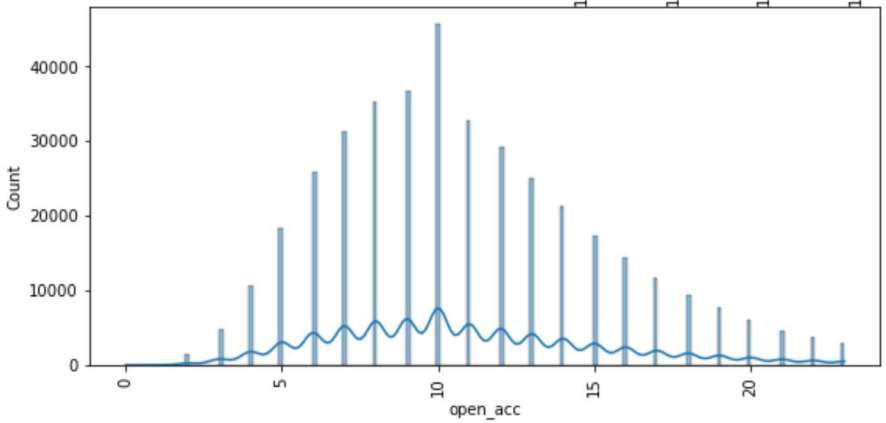

Count
0
5000
10000
15000
20000
25000
30000
35000
40000


Count
int_rate
0
2500
5000
7500
10000
12500
15000
17500
5
10
15
20
25
30


Count
installment
0
2000
4000
6000
8000
0
200
400
600
800
1000
1200
1400
1600

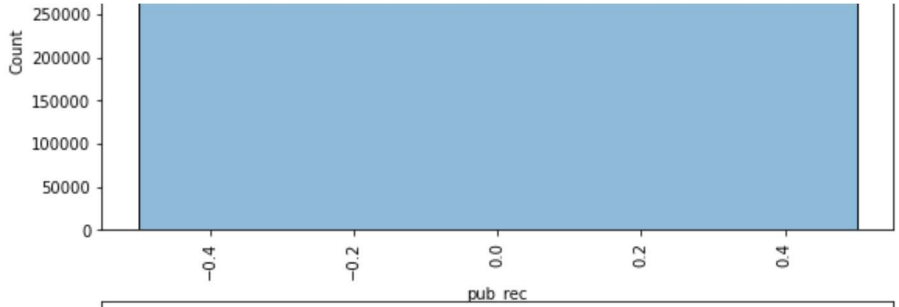

Count
0
2500
5000
7500
10000
12500
15000
17500
20000
0
20000
40000
60000
80000
100000
120000
140000
160000

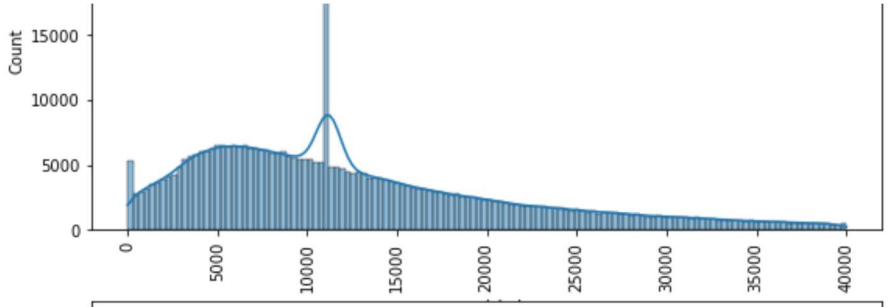

Count
dti
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25
30
35
40

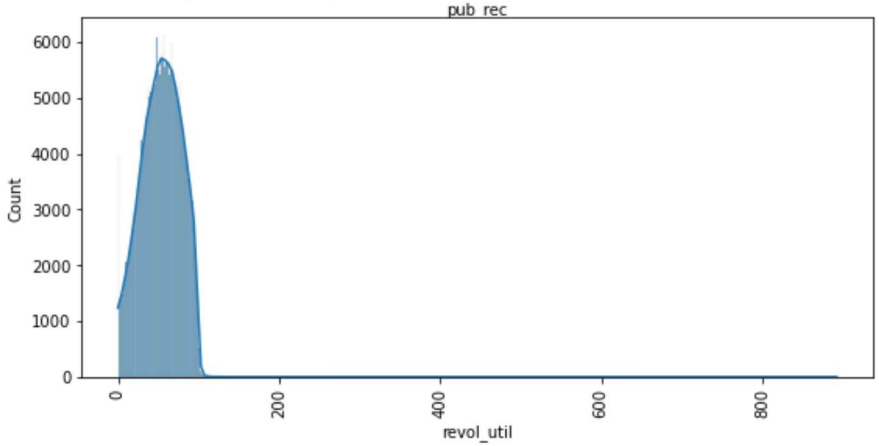

Count
open_acc
0
10000
20000
30000
40000
0
5
10
15
20

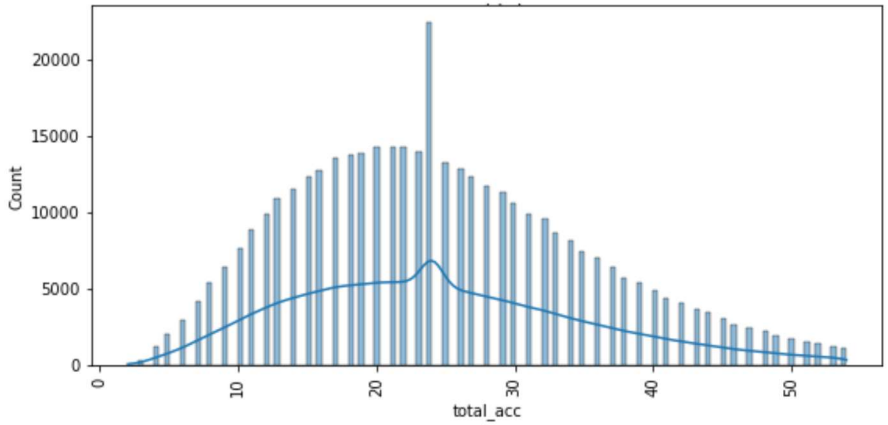

300000
350000
400000


20000
25000









Missing value treatment

In [26]:
```
missing_df = pd.DataFrame(np.round(df.isnull().sum()).loc[lambda x: x > 0])
missing_df = pd.concat([missing_df, np.round(100*df.isnull().sum()/df.shape[0], 2).loc[lambda x: x > 0]], axis=1)
missing_df.columns = ["no_of_rows", "percentage"]
missing_df.sort_values("percentage", ascending=False)
```

Out[26]:

	no_of_rows	percentage
mort_acc	37795	9.54
emp_title	22927	5.79
emp_length	18301	4.62
title	1755	0.44
pub_rec_bankruptcies	535	0.14
revol_util	276	0.07

In [27]:
```
df["mort_acc"].value_counts(dropna=False)
```

Out[27]:
```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
NaN      37795
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
10.0      2066
9.0       1656
Name: mort_acc, dtype: int64
```

Can be imputed with mode

In [28]:
```
mort_acc_mode = df["mort_acc"].value_counts().index[0]
mort_acc_mode
```

Out[28]:
```
0.0
```

In [29]:
```
df["mort_acc"].fillna(mort_acc_mode, inplace=True)
```

In [30]:
```
df["emp_title"].value_counts(dropna=False)
```

Out[30]:
```
NaN                         22927
Teacher                      4389
Manager                      4250
Registered Nurse             1856
RN                           1846
                            ...
Postman                         1
McCarthy & Holthus, LLC         1
jp flooring                     1
Histology Technologist          1
Gracon Services, Inc            1
Name: emp_title, Length: 173106, dtype: int64
```

Huge variety of job titles and a significant amount of them are not available

Cannot be imputed using mode

Can create a new category for nulls

In [31]:
```
df["emp_title"].fillna("NA", inplace=True)
```

In [32]:
```
100*df["emp_length"].value_counts(dropna=False)/df.shape[0]
```

Out[32]:
```
10+ years    31.826124
2 years       9.046537
< 1 year      8.010757
3 years       7.995606
5 years       6.690150
1 year        6.535363
4 years       6.048027
6 years       5.262480
7 years       5.256925
8 years       4.840037
NaN           4.621115
9 years       3.866879
Name: emp_length, dtype: float64
```

Can impute using mode

In [33]:
```
emp_length_mode = df["emp_length"].value_counts().index[0]
emp_length_mode
```

Out[33]:
```
'10+ years'
```

In [34]:
```
df["emp_length"].fillna(emp_length_mode, inplace=True)
```

In [35]:
```
df["title"].value_counts(dropna=False)
```

Out[35]:
```
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: title, Length: 48818, dtype: int64
```

Can be added to the category "Other"

In [36]:
```
df["title"].fillna("Other", inplace=True)
```

In [37]:
```
df["pub_rec_bankruptcies"].value_counts(dropna=False)
```

Out[37]:
```
0.0    350380
1.0     42790
2.0      2325
NaN       535
Name: pub_rec_bankruptcies, dtype: int64
```

Can be imputed with mode

In [38]:
```
df["pub_rec_bankruptcies"].mode()[0]
```

Out[38]:
```
0.0
```

In [39]:
```
df["pub_rec_bankruptcies"].fillna(df["pub_rec_bankruptcies"].mode()[0], inplace=True)
```

In [40]:
```
df["revol_util"].value_counts(dropna=False)
```

Out[40]:
```
0.00      2213
53.00      752
60.00      739
61.00      734
55.00      730
           ...
892.30       1
110.10       1
123.00       1
49.63        1
128.10       1
Name: revol_util, Length: 1227, dtype: int64
```

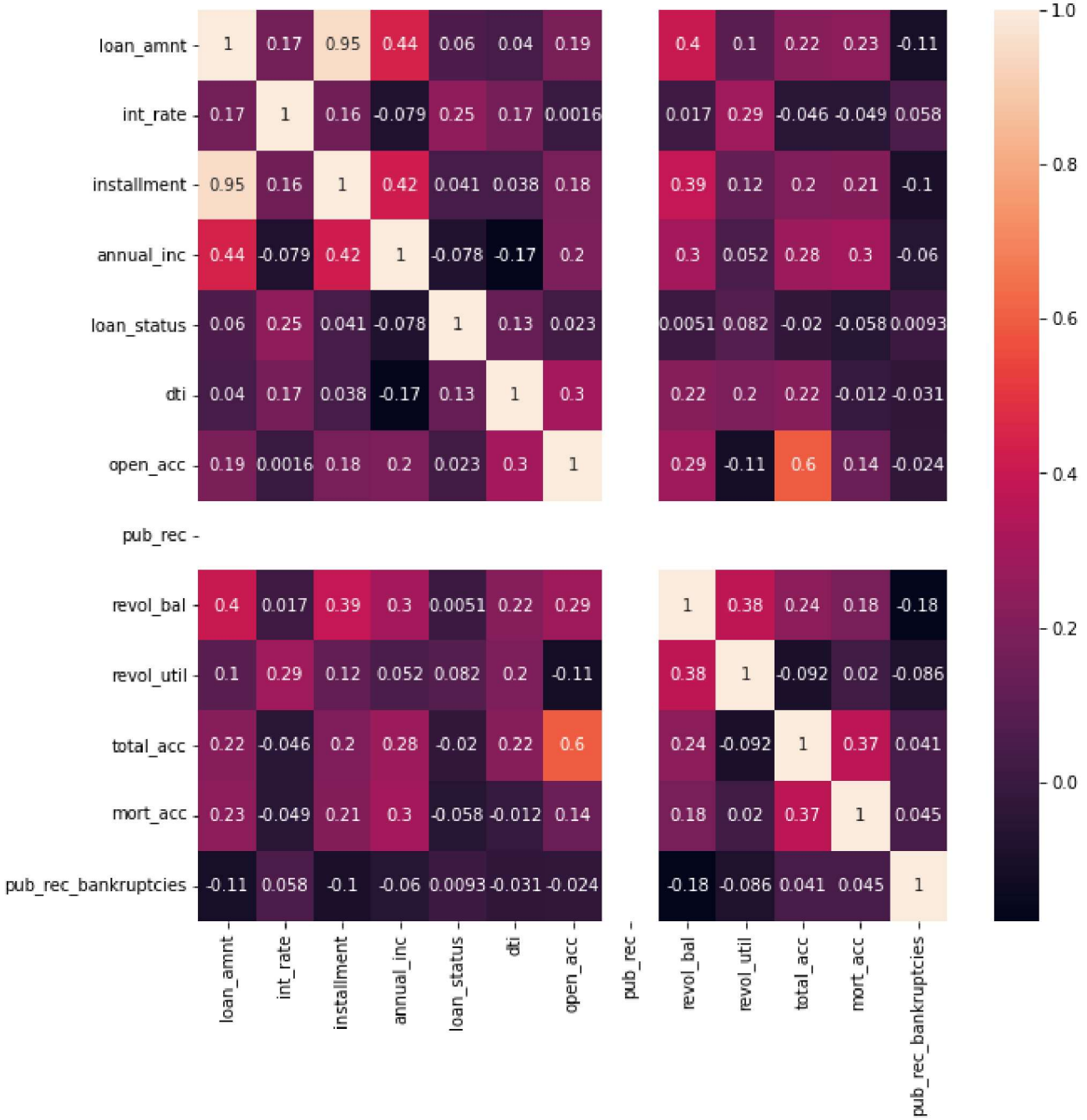
Can be imputed using mode

In [41]:
```
df["revol_util"].fillna(df["revol_util"].mode()[0], inplace=True)
```

In [42]:
```
df.isnull().sum()
```

Out[42]:
```
loan_amnt               0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
title                   0
dti                     0
earliest_cr_line        0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
initial_list_status     0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
address                 0
dtype: int64
```

In [98]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```


loan_amnt 1 0.17 0.95 0.44 0.06 0.04 0.19 0.4 0.1 0.22 0.23 -0.11
int_rate 0.17 1 0.16 -0.079 0.25 0.17 0.0016 0.017 0.29 -0.046 -0.049 0.058
installment 0.95 0.16 1 0.42 0.041 0.038 0.18 0.39 0.12 0.2 0.21 -0.1
annual_inc 0.44 -0.079 0.42 1 -0.078 -0.17 0.2 0.3 0.052 0.28 0.3 -0.06
loan_status 0.06 0.25 0.041 -0.078 1 0.13 0.023 0.0051 0.082 -0.02 -0.058 0.0093
dti 0.04 0.17 0.038 -0.17 0.13 1 0.3 0.22 0.2 0.22 -0.012 -0.031
open_acc 0.19 0.0016 0.18 0.2 0.023 0.3 1 0.29 -0.11 0.6 0.14 -0.024
pub_rec
revol_bal 0.4 0.017 0.39 0.3 0.0051 0.22 0.29 1 0.38 0.24 0.18 -0.18
revol_util 0.1 0.29 0.12 0.052 0.082 0.2 -0.11 0.38 1 -0.092 0.02 -0.086
total_acc 0.22 -0.046 0.2 0.28 -0.02 0.22 0.6 0.24 -0.092 1 0.37 0.041
mort_acc 0.23 -0.049 0.21 0.3 -0.058 -0.012 0.14 0.18 0.02 0.37 1 0.045
pub_rec_bankruptcies -0.11 0.058 -0.1 -0.06 0.0093 -0.031 -0.024 -0.18 -0.086 0.041 0.045 1
loan_amnt
int_rate
installment
annual_inc
loan_status
dti
open_acc
pub_rec
revol_bal
revol_util
total_acc
mort_acc
pub_rec_bankruptcies
1.0
0.8
0.6
0.4
0.2
0.0

Feature engineering

Pipeline for transformation

In [43]:
```
class DataframeSelector(BaseEstimator, TransformerMixin):
    def __init__(self, att_list):
        self.att_list = att_list

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X[self.att_list]
```

In [44]:
```
# Generic tranformation pipelines for the different kind of columns

num_pipeline = Pipeline([
    ("selector", DataframeSelector(num_cols)),
    ("std_scaler", StandardScaler())
])

cat_pipeline = Pipeline([
    ("selector", DataframeSelector(cat_cols)),
    ('one_hot_encoder', OneHotEncoder(sparse=False))
])

full_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, num_cols),
    ("cat_pipeline", cat_pipeline, cat_cols)
], remainder='passthrough')
```

In [45]:
```
# Functions to retrieve feature names after pipeline

def get_feature_out(estimator, feature_in):
    if hasattr(estimator,'get_feature_names'):
        if isinstance(estimator, _VectorizerMixin):
            # handling all vectorizers
            return [f'vec_{f}' \
                for f in estimator.get_feature_names_out()]
        else:
            return estimator.get_feature_names_out(feature_in)
```

```
    elif isinstance(estimator, SelectorMixin):
        return np.array(feature_in)[estimator.get_support()]
    else:
        return feature_in


def get_ct_feature_names(ct):
    # handles all estimators, pipelines inside ColumnTransfomer
    # doesn't work when remainder =='passthrough'
    # which requires the input column names.
    output_features = []

    for name, estimator, features in ct.transformers_:
        if name!='remainder':
            if isinstance(estimator, Pipeline):
                current_features = features
                for step in estimator:
                    current_features = get_feature_out(step, current_features)
                features_out = current_features
            else:
                features_out = get_feature_out(estimator, features)
            output_features.extend(features_out)
        elif estimator=='passthrough':
            output_features.extend(ct.feature_names_in_[features])

    return output_features
```

In [46]:
```
input_df = df.drop(str_cols, axis=1)
input_df.drop(datetime_cols, axis=1, inplace=True)

transformed_data = full_pipeline.fit_transform(input_df)
```

In [47]:
```
tx_data_col_names = get_ct_feature_names(full_pipeline)
```

In [48]:
```
transformed_data_df = pd.DataFrame(transformed_data, columns=tx_data_col_names)
```

In [49]:
```
corr = transformed_data_df.corr()["loan_status"]
```

In [50]:
```
corr.loc[lambda x: (x > 0.1) | (x < -0.1)].sort_values()
```

Out[50]:
```
term_ 36 months   -0.173246
grade_A           -0.147591
grade_B           -0.114123
grade_D            0.101877
grade_F            0.102158
grade_E            0.131385
dti                0.132177
term_ 60 months    0.173246
int_rate           0.247758
loan_status        1.000000
Name: loan_status, dtype: float64
```

Train-test split

In [51]:
```
X = transformed_data_df.drop("loan_status", axis=1)
y = transformed_data_df["loan_status"].copy()

# Stratified sampling since the data is imbalanced
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Model building

In [52]:
```
log_reg = LogisticRegression(max_iter=400)
log_reg.fit(X_train, y_train)
```

Out[52]:
```
LogisticRegression(max_iter=400)
```

In [53]:
```
y_pred = log_reg.predict(X_test)
```

In [54]:
```
print("model score: %.3f" % log_reg.score(X_test, y_test))
```

```
model score: 0.806
```

In [55]:
```
log_reg.coef_, log_reg.intercept_
```

Out[55]:
```
(array([[-0.12806345, -0.39769521,  0.21148415, -0.18635754,  0.19458476,
          0.09588252,  0.        , -0.05277502,  0.0767783 , -0.06954634,
         -0.43282294,  0.11171694, -1.83291913, -0.98578642, -0.33743255,
          0.14989208,  0.54994301,  0.92826533,  1.20693169, -1.01991188,
         -0.5041775 , -0.31132113, -0.12859617,  0.13108755, -0.52476262,
         -0.3667707 , -0.21311845, -0.03008415,  0.14894951, -0.34648188,
         -0.19216162, -0.04042566,  0.0692336 ,  0.17240301, -0.15702559,
         -0.0511697 , -0.00800307,  0.14986195,  0.21622848, -0.10535644,
          0.04568259,  0.10752486,  0.19396533,  0.30812666, -0.11555593,
          0.10091167,  0.17192565,  0.31045384,  0.4605301 ,  0.18692094,
          0.25996746,  0.35888228,  0.09718167,  0.30397933, -0.0366452 ,
          0.01061069, -0.05373268, -0.06617332, -0.07479442, -0.04512614,
         -0.03268104, -0.030805  , -0.00227484,  0.00733119,  0.00318476,
         -0.03039695, -0.1665066 , -0.01028417, -0.18947164, -0.02706251,
          0.10261587, -0.18566445, -0.04007705, -0.0953645 , -0.15148897,
         -0.06029233, -0.02995322, -0.04627922,  0.029527  , -0.12615572,
         -0.02584983,  0.08423325,  0.0571167 , -0.02060877,  0.16217764,
          0.40596741, -0.03607993, -0.56342   , -0.15921414, -0.16189186,
          0.30553736,  0.15581015, -0.78245351,  0.07750456,  0.12238458,
          0.08238736,  0.01047828,  0.00348511, -0.01392843,  0.0120998 ,
         -0.06408095, -0.11603991, -0.23949947, -0.19589693, -0.12587373,
         -0.1392584 , -0.05597387]]),
 array([-0.35942973]))
```

In [57]:
```
coef_df = pd.DataFrame(pd.Series(np.array(log_reg.coef_[0]), index=tx_data_col_names[:-1]).sort_values(), columns=["coeff"])
```

In [58]:
```
coef_df.head(10)
```

Out[58]:

	coeff
grade_A	-1.832919
sub_grade_A1	-1.019912
grade_B	-0.985786
application_type_JOINT	-0.782454
purpose_wedding	-0.563420
sub_grade_B1	-0.524763
sub_grade_A2	-0.504177
term_ 36 months	-0.432823
int_rate	-0.397695
sub_grade_B2	-0.366771

In [59]:
```
coef_df.tail(10)
```

Out[59]:

	coeff
sub_grade_G5	0.303979
application_type_DIRECT_PAY	0.305537
sub_grade_E5	0.308127
sub_grade_F4	0.310454
sub_grade_G3	0.358882
purpose_small_business	0.405967
sub_grade_F5	0.460530
grade_E	0.549943
grade_F	0.928265
grade_G	1.206932

Observation

We can notice that that grades and subgrades are very essential in deciding the probability of a defaulter.
These parameters significantly indicate that individual is going to default:

- Grade: E/F/G
- Application type: DIRECT_PAY
- Purpose: Small business

These parameters significantly indicate that individual is NOT going to default:

- Grade: A/B
- Application type: JOINT
- Purpose: Wedding
- Home Ownership: NONE
- Interest rate
- term: 36 months

Recommendations:

- The parameters that indicate that an individual is going to default should be exercised with caution
- For the parameters that indicate that the individual won't default, loans can be disbursed to them
- Grade is probably the most important feature to consider while disbursing the loan

Model Evaluation

In [60]:
```
y_actual = pd.Series(y_test, name='Actual')
y_predicted = pd.Series(y_pred, name='Predicted')
```

In [62]:
```
pd.crosstab(y_actual, list(y_predicted), margins=True).transpose()
```

Out[62]:

Actual	0.0	1.0	All
col_0			
0.0	62871	14565	77436
1.0	800	970	1770
All	63671	15535	79206

In [63]:
```
print(classification_report(y_actual, y_predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.99      0.89     63671
         1.0       0.55      0.06      0.11     15535

    accuracy                           0.81     79206
   macro avg       0.68      0.52      0.50     79206
weighted avg       0.76      0.81      0.74     79206
```

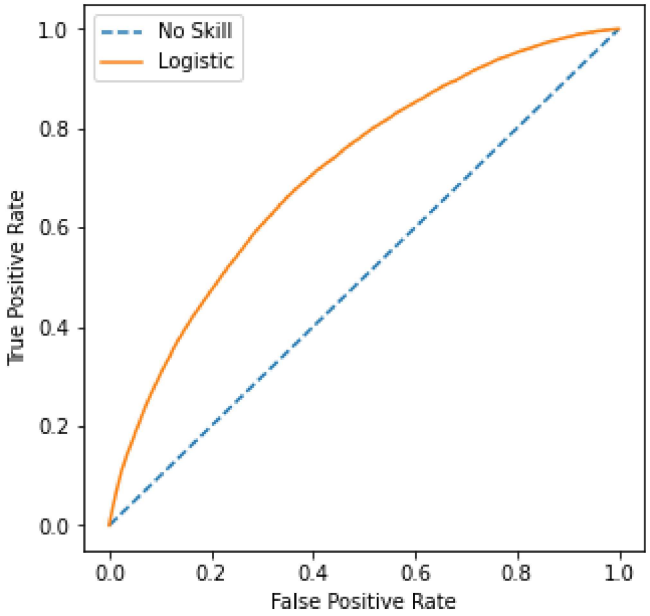
Observation

- Precision is okay but recall and f1 scores are pretty low for the defaulters.

In [64]:
```
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = log_reg.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.712
```

In [65]:
```
plt.figure(figsize=(5,5))

# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



Observation

- ROC score is good, could be better with more feature engg

SMOTE

Since the data is imabalanced, we would apply SMOTE for oversampling and also to reduce false positives and increase model performance.

In [66]:
```
X_resampled, y_resampled = SMOTE().fit_resample(X_train, y_train)
```

In [67]:
```
y_resampled.value_counts()
```

Out[67]:
```
0.0    254686
1.0    254686
Name: loan_status, dtype: int64
```

In [68]:
```
log_reg_2 = LogisticRegression(max_iter=400)

# Fit the model with the data that has been resampled with SMOTE
log_reg_2.fit(X_resampled, y_resampled)
```

Out[68]:
```
LogisticRegression(max_iter=400)
```

In [69]:
```
# Predict on the test set (not resampled to obtain honest evaluation)
y_pred_2 = log_reg_2.predict(X_test)
```

In [70]:
```
print(classification_report(y_actual, y_predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.99      0.89     63671
         1.0       0.55      0.06      0.11     15535

    accuracy                           0.81     79206
   macro avg       0.68      0.52      0.50     79206
weighted avg       0.76      0.81      0.74     79206
```

In [71]:
```
print(classification_report(y_actual, y_pred_2))
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.64      0.75     63671
         1.0       0.31      0.66      0.42     15535

    accuracy                           0.65     79206
   macro avg       0.60      0.65      0.58     79206
weighted avg       0.77      0.65      0.68     79206
```

In [90]:
```
pd.crosstab(y_actual, list(y_predicted), margins=True).transpose()
```

Out[90]:

Actual	0.0	1.0	All
col_0			
0.0	62871	14565	77436
1.0	800	970	1770
All	63671	15535	79206

In [73]:
```
y_predicted_2 = pd.Series(y_pred_2, name='Predicted_SMOTE')
pd.crosstab(y_actual, y_predicted_2, margins=True).transpose()
```

Out[73]:

Actual	0.0	1.0	All
Predicted_SMOTE			
0.0	7504	1819	9323
1.0	5281	1267	6548
All	12785	3086	15871

Observation

- Recall and f1 score has increased a lot but the precision has decreased. We might to consider our model based on the business requirement.
- From the SMOTE confusion matrix, we can clearly see that the TP has increased a lot and the FN has also decreased

Recommendation

- If it is important that the defaulters be predicted with better accuracy we should pick up the model implemented using SMOTE
- If it is important that the loan be disbursed to potental customers, we can go with the first model

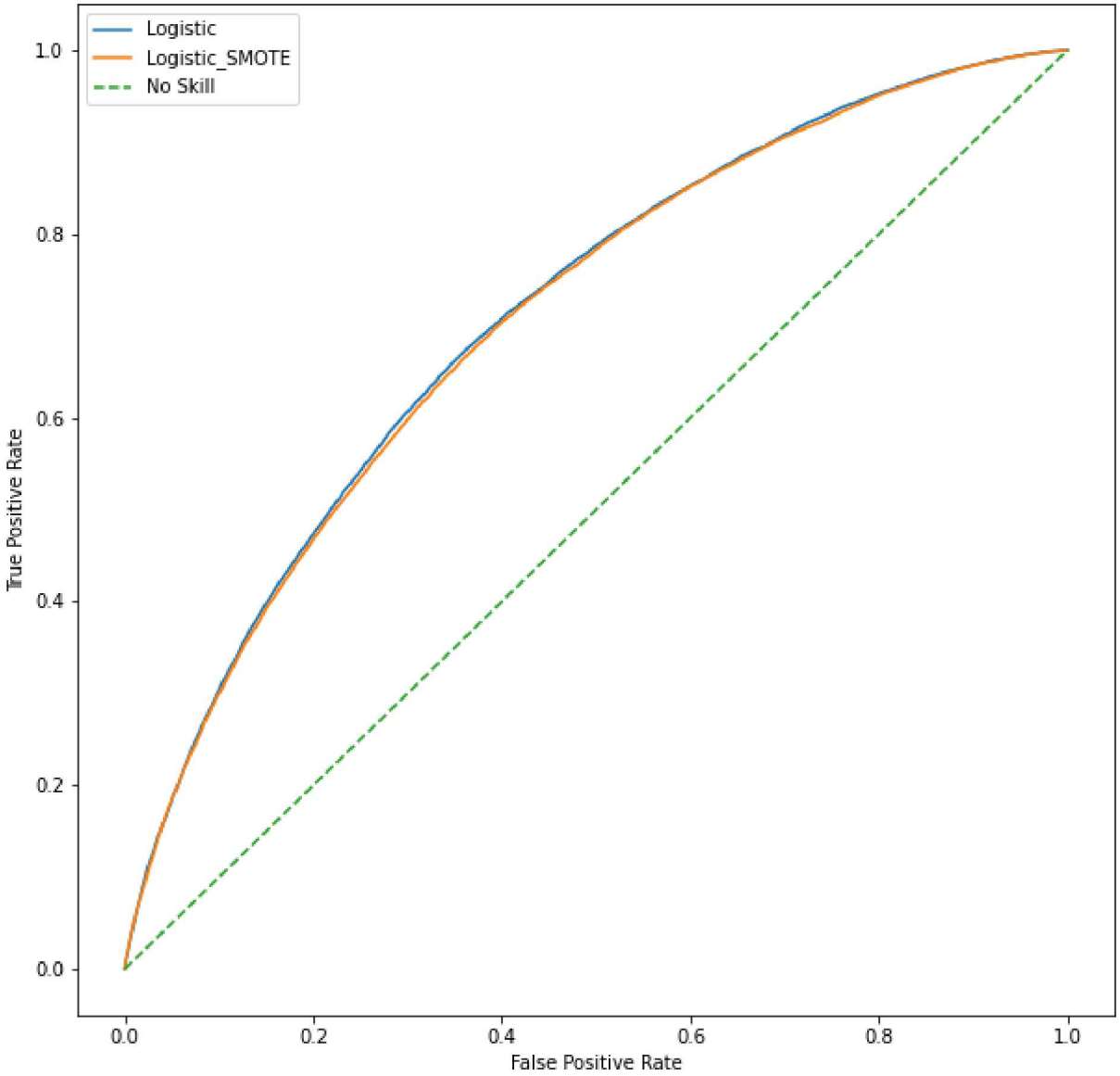
In [74]:
```
lr_probs_2 = log_reg_2.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs_2 = lr_probs_2[:, 1]
# calculate scores
lr_auc_2 = roc_auc_score(y_test, lr_probs_2)
# summarize scores
print('Logistic: ROC AUC=%.3f' % (lr_auc_2))
```

```
# calculate roc curves
lr_fpr_2, lr_tpr_2, _ = roc_curve(y_test, lr_probs_2)
```

```
Logistic: ROC AUC=0.708
```

In [75]:

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

# plot the roc curve for the model
plt.plot(lr_fpr, lr_tpr, label='Logistic')
plt.plot(lr_fpr_2, lr_tpr_2, label='Logistic_SMOTE')
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```

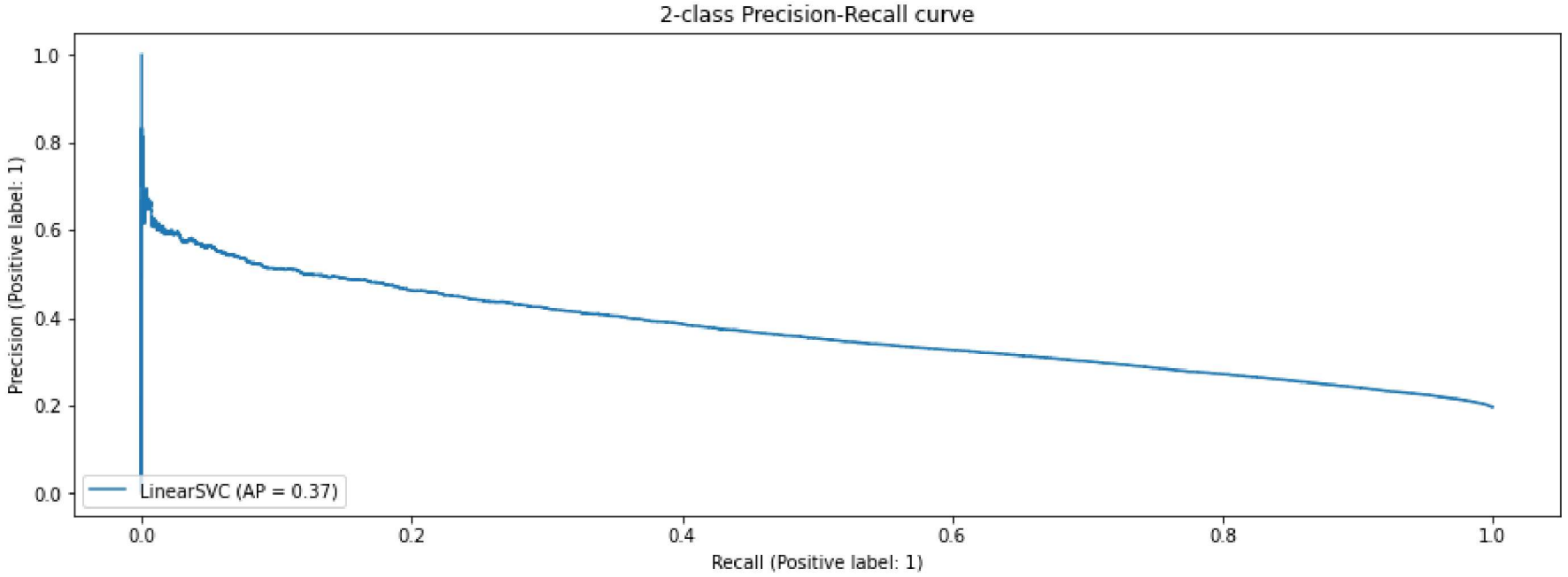


Observations

- The ROC-AUC curve is almost the same for both the models

In [76]:
```
precision_, recall_, proba = precision_recall_curve(y_test, lr_probs_2)
```

In [77]:
```
fig, ax = plt.subplots(figsize=(15, 5))
display = PrecisionRecallDisplay.from_predictions(y_test, lr_probs_2, name="LinearSVC", ax=ax)
_ = display.ax_.set_title("2-class Precision-Recall curve")
```



In [78]:
```
optimal_proba_cutoff = sorted(list(zip(np.abs(precision_ - recall_), proba)), key=lambda i: i[0], reverse=False)[0][1]
roc_predictions = [1 if i >= optimal_proba_cutoff else 0 for i in lr_probs_2]
```

In [79]:
```
print(optimal_proba_cutoff)
```

```
0.6347665841992145
```

In [80]:
```
print(classification_report(y_actual, roc_predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.85      0.85     63671
         1.0       0.39      0.39      0.39     15535

    accuracy                           0.76     79206
   macro avg       0.62      0.62      0.62     79206
weighted avg       0.76      0.76      0.76     79206
```

In [81]:
```
print(classification_report(list(y_actual), list(y_predicted_2)))
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.64      0.75     63671
         1.0       0.31      0.66      0.42     15535

    accuracy                           0.65     79206
   macro avg       0.60      0.65      0.58     79206
weighted avg       0.77      0.65      0.68     79206
```

From precision-recall curve, we try to find the optimal threshold. It comes out to be 0.634. However, the threshold should be tuned to match the business objectives.

With the new thresholds, the precision does not increase and hence the bank needs to take the decison wisely.

Questionnaire (Answers should present in the text editor along with insights):

- What percentage of customers have fully paid their Loan Amount? - **80%**
- Comment about the correlation between Loan Amount and Installment features. - **High correlation**
- The majority of people have home ownership as **MORTGAGE**.
- People with grades 'A' are more likely to fully pay their loan. **(T)**
- Name the top 2 afforded job titles. - **Teacher and Manager**
- Thinking from a bank's perspective, which metric should our primary focus be on..
  - ROC AUC
  - **Precision - correct** High precision means the bank will be able to identify the defaulters accurately
  - Recall
  - F1 Score
- How does the gap in precision and recall affect the bank?
  - It is important that the focus of the bank be on precision in order to predict the defaulters correctly. Lower rates of FP is important to the bank as well so that it does not identify the incorrect defaulters. However, if the recall is high, which means the FN is low, this would also benefit the bank so that the it does not disburse loans to actual defaulters. It depends on the strategy that the bank is planning to implement.
- Which were the features that heavily affected the outcome?

    - Grade
    - Application type
    - Purpose
    - Interest rate
    - term
- Will the results be affected by geographical location? **Yes**

In [ ]: