LoanTap Logistic Regression

Context:

- LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen.
- LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.
- LoanTap deploys formal credit to salaried individuals and businesses 4 main financial instruments: Personal Loan, EMI Free Loan, Personal Overdraft and Advance Salary Loan. This case study will focus on the underwriting process behind Personal Loan only.

Problem Statement:

- Help LoanTap to determine if a credit line should be extended to individuals or not.
- Help LoanTap to set the repayment terms ang give business recommendations regarding the same.

Column Profiling

Exploratory Data Analysis

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

In [2]:
```
df = pd.read_csv(r'\Users\Home\Downloads\logistic_regression.csv')
```

In [3]:
```
df
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pub_rec	rev
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	0.0	36
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	0.0	20
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	0.0	11
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	6.0	0.0	5
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	13.0	0.0	24
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
396025	10000.0	60 months	10.99	217.38	B	B4	licensed bankere	2 years	RENT	40000.0	...	6.0	0.0	1
396026	21000.0	36 months	12.29	700.42	C	C1	Agent	5 years	MORTGAGE	110000.0	...	6.0	0.0	43
396027	5000.0	36 months	9.99	161.32	B	B1	City Carrier	10+ years	RENT	56500.0	...	15.0	0.0	32
396028	21000.0	60 months	15.31	503.02	C	C2	Gracon Services, Inc	10+ years	MORTGAGE	64000.0	...	9.0	0.0	15
396029	2000.0	36 months	13.61	67.98	C	C2	Internal Revenue Service	10+ years	RENT	42996.0	...	3.0	0.0	4

396030 rows × 27 columns

In [4]:

```
df.shape #shape of data
```

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

In [5]:
```
df.info() #data type of all attributes
```

```
<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
```

In [6]:
```
df.describe(include='all') #statistical summary
```

Out[6]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_
count	396030.000000	396030	396030.000000	396030.000000	396030	396030	373103	377729	396030	3.960300e+05	...	396030.000
unique	NaN	2	NaN	NaN	7	35	173105	11	6	NaN	...	N
top	NaN	36 months	NaN	NaN	B	B3	Teacher	10+ years	MORTGAGE	NaN	...	N
freq	NaN	302005	NaN	NaN	116018	26655	4389	126041	198348	NaN	...	N
mean	14113.888089	NaN	13.639400	431.849698	NaN	NaN	NaN	NaN	NaN	7.420318e+04	...	11.311
std	8357.441341	NaN	4.472157	250.727790	NaN	NaN	NaN	NaN	NaN	6.163762e+04	...	5.137
min	500.000000	NaN	5.320000	16.080000	NaN	NaN	NaN	NaN	NaN	0.000000e+00	...	0.000
25%	8000.000000	NaN	10.490000	250.330000	NaN	NaN	NaN	NaN	NaN	4.500000e+04	...	8.000
50%	12000.000000	NaN	13.330000	375.430000	NaN	NaN	NaN	NaN	NaN	6.400000e+04	...	10.000
75%	20000.000000	NaN	16.490000	567.300000	NaN	NaN	NaN	NaN	NaN	9.000000e+04	...	14.000
max	40000.000000	NaN	30.990000	1533.810000	NaN	NaN	NaN	NaN	NaN	8.706582e+06	...	90.000

11 rows × 27 columns

Columns such as Loan Amount, Installments, Annual Income and revol_bal have large difference in mean and median. This implies that outliers are present in the data.

In [7]:
```
df.isna().sum() #checking for null values
```

Out[7]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 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                    1755
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
```

In [8]:
```
df.nunique()
```

Out[8]:
```
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                    48817
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
```

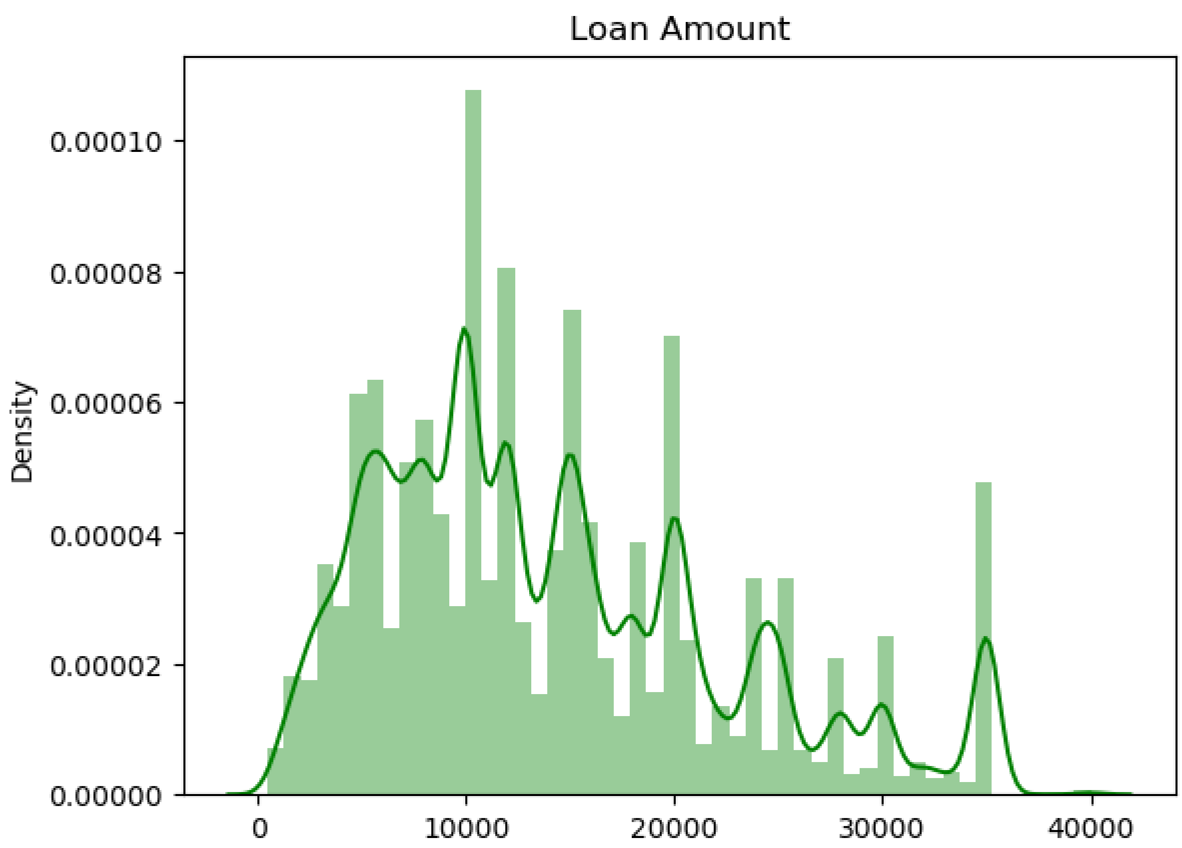
In [9]:
```
#converting string to datetime format
df['issue_d'] = pd.to_datetime(df['issue_d'])
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
```

Univariate Analysis

In [10]:
```
sns.distplot(x=df['loan_amnt'],color="Green")
plt.title('Loan Amount')
```

```
C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and
will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexib
ility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[10]: Text(0.5, 1.0, 'Loan Amount')

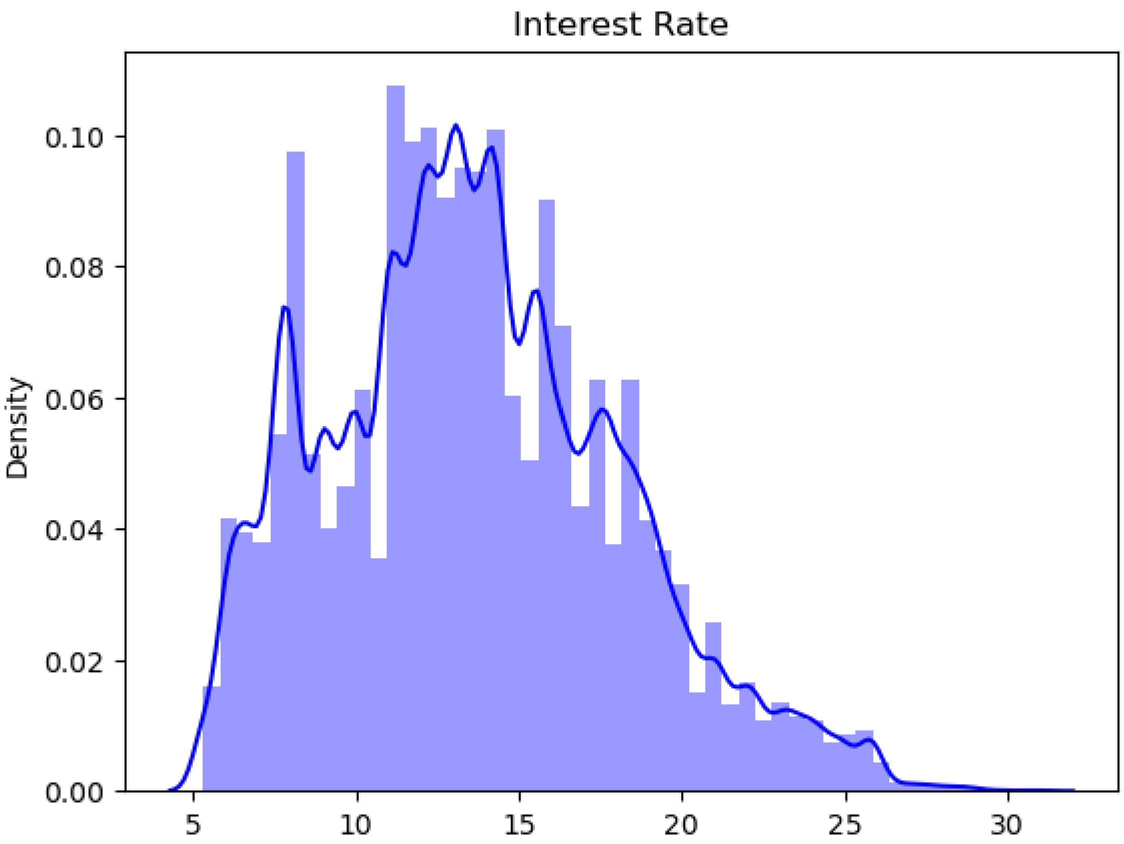


Loan Amount data is right skewed.

In [11]:
```
sns.distplot(x=df['int_rate'],color="Blue")
plt.title('Interest Rate')
```

```
C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[11]: Text(0.5, 1.0, 'Interest Rate')

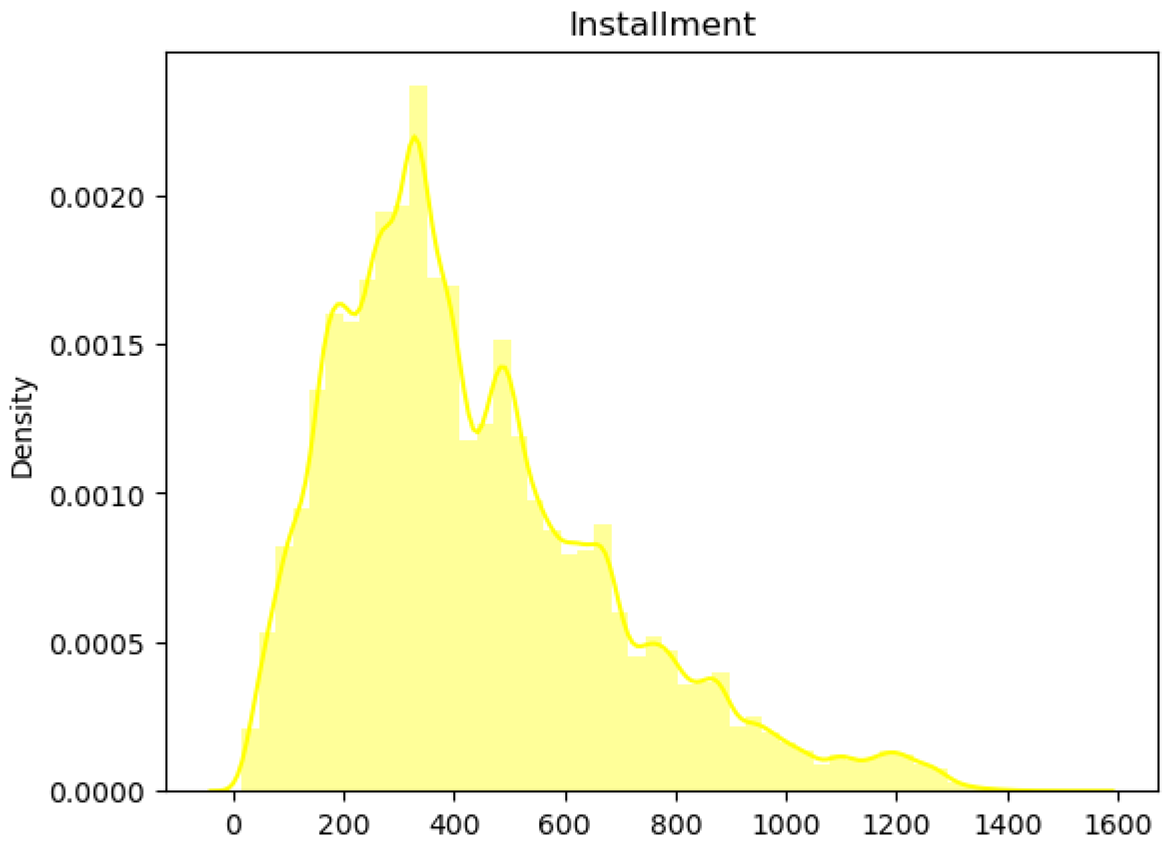


Interest rate is normally distributed.

In [12]:
```
sns.distplot(x=df['installment'],color="Yellow")
plt.title('Installment')
```

```
C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[12]:
```
Text(0.5, 1.0, 'Installment')
```

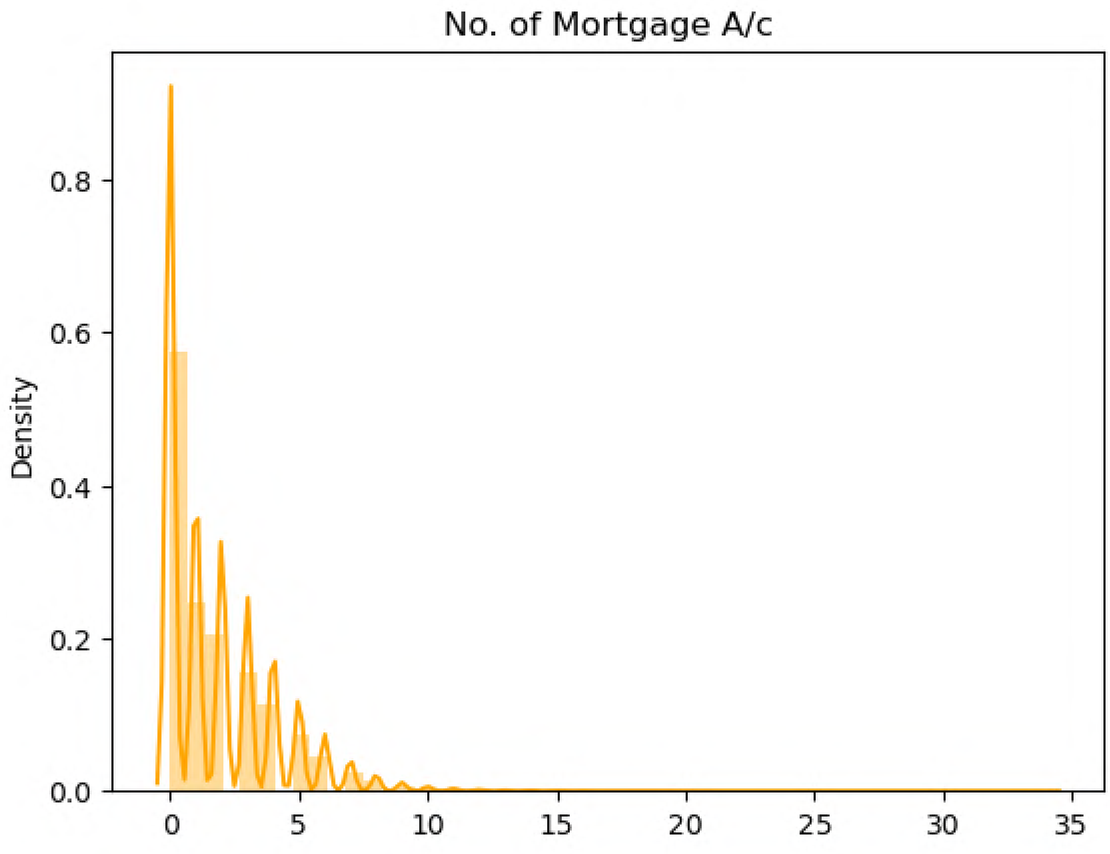


Installment amount data is also right skewed.

In [13]:
```
sns.distplot(x=df['mort_acc'],color="Orange")
plt.title('No. of Mortgage A/c')
```

```
C:\Users\Home\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

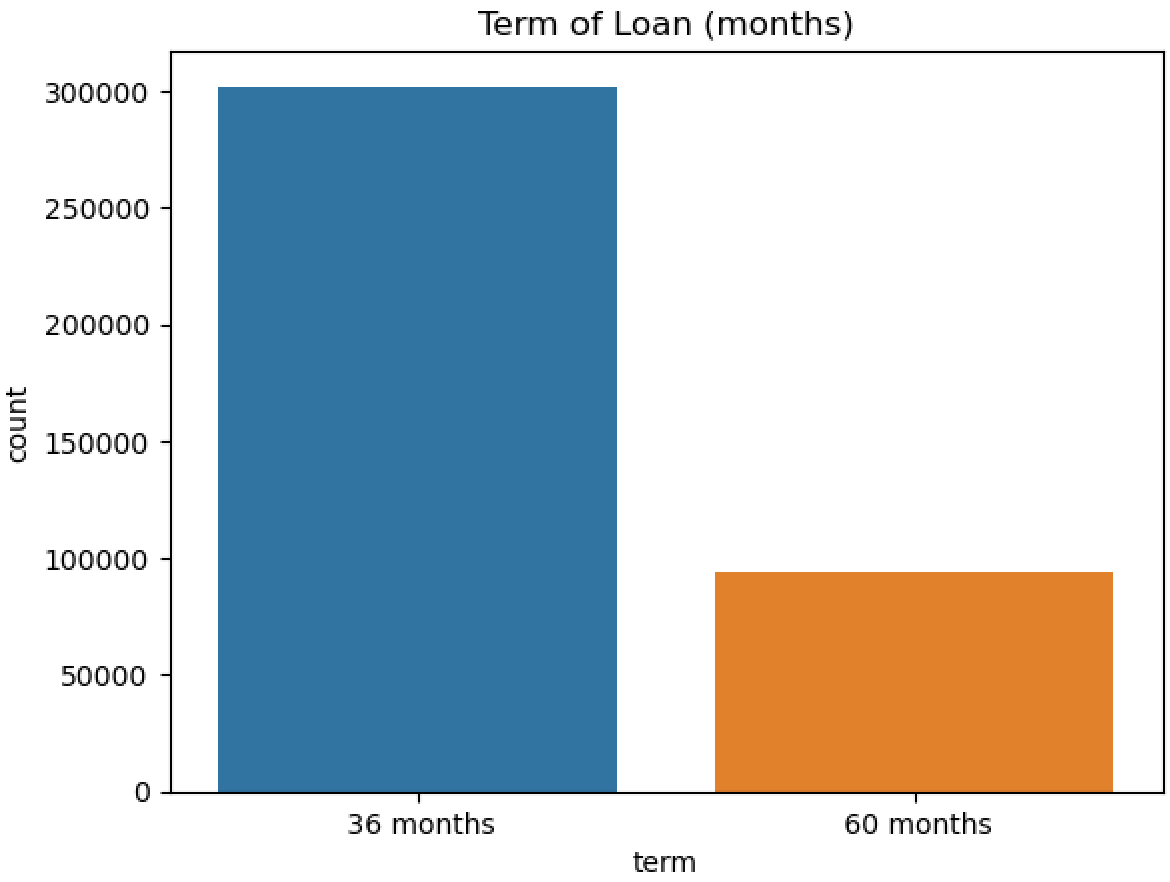
Out[13]:
```
Text(0.5, 1.0, 'No. of Mortgage A/c')
```



Mortgage Account amount data is also right skewed.

In [14]:
```
sns.countplot(x=df['term'])
plt.title('Term of Loan (months)')
df.term.value_counts()
```

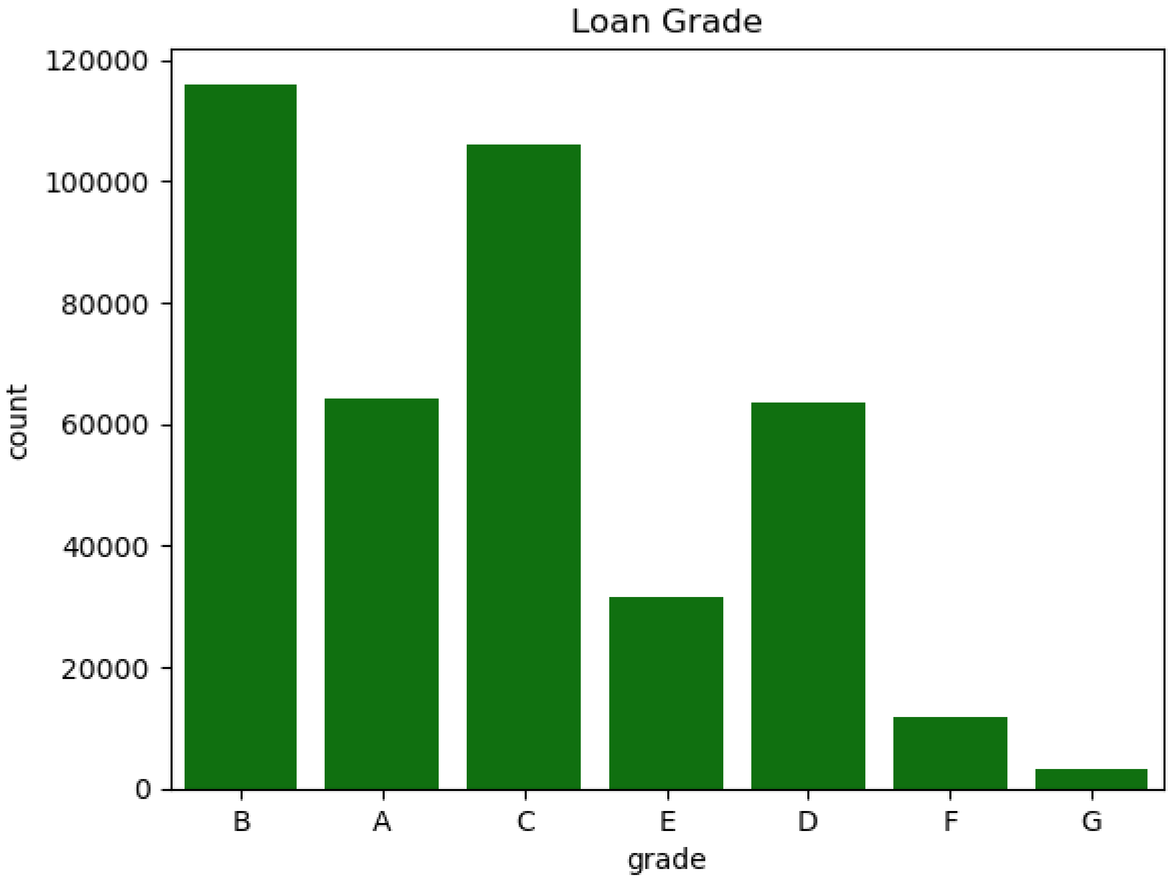
Out[14]:
```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```



302005 people prefer payment term of 36 months, while 94025 prefer 60 months as payment term.

In [15]:
```
sns.countplot(x=df['grade'],color="Green")
plt.title('Loan Grade')
df.grade.value_counts()
```

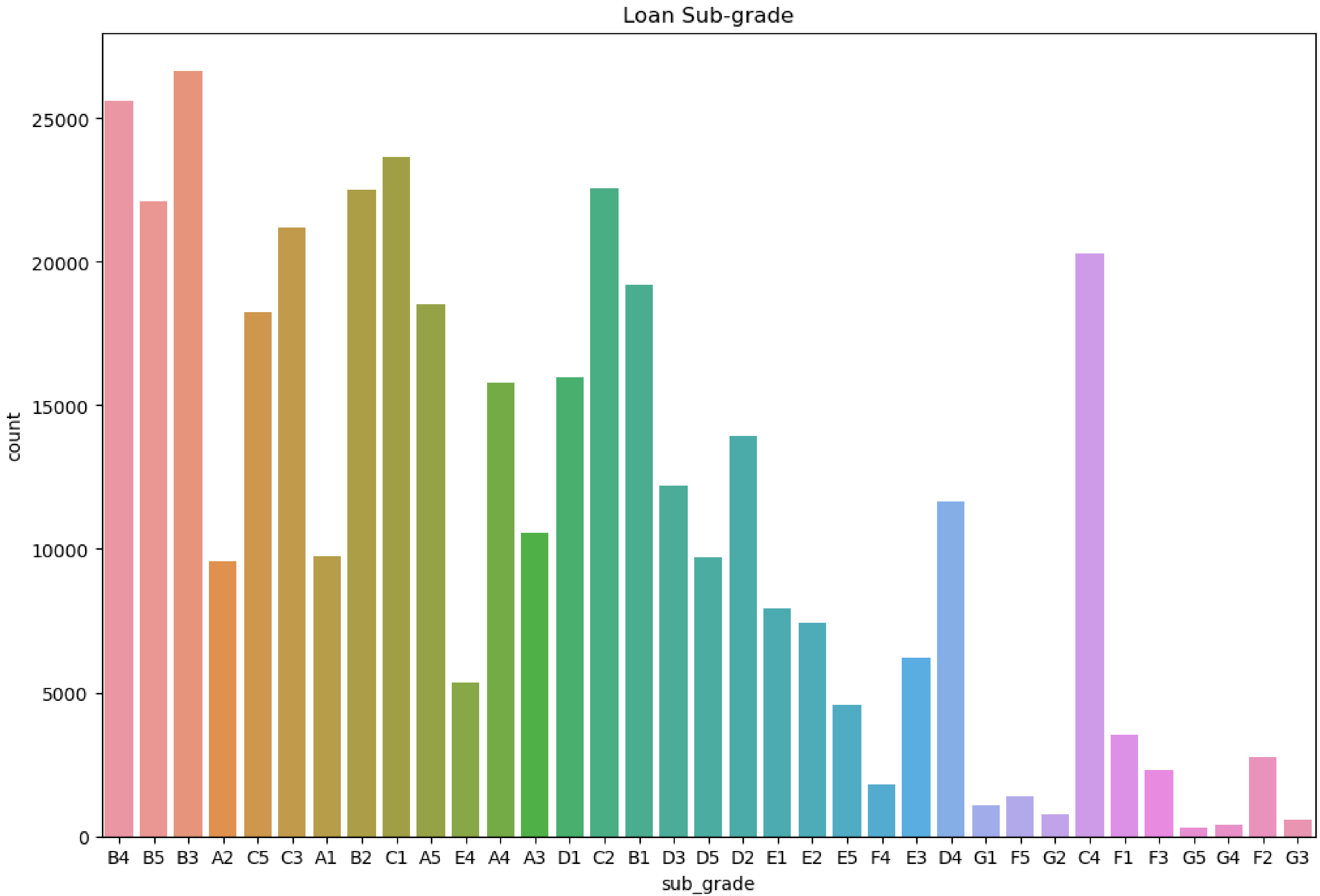
Out[15]:
```
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: grade, dtype: int64
```



Highest amount of loan takers lie in the Grade B category.

In [16]:
```
plt.figure(figsize=(12, 8))
sns.countplot(x=df['sub_grade'])
plt.title('Loan Sub-grade')
df.sub_grade.value_counts()
```

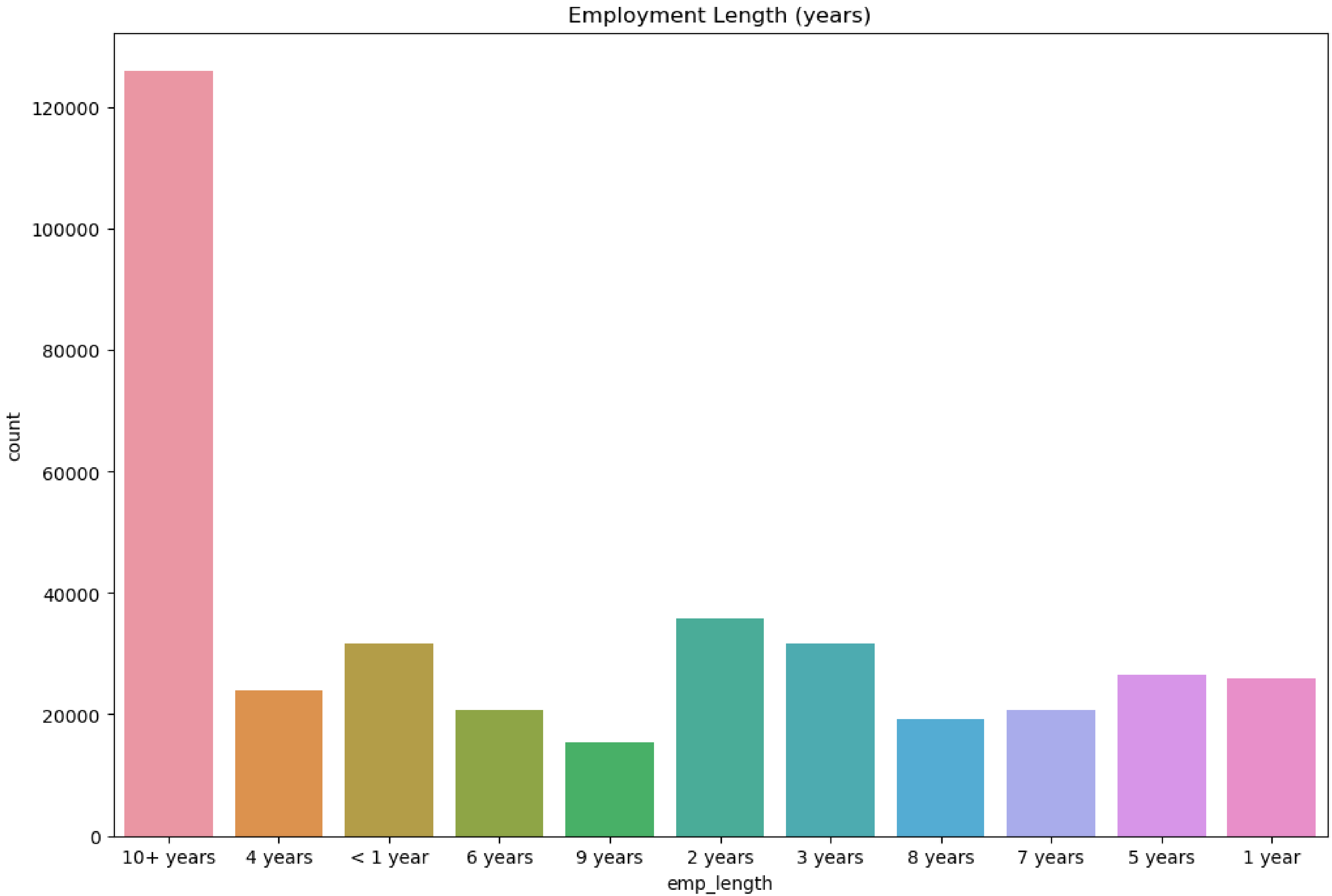
Out[16]:
```
B3    26655
B4    25601
C1    23662
C2    22580
B2    22495
B5    22085
C3    21221
C4    20280
B1    19182
A5    18526
C5    18244
D1    15993
A4    15789
D2    13951
D3    12223
D4    11657
A3    10576
A1     9729
D5     9700
A2     9567
E1     7917
E2     7431
E3     6207
E4     5361
E5     4572
F1     3536
F2     2766
F3     2286
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: sub_grade, dtype: int64
```



Highest amount of loan takers lie in the Grade B3 category.

In [17]:
```
plt.figure(figsize=(12, 8))
sns.countplot(x=df['emp_length'])
plt.title('Employment Length (years)')
df.emp_length.value_counts()
```

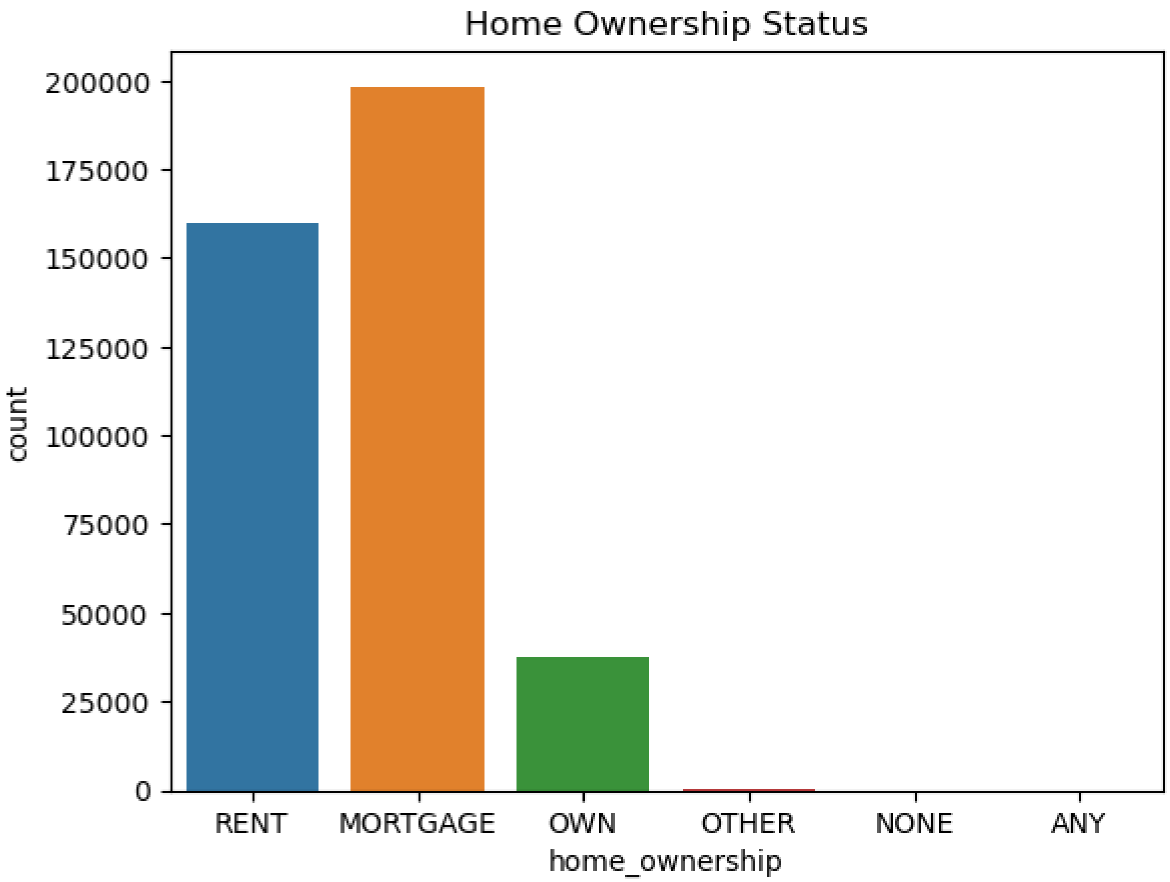
Out[17]:
```
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
9 years       15314
Name: emp_length, dtype: int64
```



Most number of loan takers have 10+ years of employment experience.

In [18]:
```
sns.countplot(x=df['home_ownership'])
plt.title('Home Ownership Status')
df.home_ownership.value_counts()
```

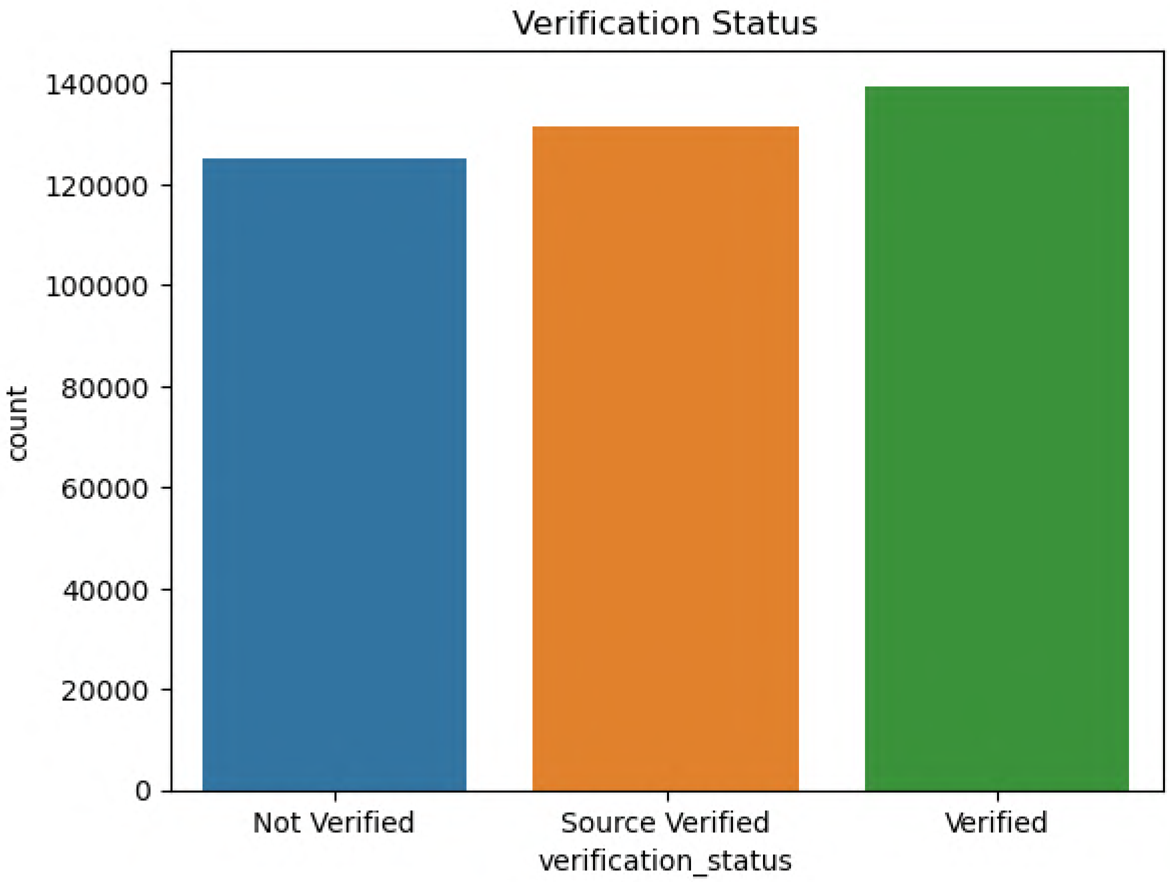
Out[18]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```



People with home ownership status as mortgage are the highest number of loan takers.

In [19]:
```
sns.countplot(x=df['verification_status'])
plt.title('Verification Status')
df.verification_status.value_counts()
```

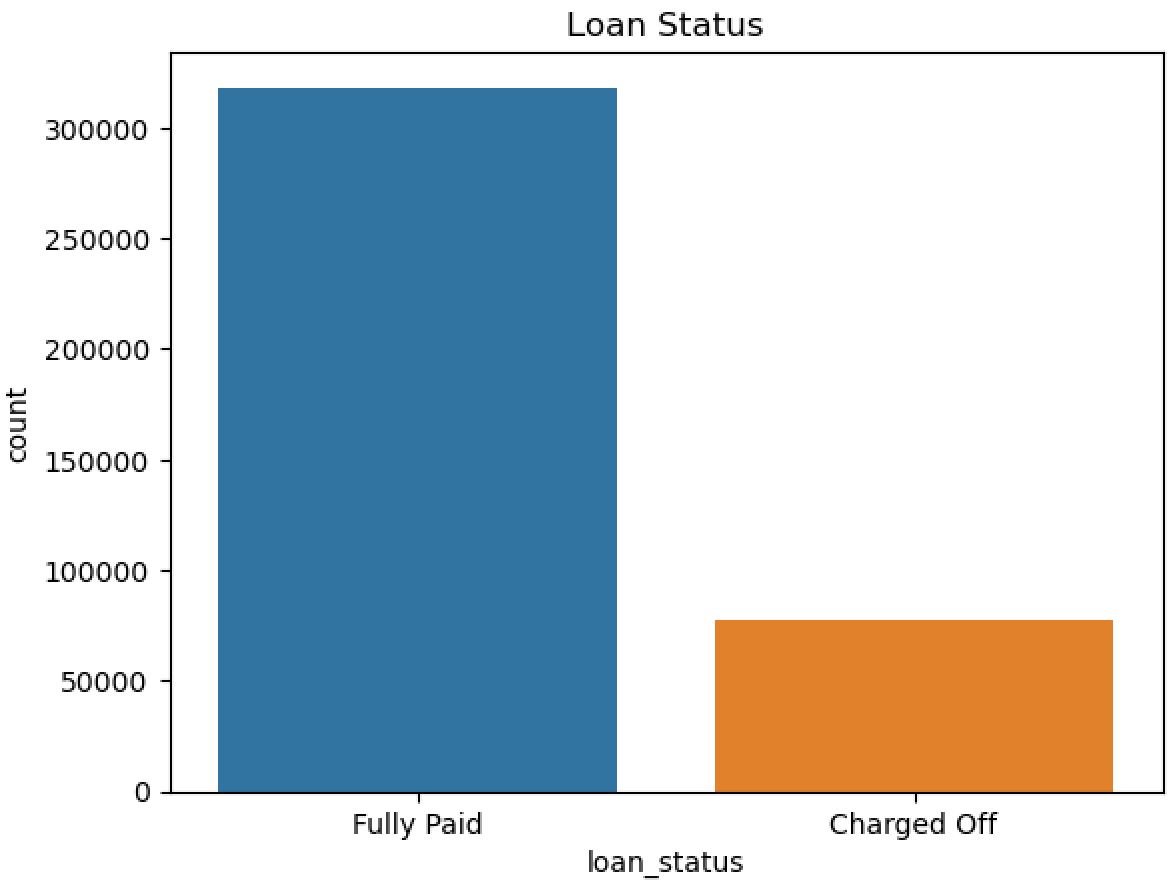
Out[19]:
```
Verified           139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64
```



125082 number of borrowers are not verified, while the rest are verified.

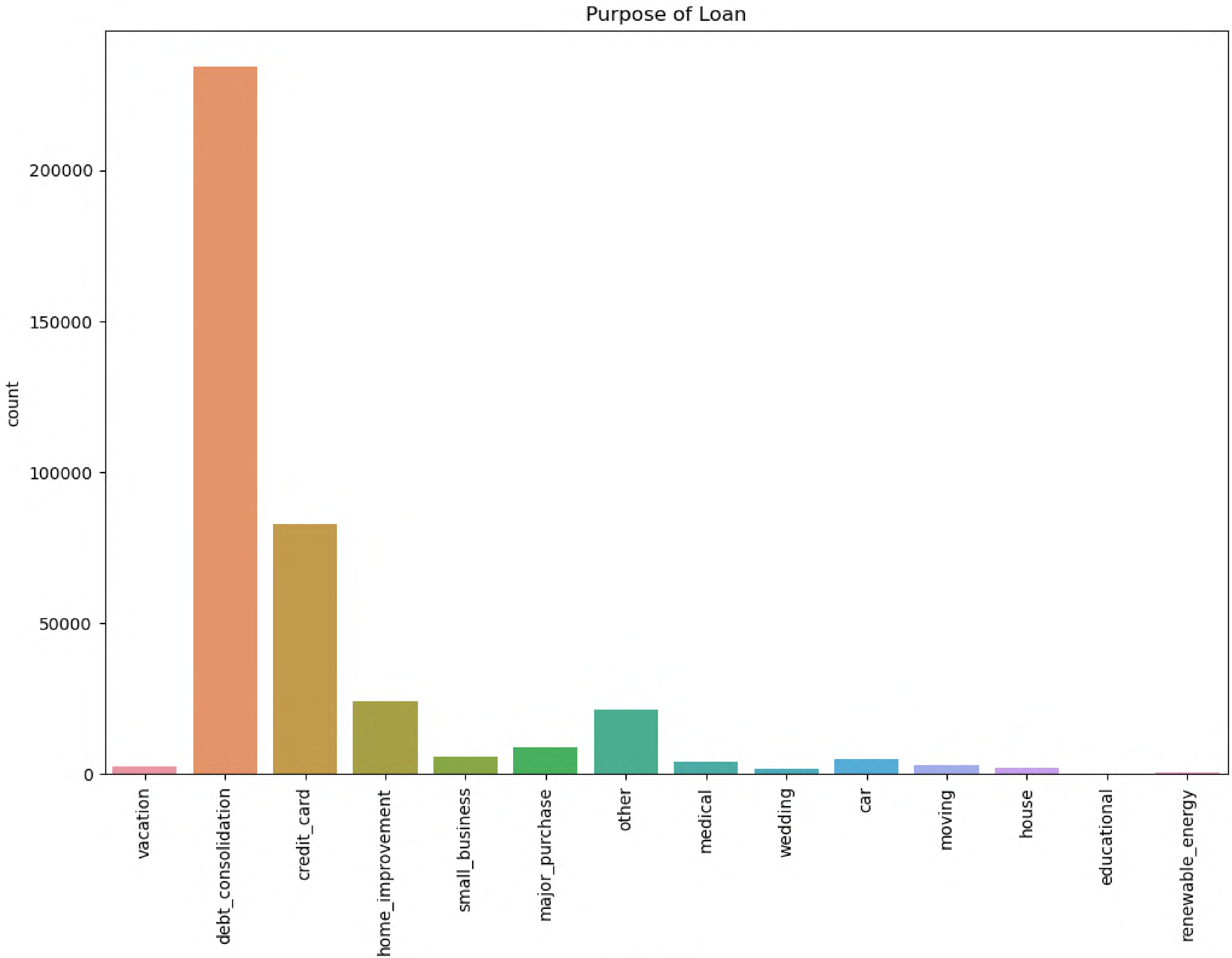
In [20]:
```
sns.countplot(x=df['loan_status'])
plt.title('Loan Status')
df.loan_status.value_counts(normalize=True)*100
```

Out[20]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```



318357 borrowers fully paid off the loan, while 77673 did not.

In [21]:
```
plt.figure(figsize=(12, 8))
sns.countplot(x=df['purpose'])
plt.title('Purpose of Loan')
plt.xticks(rotation = 90)
plt.show()
```

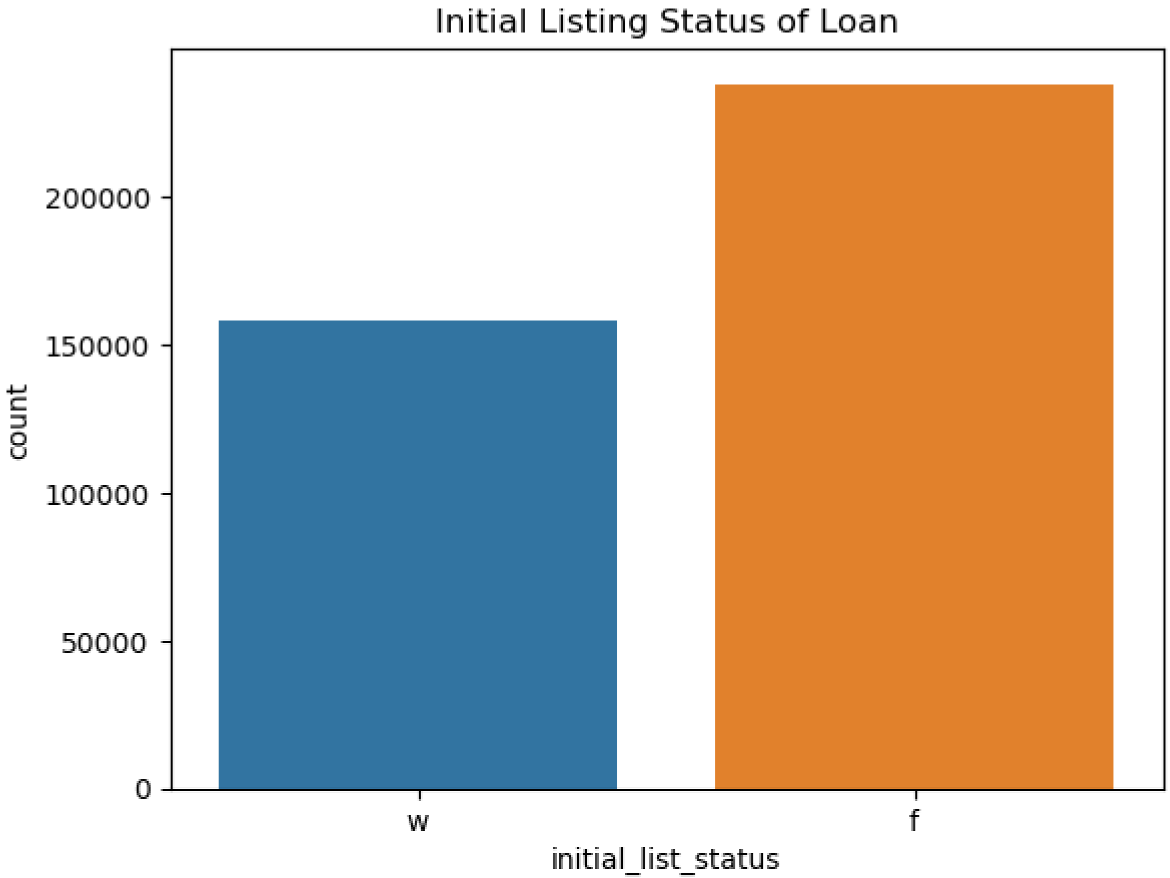

Purpose of Loan
count
200000
150000
100000
50000
0
vacation
debt_consolidation
credit_card
home_improvement
small_business
major_purchase
other
medical
wedding
car
moving
house
educational
renewable_energy

purpose

Most number of borrowers take loan for the purpose of debt consolidation.

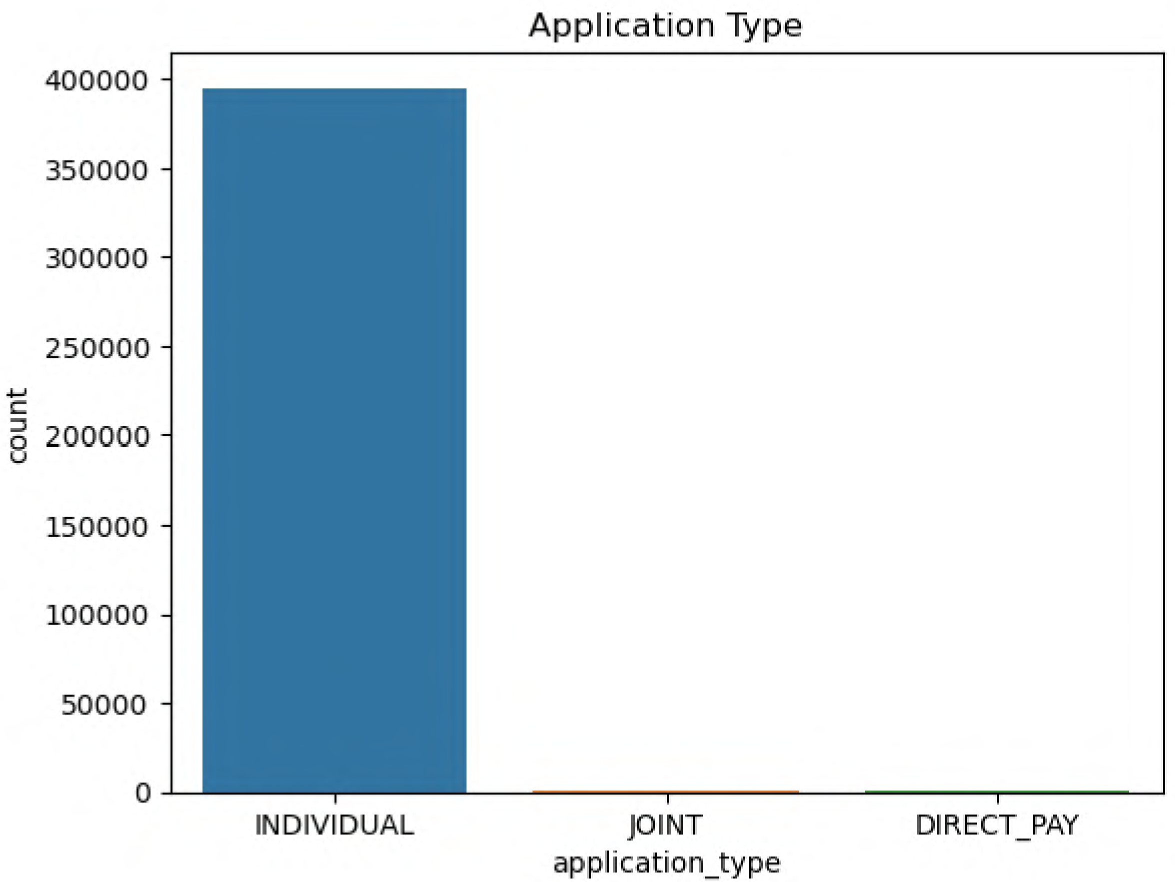
In [22]:
```
sns.countplot(x=df['initial_list_status'])
plt.title('Initial Listing Status of Loan')
df.initial_list_status.value_counts()
```

Out[22]:
```
f    238066
w    157964
Name: initial_list_status, dtype: int64
```



In [23]:
```
sns.countplot(x=df['application_type'])
plt.title('Application Type')
df.application_type.value_counts()
```

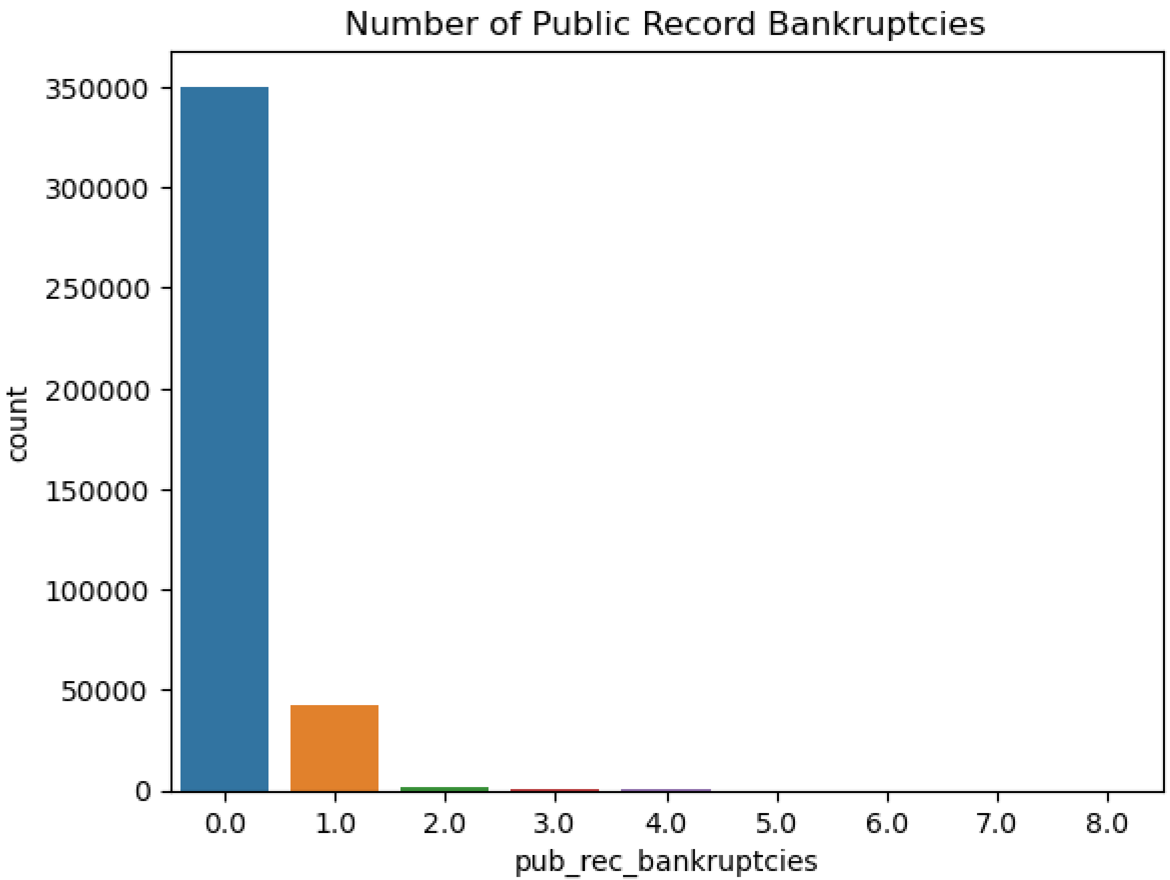
Out[23]:
```
INDIVIDUAL     395319
JOINT             425
DIRECT_PAY        286
Name: application_type, dtype: int64
```



Most number of borrowers are individuals.

In [24]:
```
sns.countplot(x=df['pub_rec_bankruptcies'])
plt.title('Number of Public Record Bankruptcies')
df.pub_rec_bankruptcies.value_counts()
```

Out[24]:
```
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
```

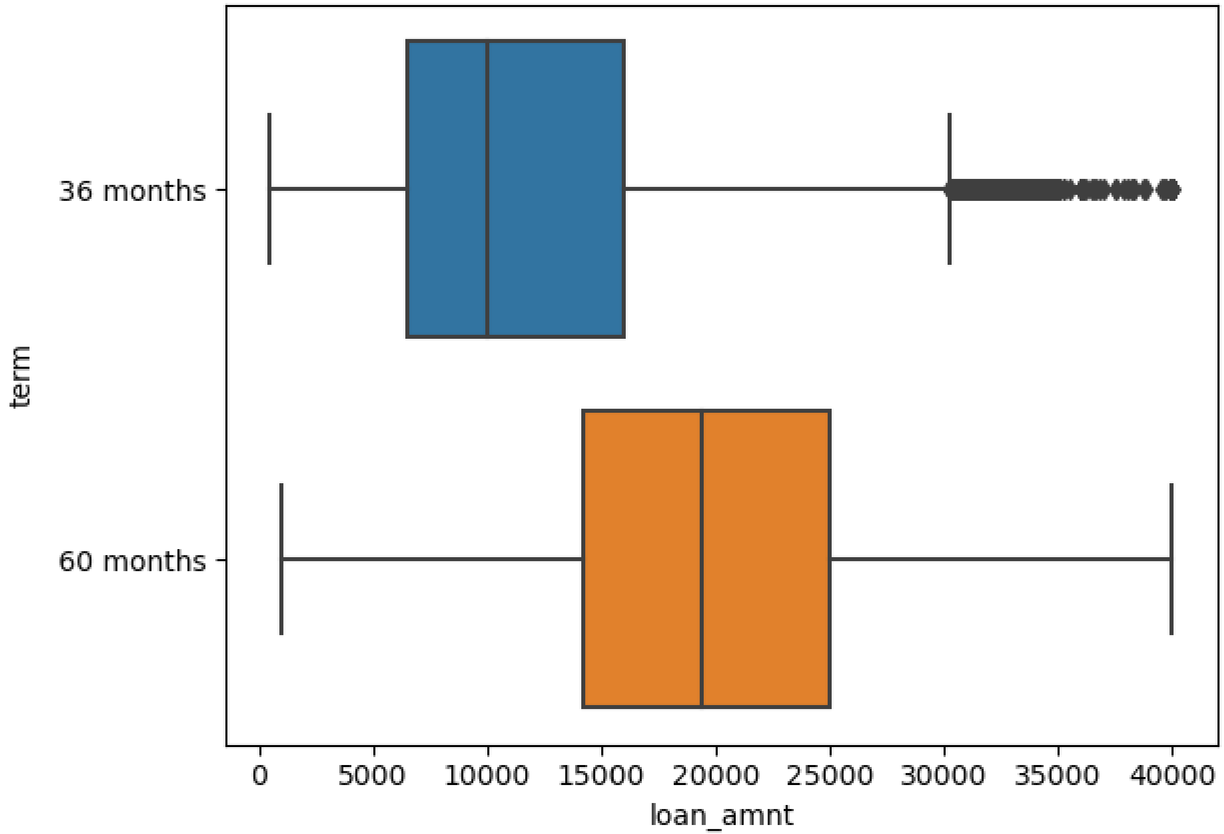


Most number of borrowers don't have any public record of bankruptcies.

Bivariate Analysis

In [25]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['term'], orient = 'h')
```

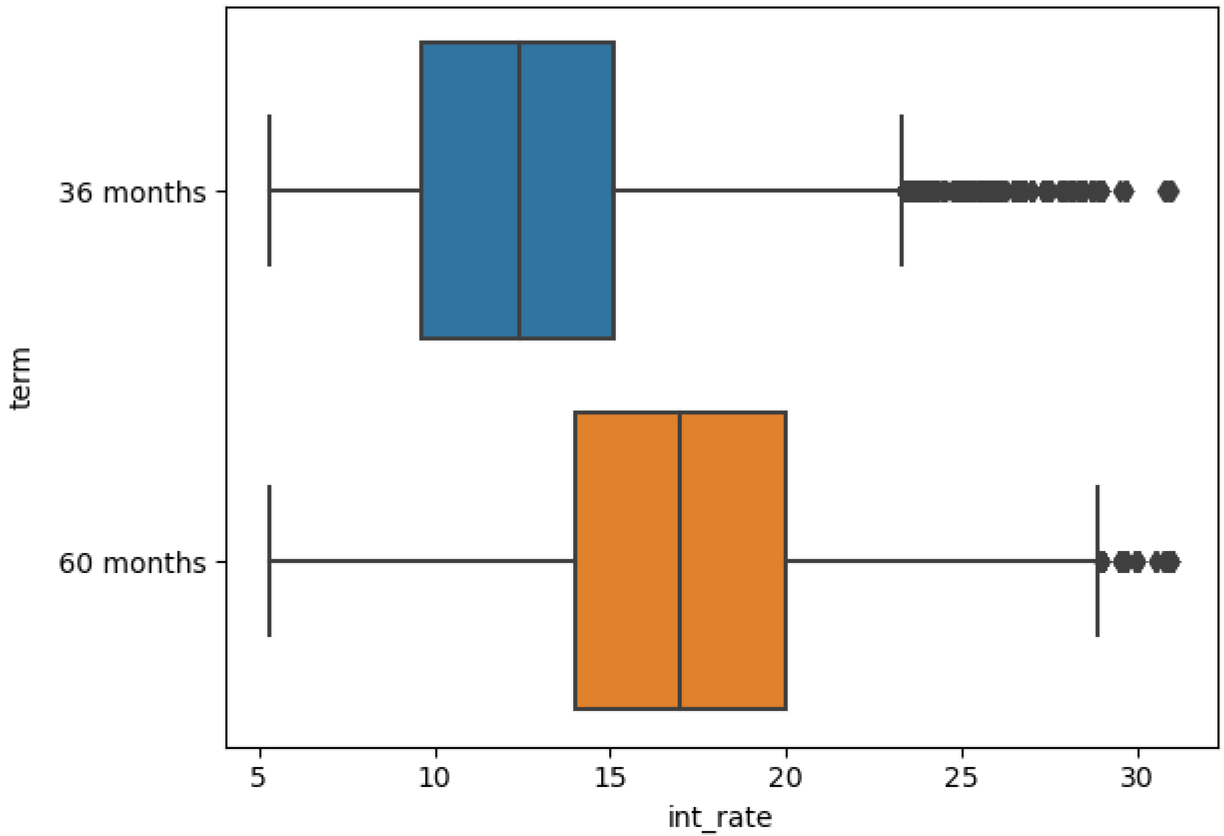
Out[25]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='term'>
```



Higher the loan amount, higher the term of the loan.

In [26]:
```
sns.boxplot(data = df, x = df['int_rate'], y = df['term'], orient = 'h')
```

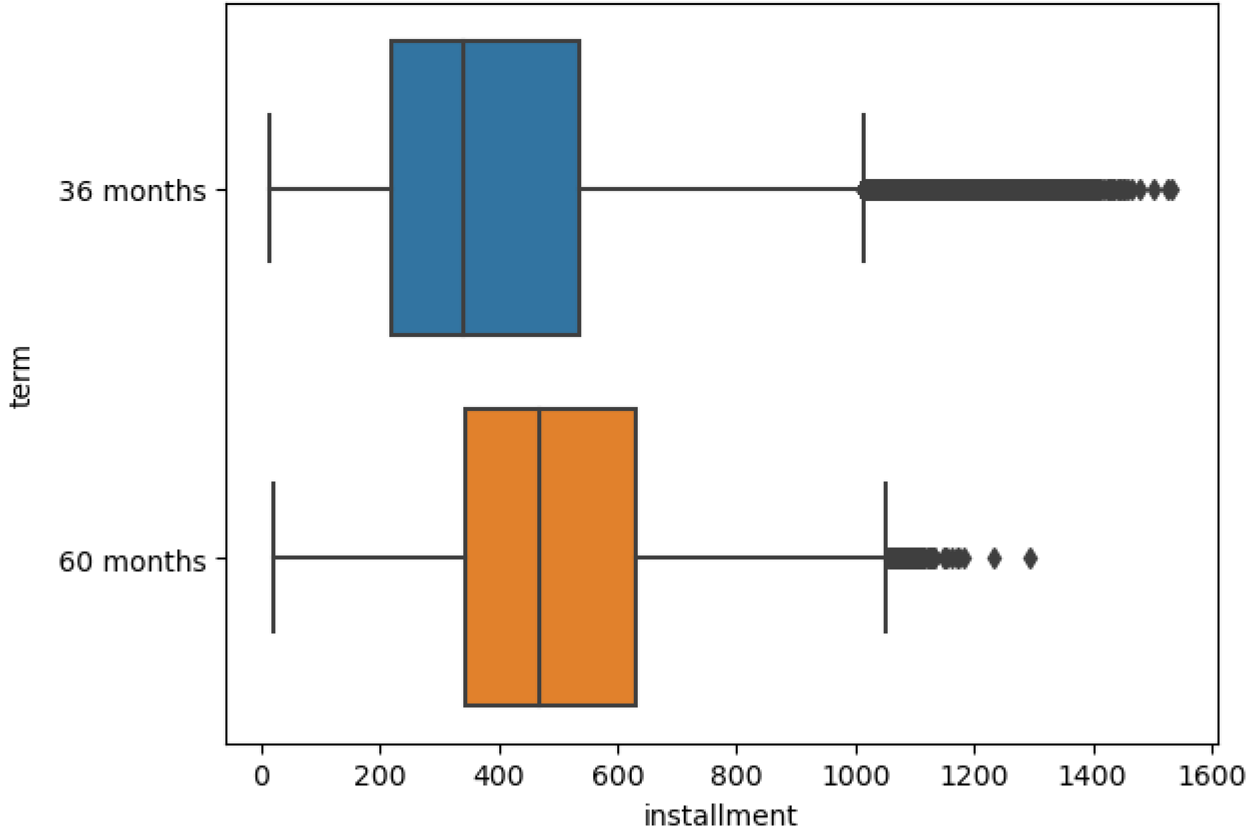
Out[26]:
```
<AxesSubplot:xlabel='int_rate', ylabel='term'>
```



Higher the term of the loan amount, higher the interest rate.

In [27]:
```
sns.boxplot(data = df, x = df['installment'], y = df['term'], orient = 'h')
```

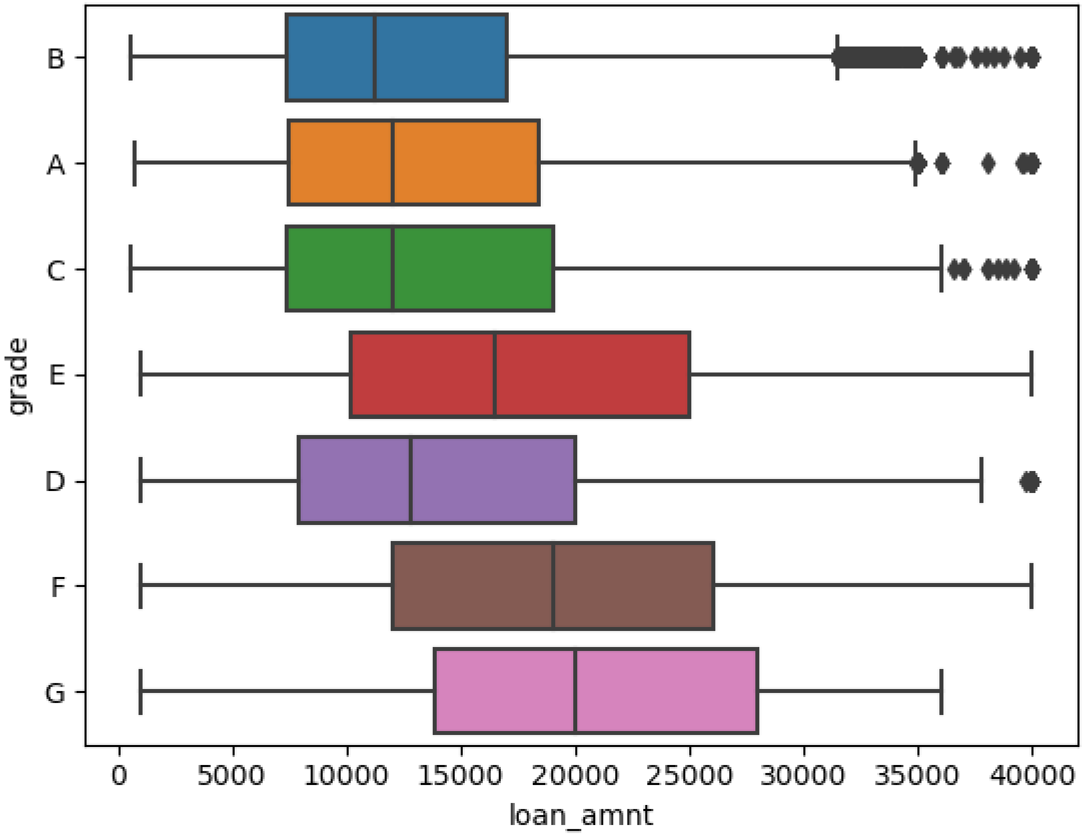
Out[27]:
```
<AxesSubplot:xlabel='installment', ylabel='term'>
```



The installment amount of loan amount for longer term is higher. This could be due to the high loan amount.

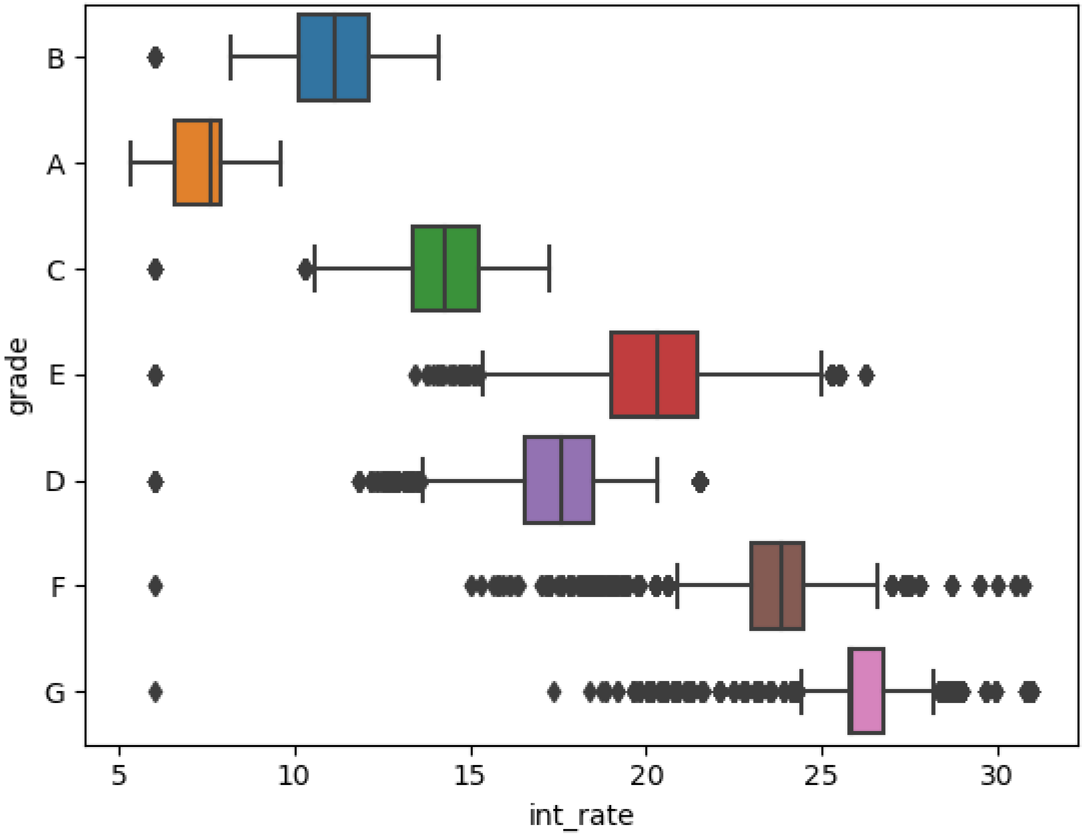
In [28]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['grade'], orient = 'h')
```

Out[28]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='grade'>
```



In [29]:
```
sns.boxplot(data = df, x = df['int_rate'], y = df['grade'], orient = 'h')
```

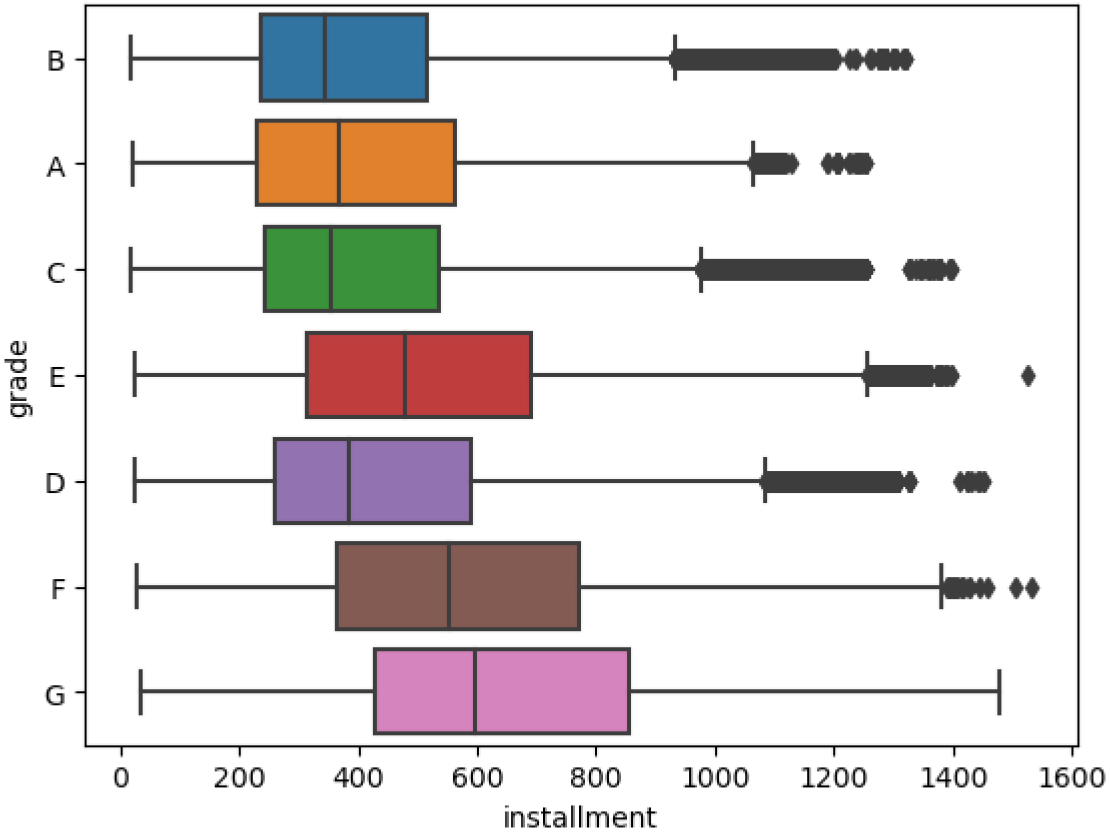
Out[29]:
```
<AxesSubplot:xlabel='int_rate', ylabel='grade'>
```



Borrowers in Grade A have the lowerst interest rate. This implies that it is safe to give them loans. While borrowers in Grade G category have the highest interest rate. This implies that it is risky to give them loan.

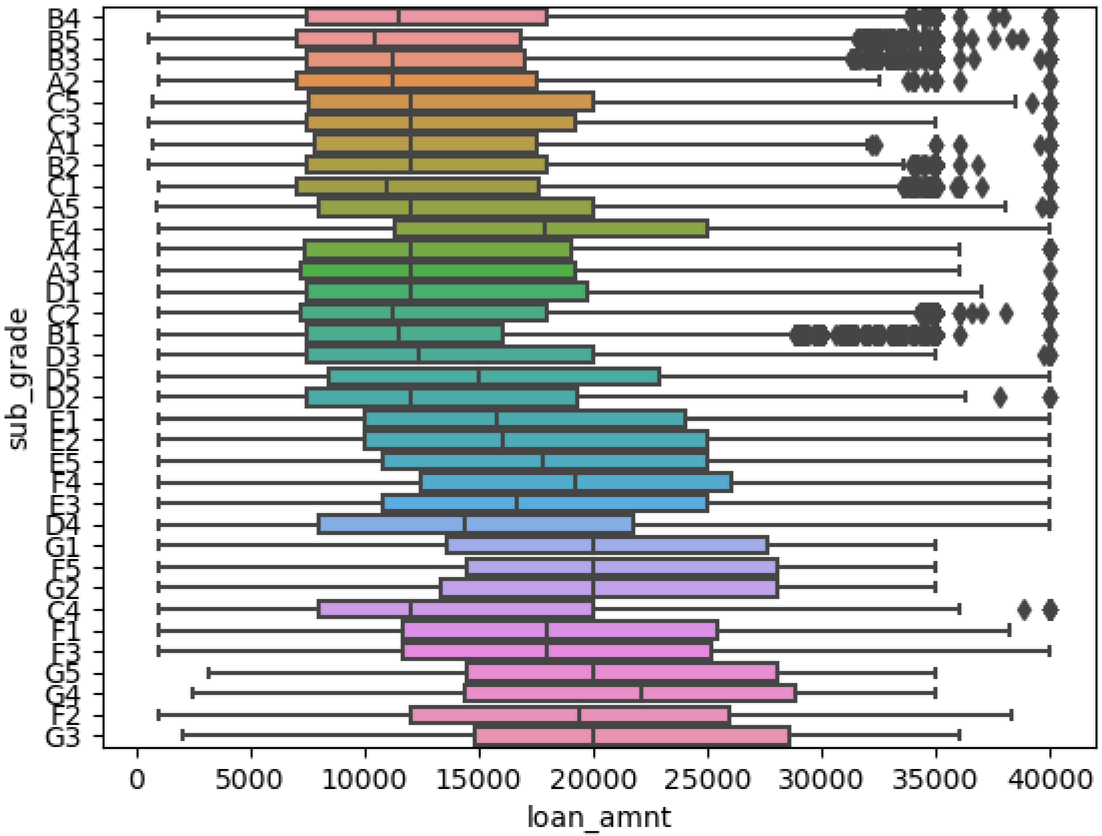
In [30]:
```
sns.boxplot(data = df, x = df['installment'], y = df['grade'], orient = 'h')
```

Out[30]:
```
<AxesSubplot:xlabel='installment', ylabel='grade'>
```



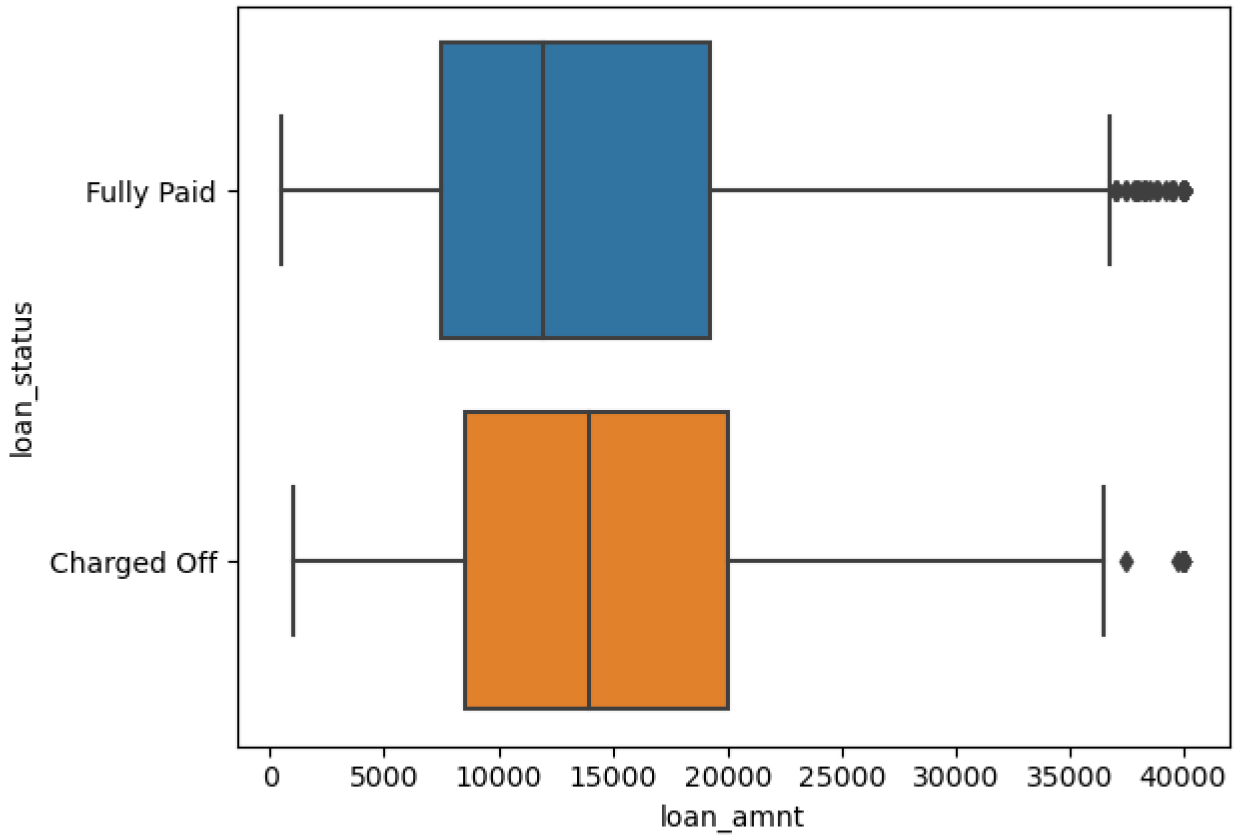
In [31]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['sub_grade'], orient = 'h')
```

Out[31]: <AxesSubplot:xlabel='loan_amnt', ylabel='sub_grade'>



In [32]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['loan_status'], orient = 'h')
```

Out[32]: <AxesSubplot:xlabel='loan_amnt', ylabel='loan_status'>



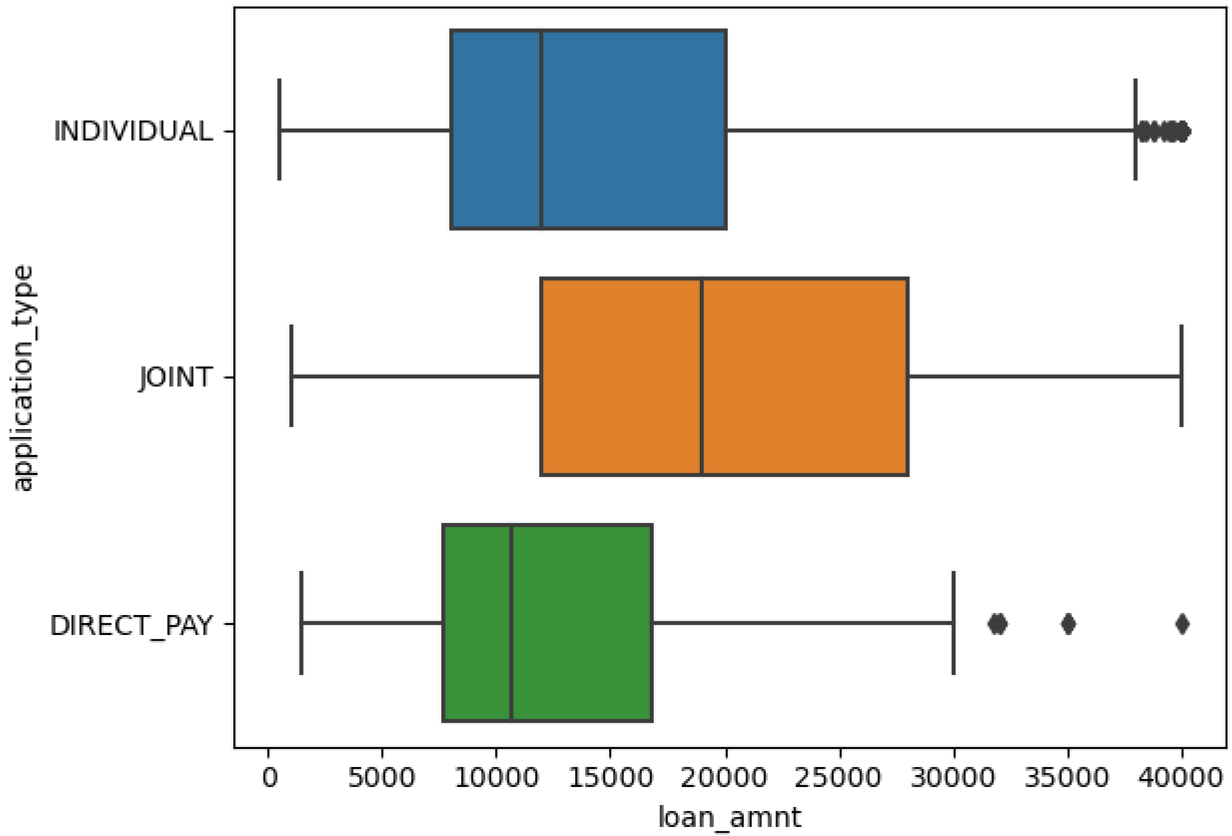
In [33]:
```
df.groupby(by='loan_status')['loan_amnt'].describe()
```

Out[33]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

In [34]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['application_type'], orient = 'h')
```

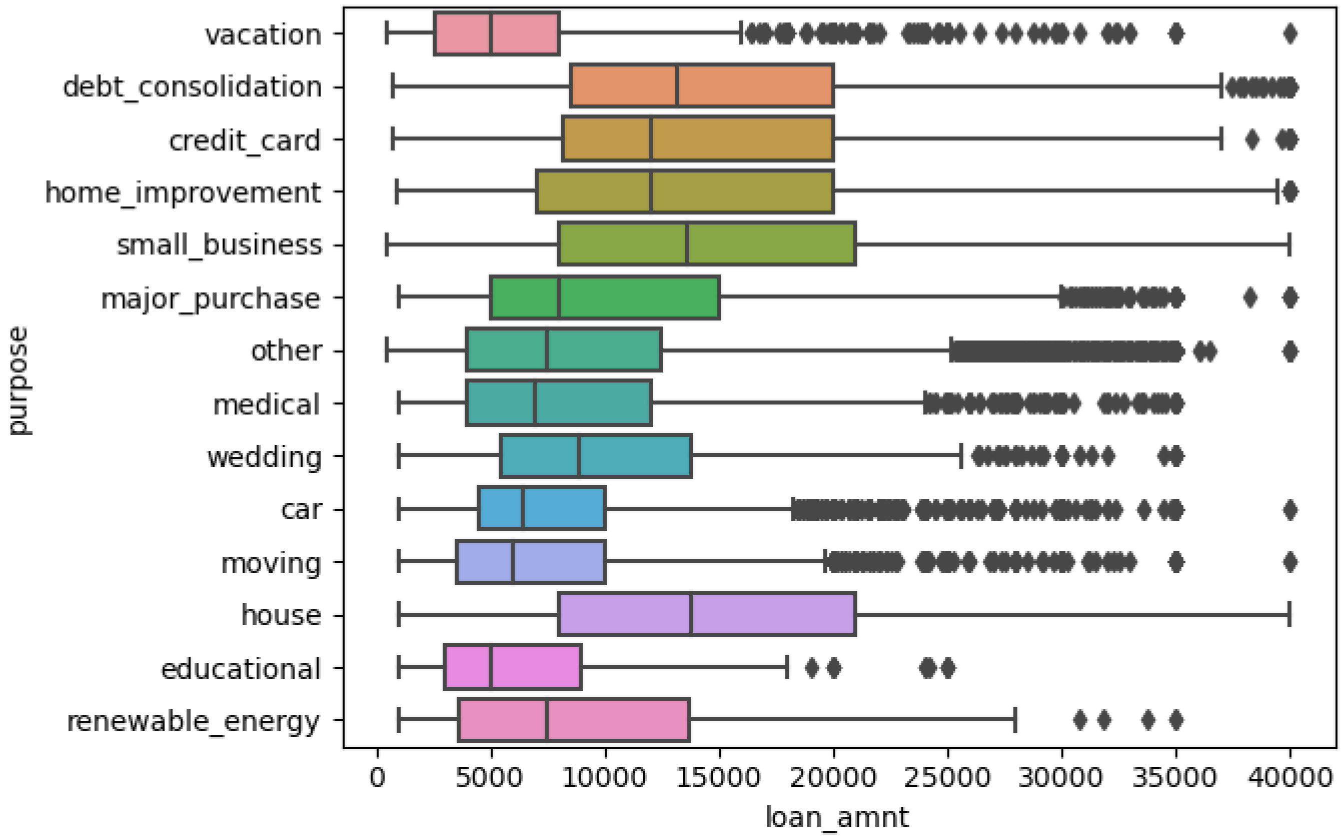
Out[34]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='application_type'>
```



Joint applicants take larger amount as loan.

In [35]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['purpose'], orient = 'h')
```

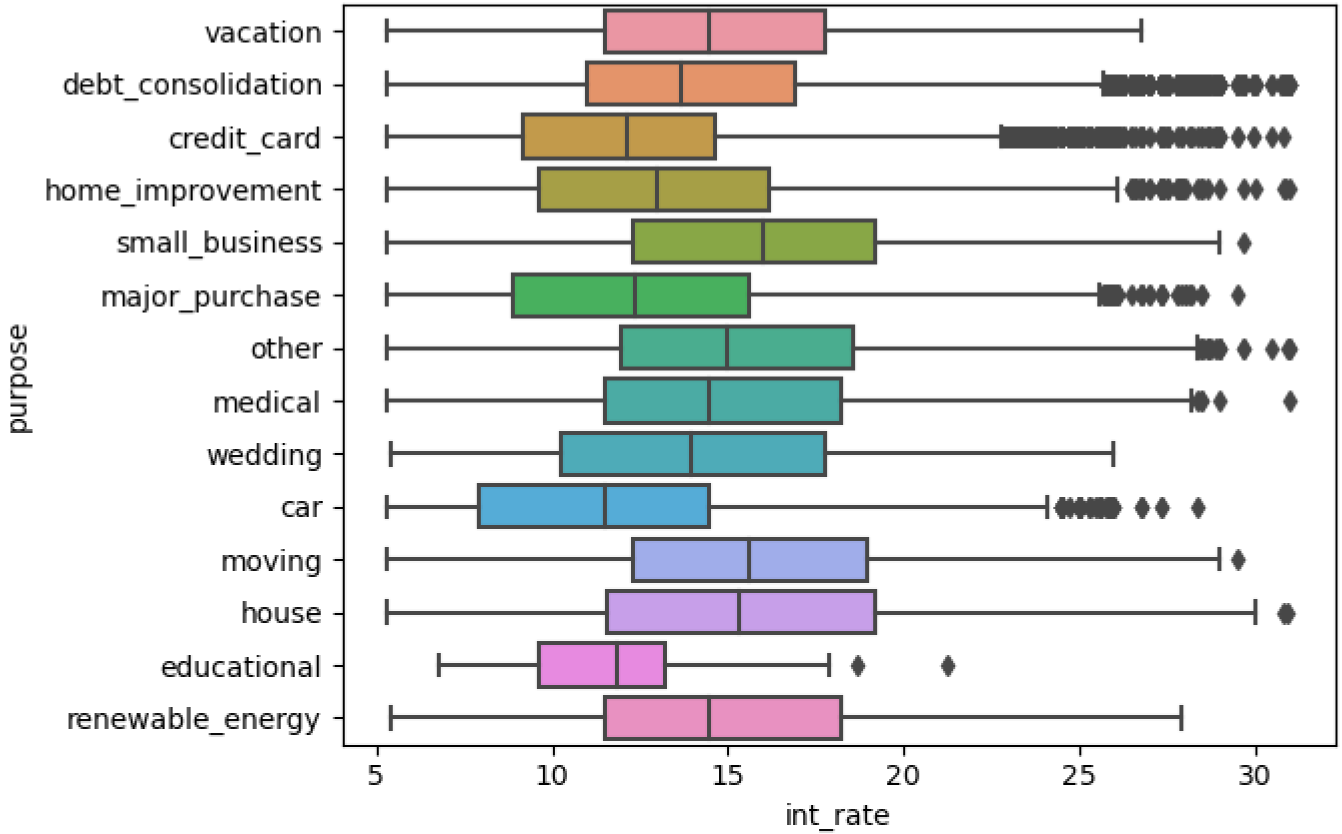
Out[35]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='purpose'>
```



Debt consildation and housing loans are some of the most common reasons for taking loans.

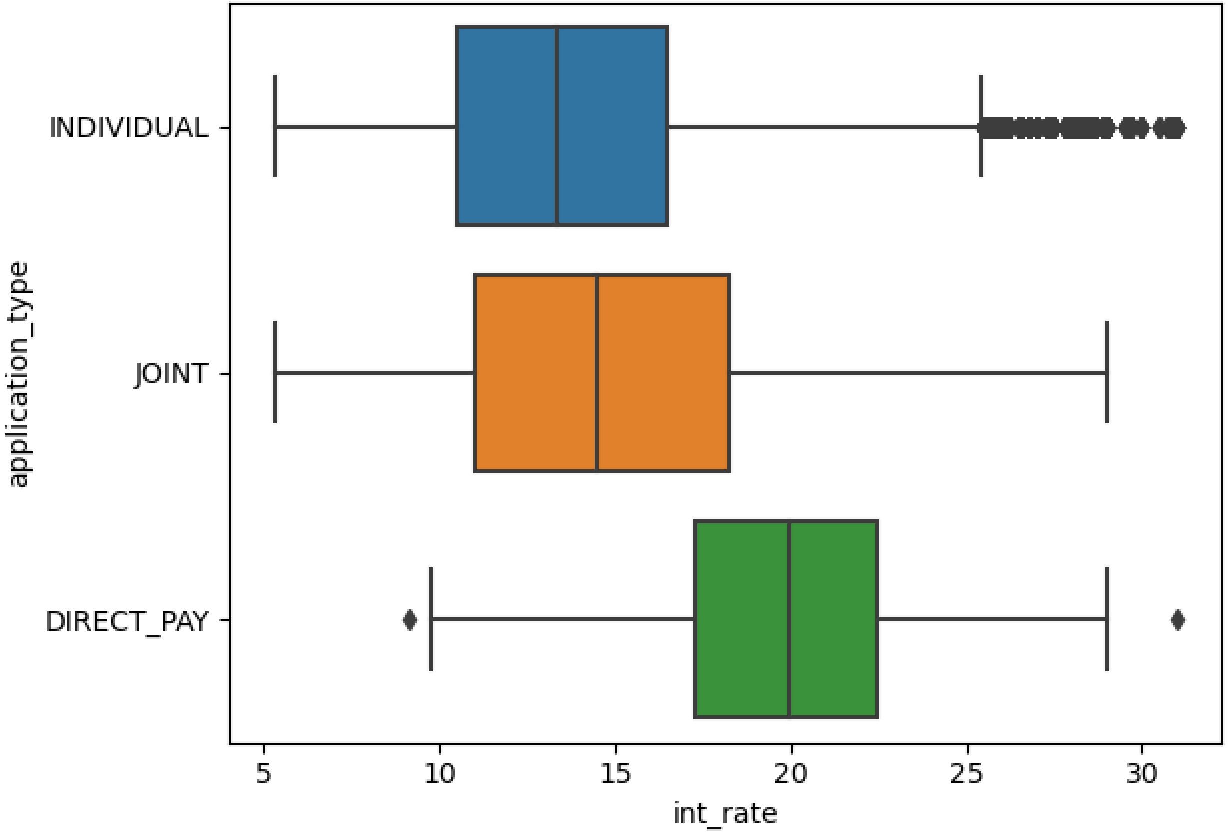
In [36]:
```
sns.boxplot(data = df, x = df['int_rate'], y = df['purpose'], orient = 'h')
```

Out[36]:
```
<AxesSubplot:xlabel='int_rate', ylabel='purpose'>
```



In [37]:
```python
sns.boxplot(data = df, x = df['int_rate'], y = df['application_type'], orient = 'h')
```

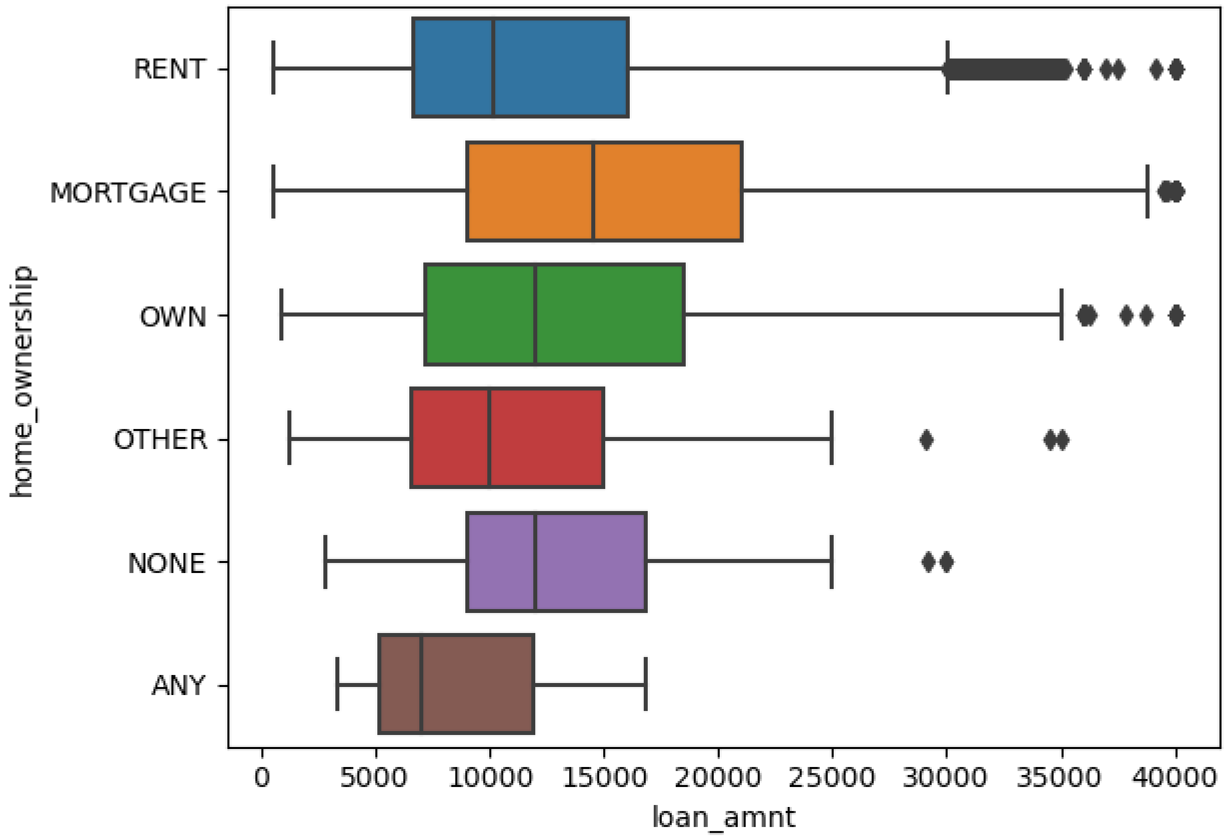
Out[37]: <AxesSubplot:xlabel='int_rate', ylabel='application_type'>



Direct pay applicants are charged higher amount of interest rate.

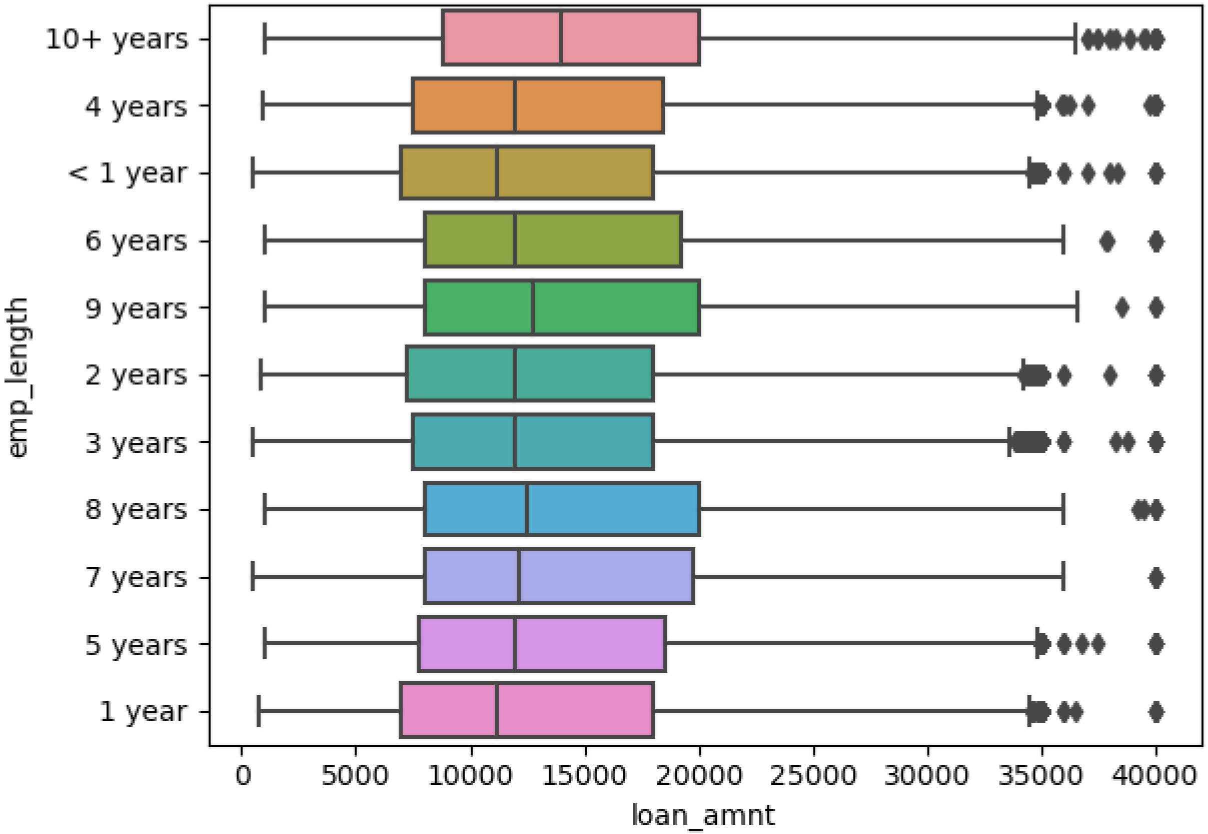
In [38]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['home_ownership'], orient = 'h')
```

Out[38]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='home_ownership'>
```



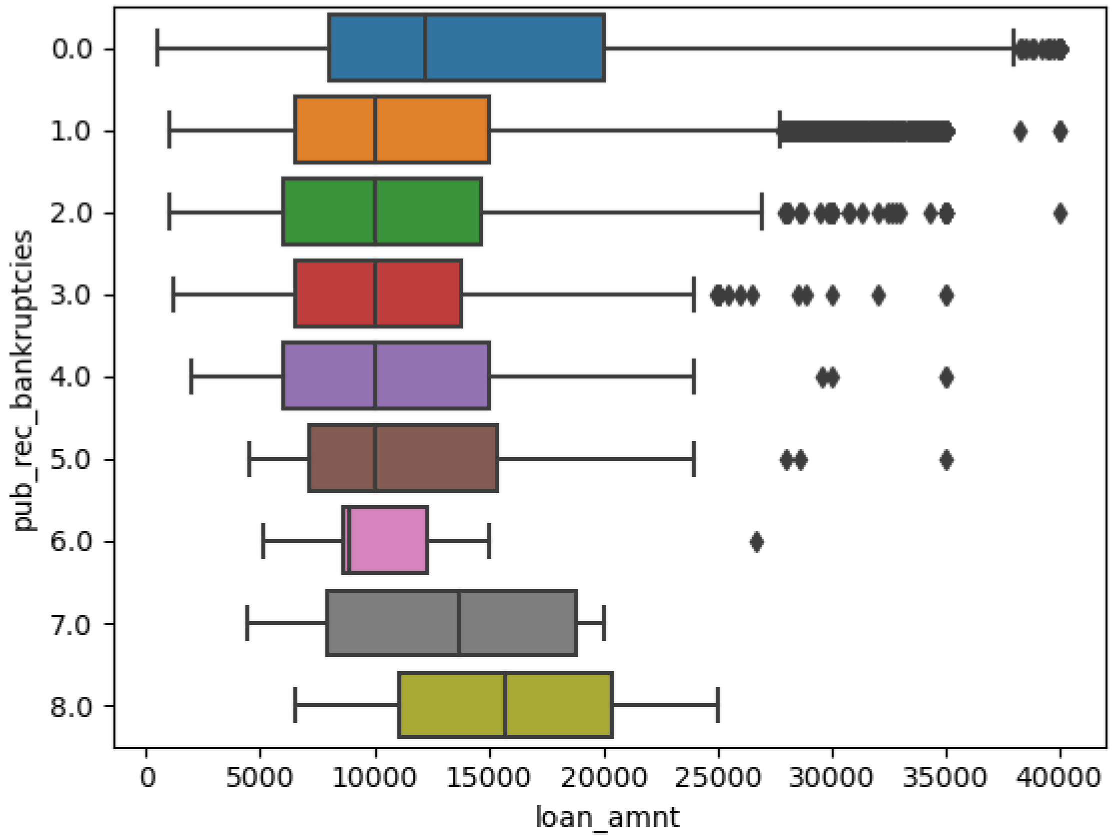
In [39]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['emp_length'], orient = 'h')
```

Out[39]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='emp_length'>
```



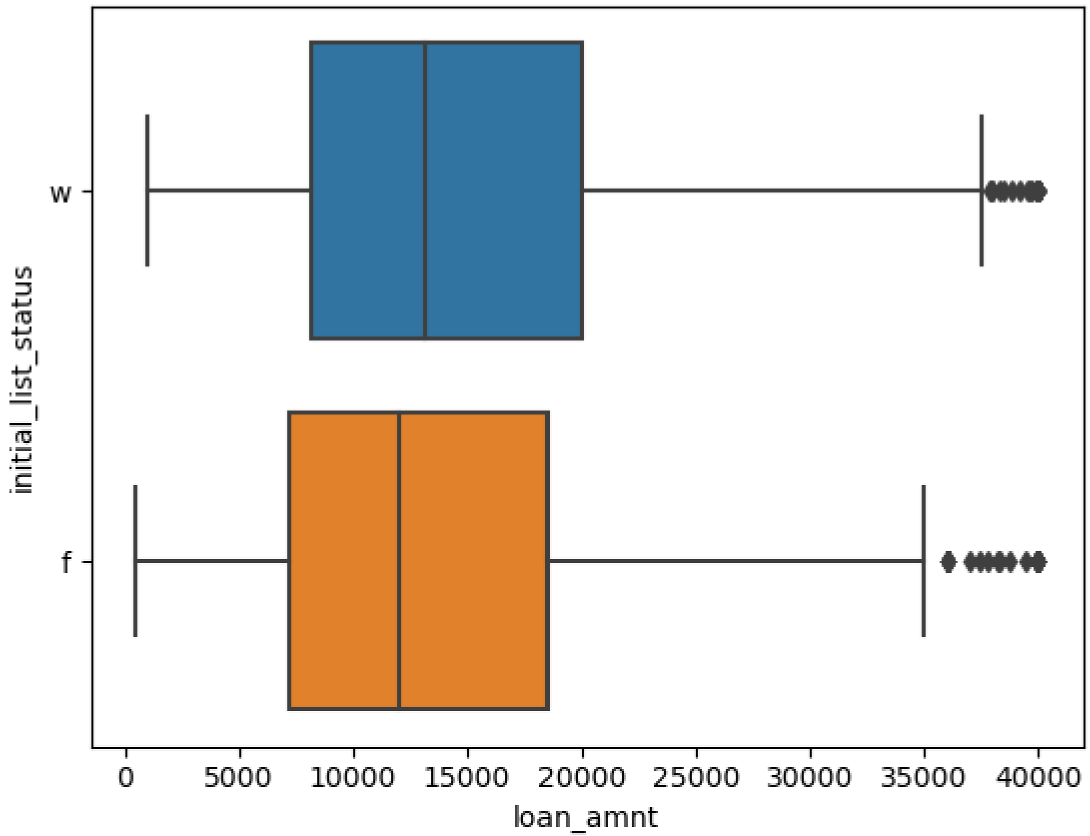
In [40]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['pub_rec_bankruptcies'], orient = 'h')
```

Out[40]: <AxesSubplot:xlabel='loan_amnt', ylabel='pub_rec_bankruptcies'>



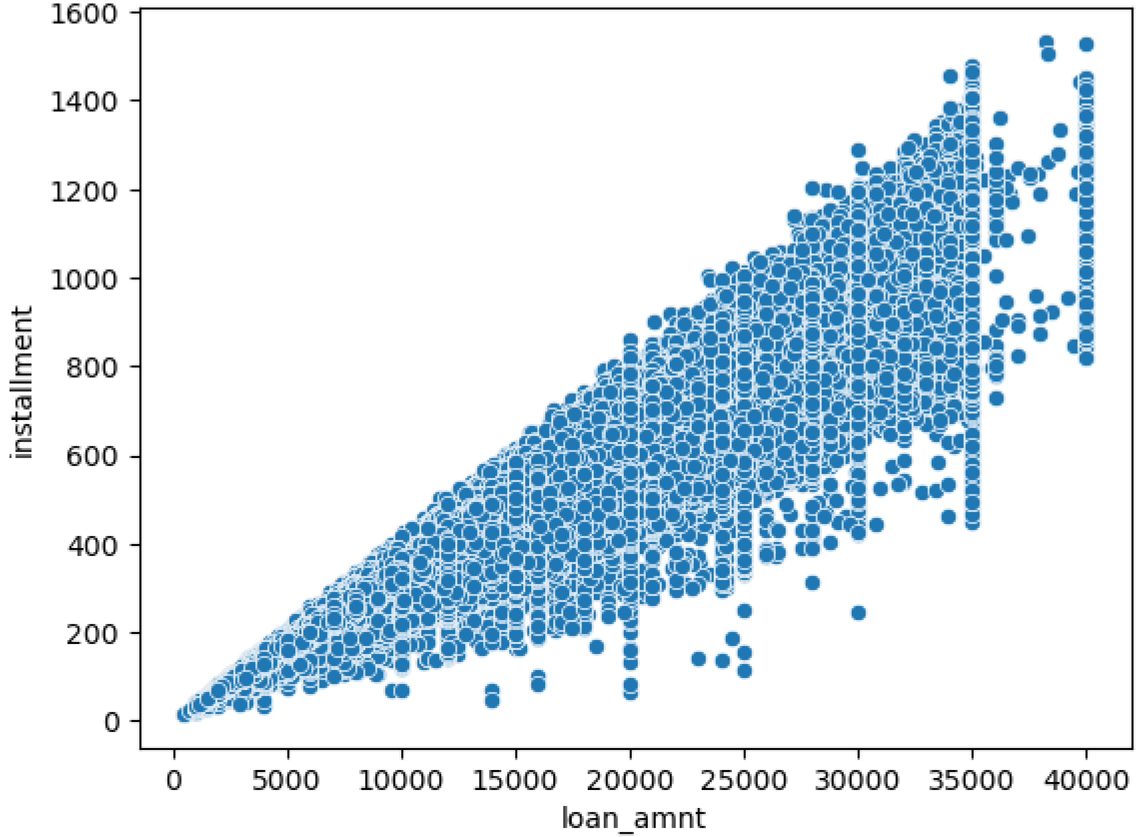
In [41]:
```
sns.boxplot(data = df, x = df['loan_amnt'], y = df['initial_list_status'], orient = 'h')
```

Out[41]: <AxesSubplot:xlabel='loan_amnt', ylabel='initial_list_status'>



In [42]:
```
sns.scatterplot(data = df, x = df['loan_amnt'], y = df['installment'])
```

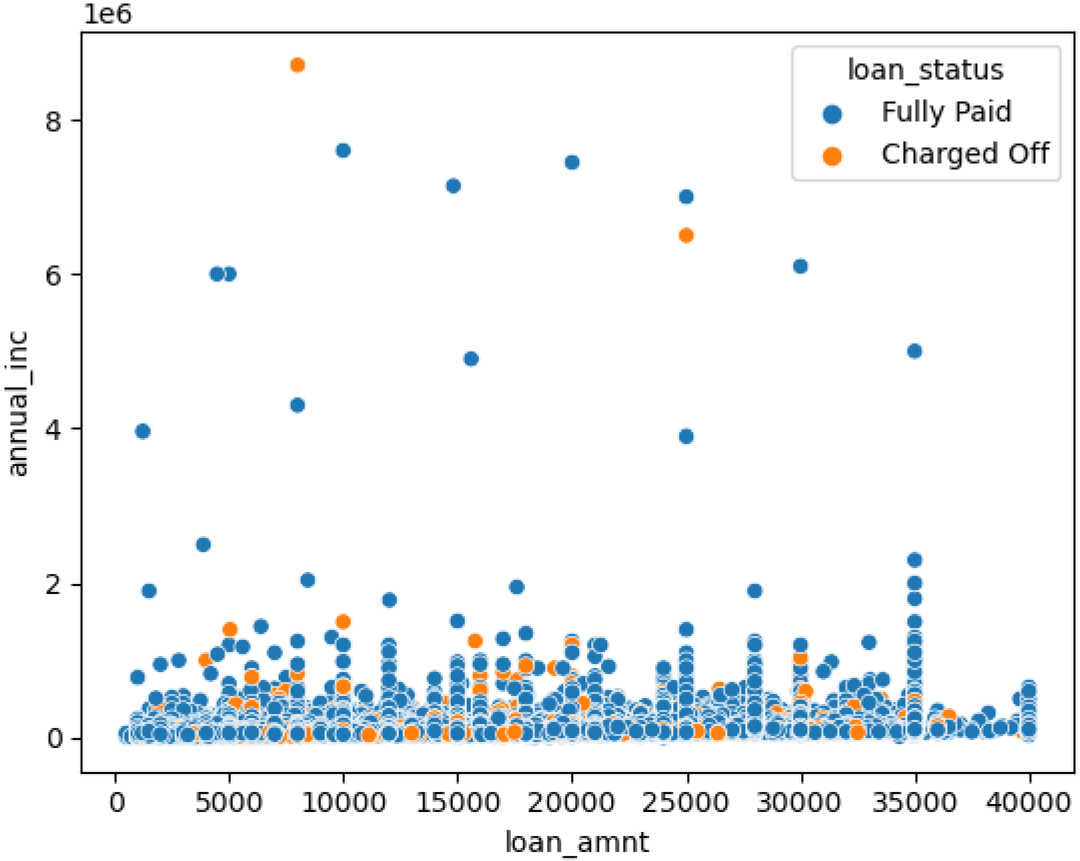
Out[42]: <AxesSubplot:xlabel='loan_amnt', ylabel='installment'>



There is a direct relationship between loan amount and installment.

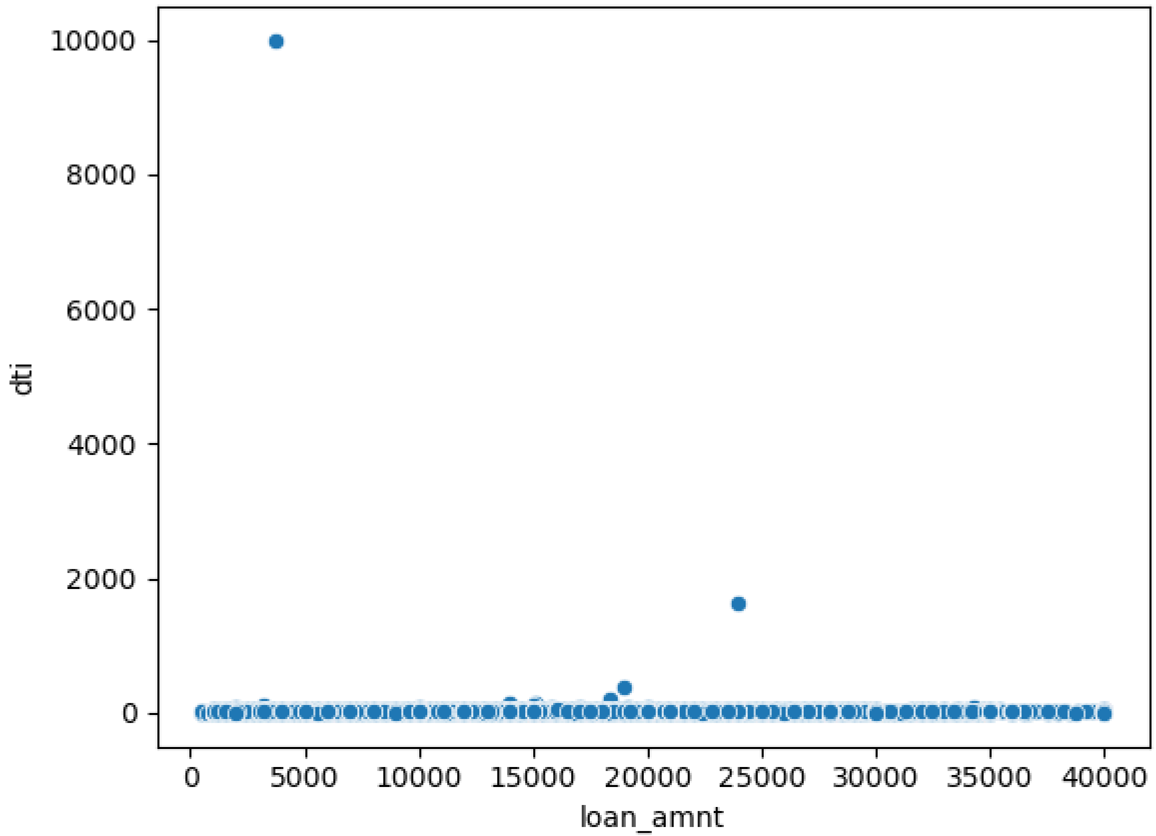
In [43]:
```python
sns.scatterplot(data = df, x = df['loan_amnt'], y = df['annual_inc'], hue = 'loan_status')
```

Out[43]: <AxesSubplot:xlabel='loan_amnt', ylabel='annual_inc'>



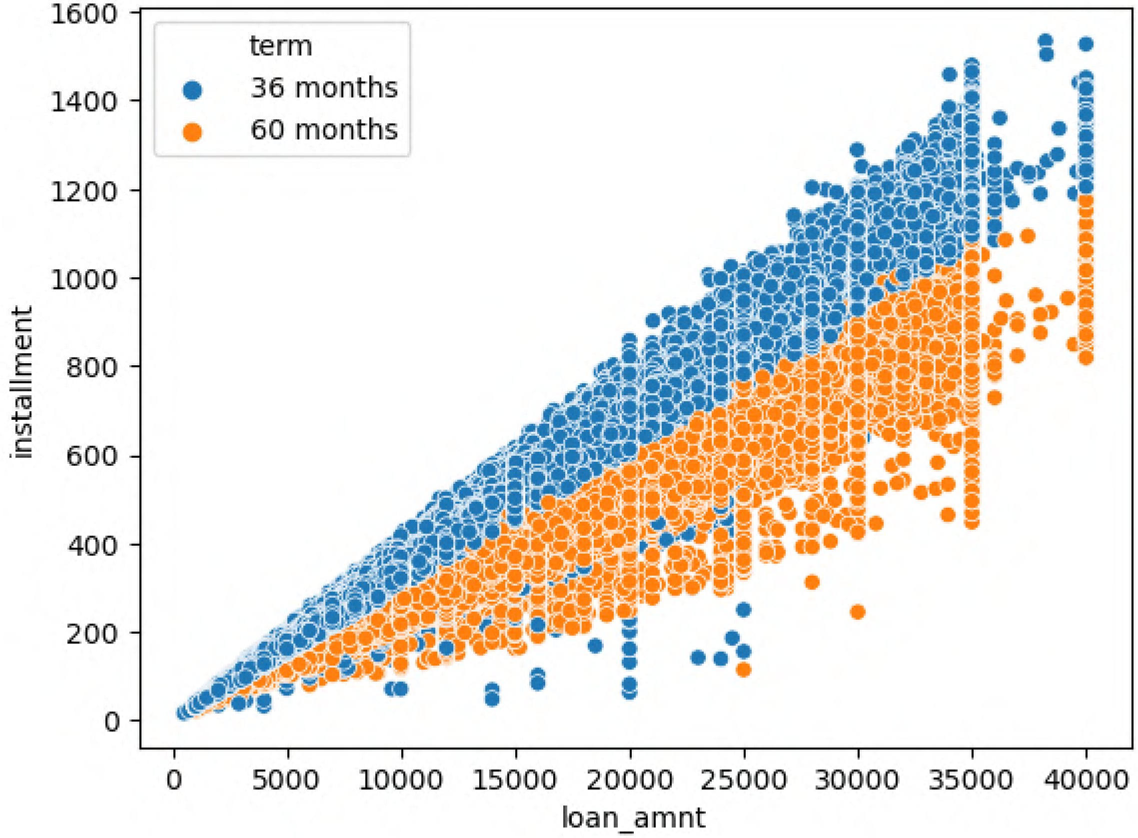
In [44]:
```
sns.scatterplot(data = df, x = df['loan_amnt'], y = df['dti'])
```

Out[44]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='dti'>
```




```
In [45]: sns.scatterplot(data = df, x = df['loan_amnt'], y = df['installment'], hue = df['term'])
```

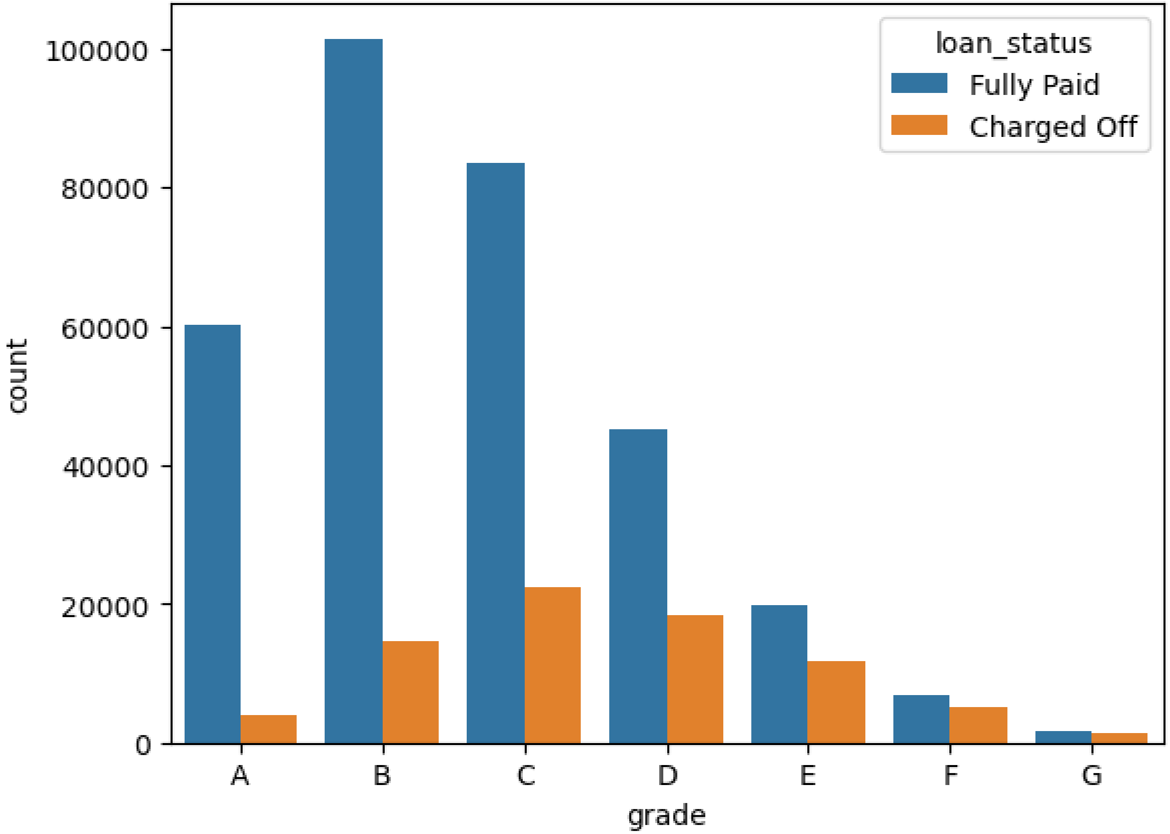
```
Out[45]: <AxesSubplot:xlabel='loan_amnt', ylabel='installment'>
```



Borrowers with lesser term of loan have higher installments.

In [46]:
```
grade = sorted(df.grade.unique().tolist())
sns.countplot(data = df, x = 'grade', hue = 'loan_status', order = grade)
```

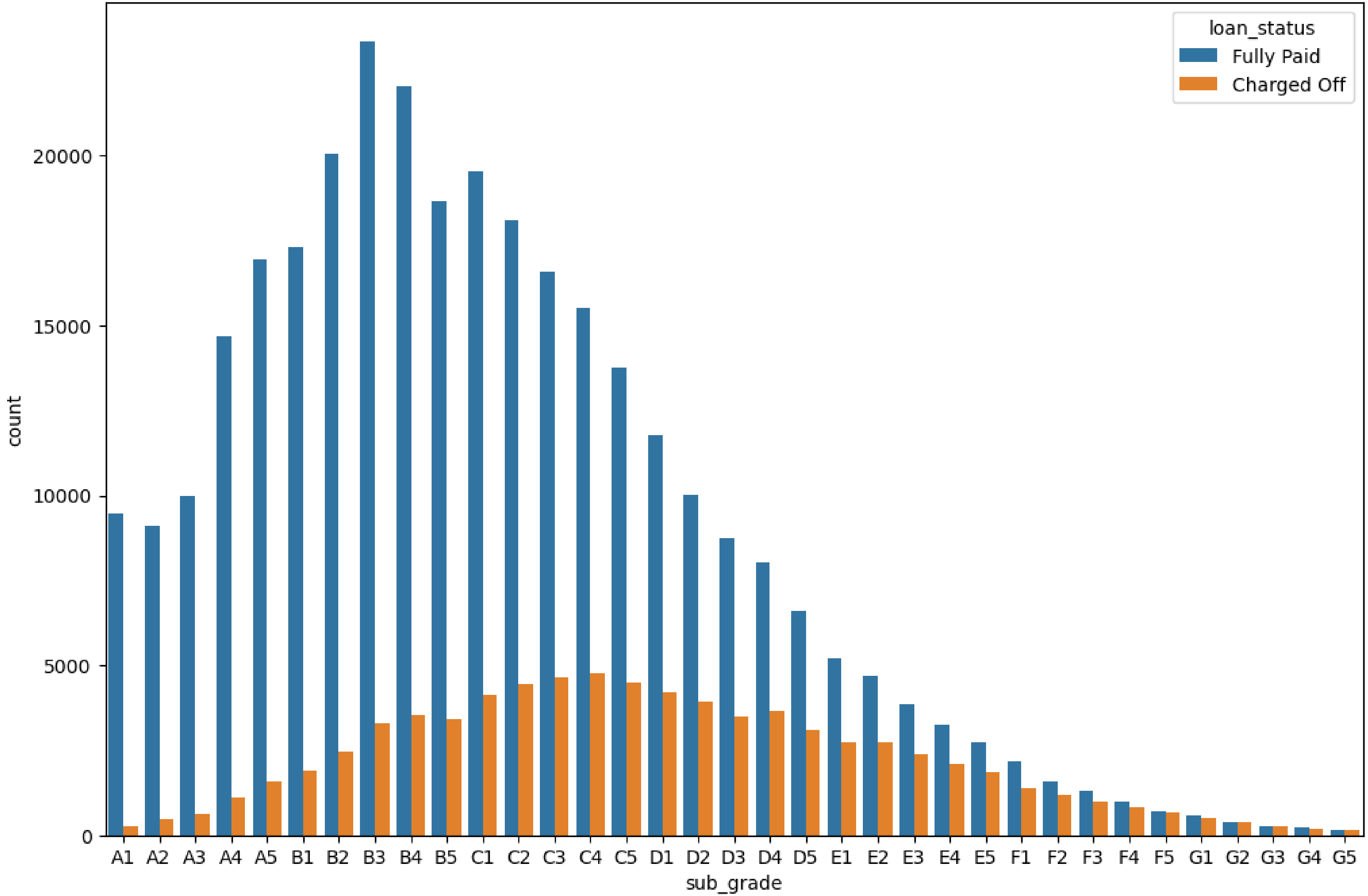
Out[46]:
```
<AxesSubplot:xlabel='grade', ylabel='count'>
```



Maximum number of people in Grade B category pay off their loans. People in GradeE, F and G are less likely to pay off their loans.

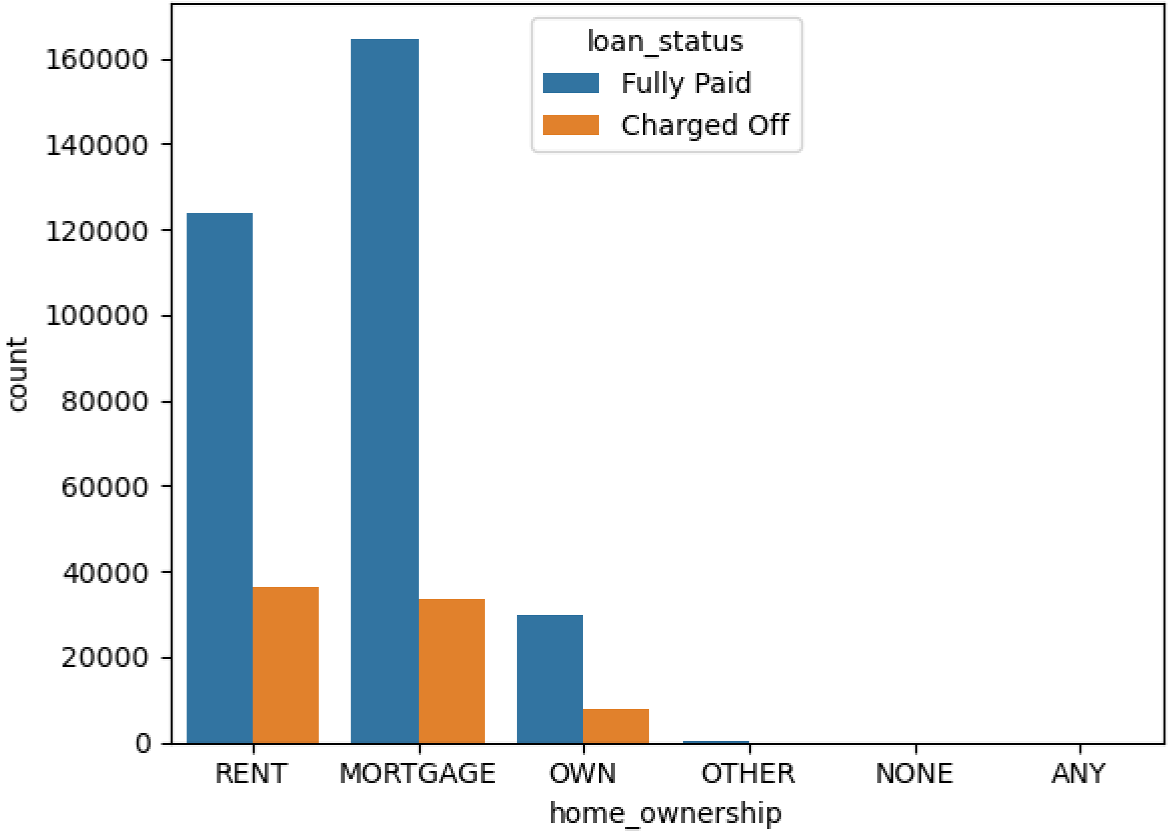
In [47]:
```
plt.figure(figsize=(12, 8))
sub_grade = sorted(df.sub_grade.unique().tolist())
sns.countplot(data = df, x = 'sub_grade', hue = 'loan_status', order = sub_grade)
```

Out[47]: <AxesSubplot:xlabel='sub_grade', ylabel='count'>



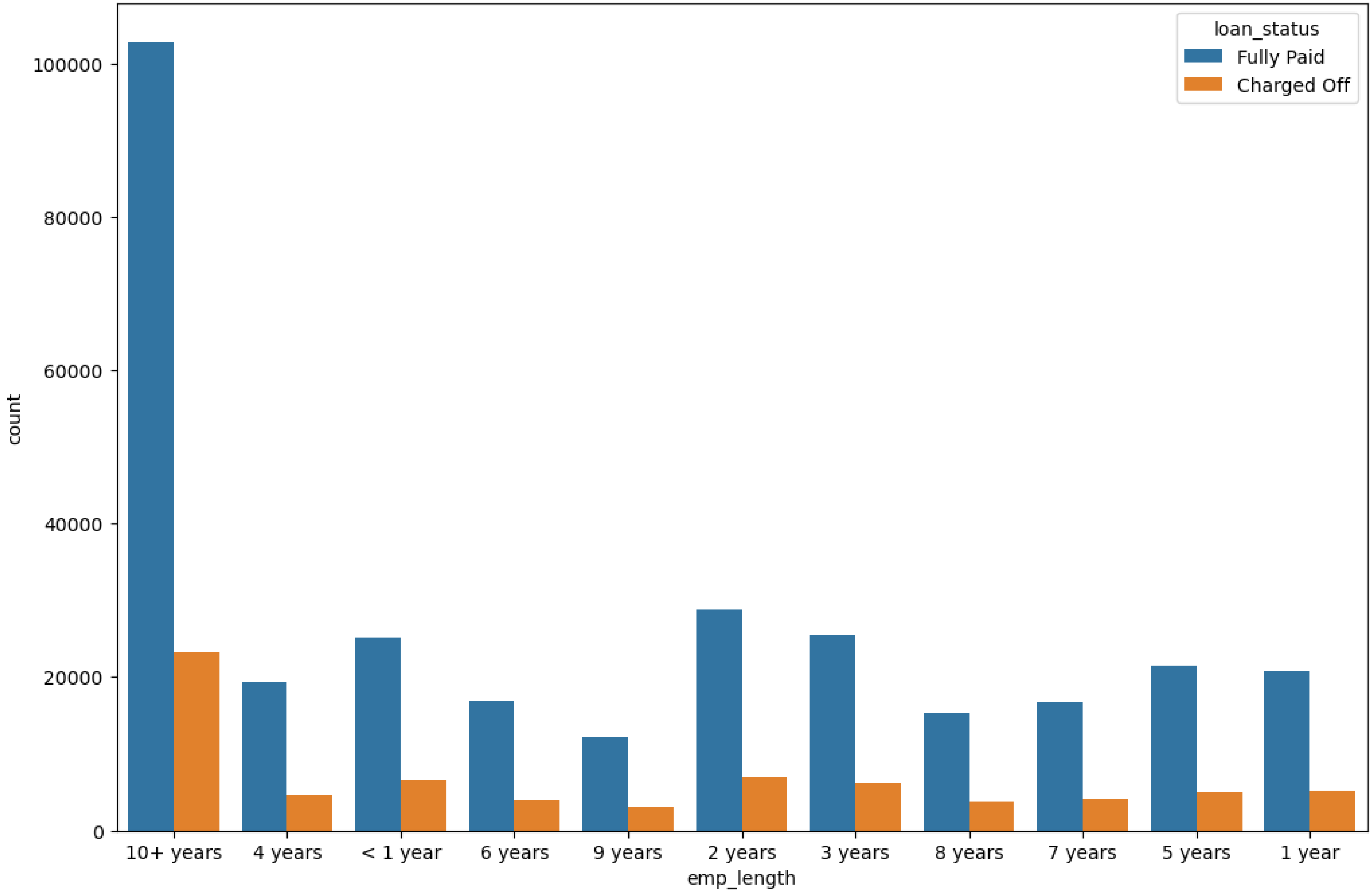
In [48]:
```
sns.countplot(data = df, x = df['home_ownership'], hue = df['loan_status'])
```

Out[48]:
```
<AxesSubplot:xlabel='home_ownership', ylabel='count'>
```



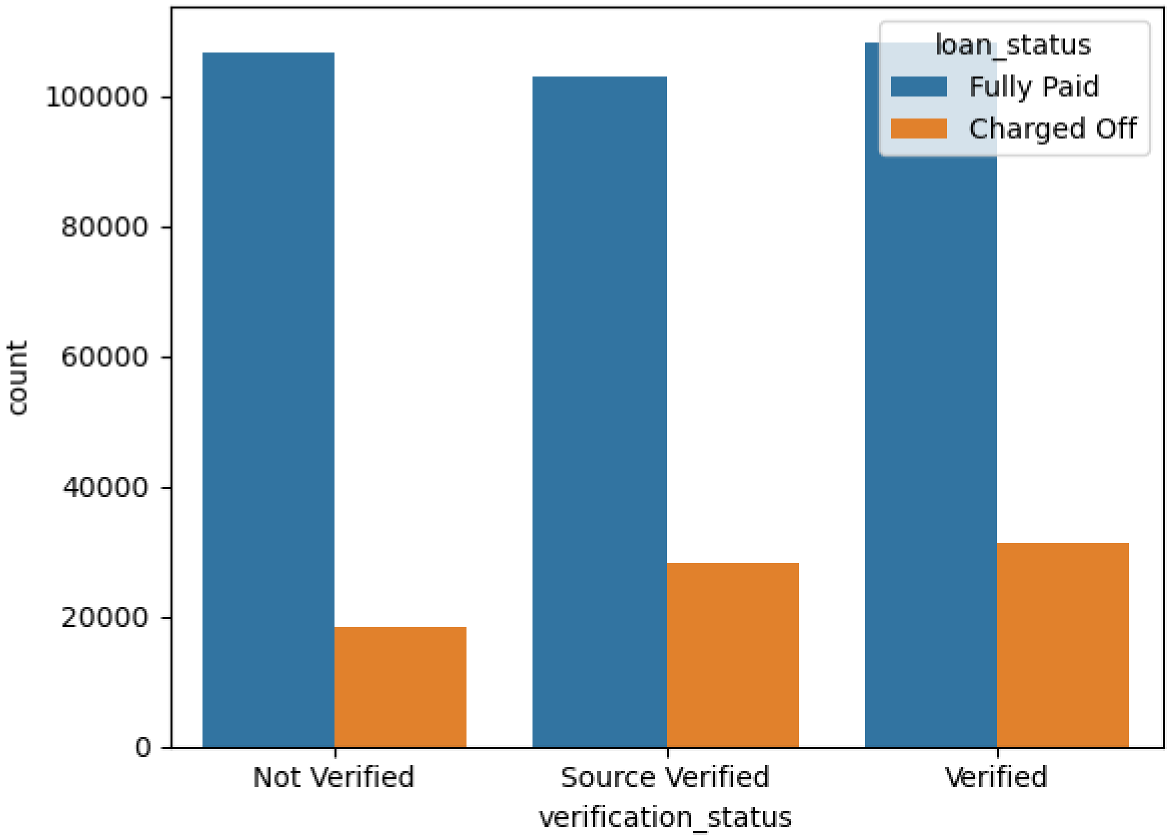
In [49]:
```
plt.figure(figsize=(12, 8))
sns.countplot(data = df, x = df['emp_length'], hue = df['loan_status'])
```

Out[49]:
```
<AxesSubplot:xlabel='emp_length', ylabel='count'>
```



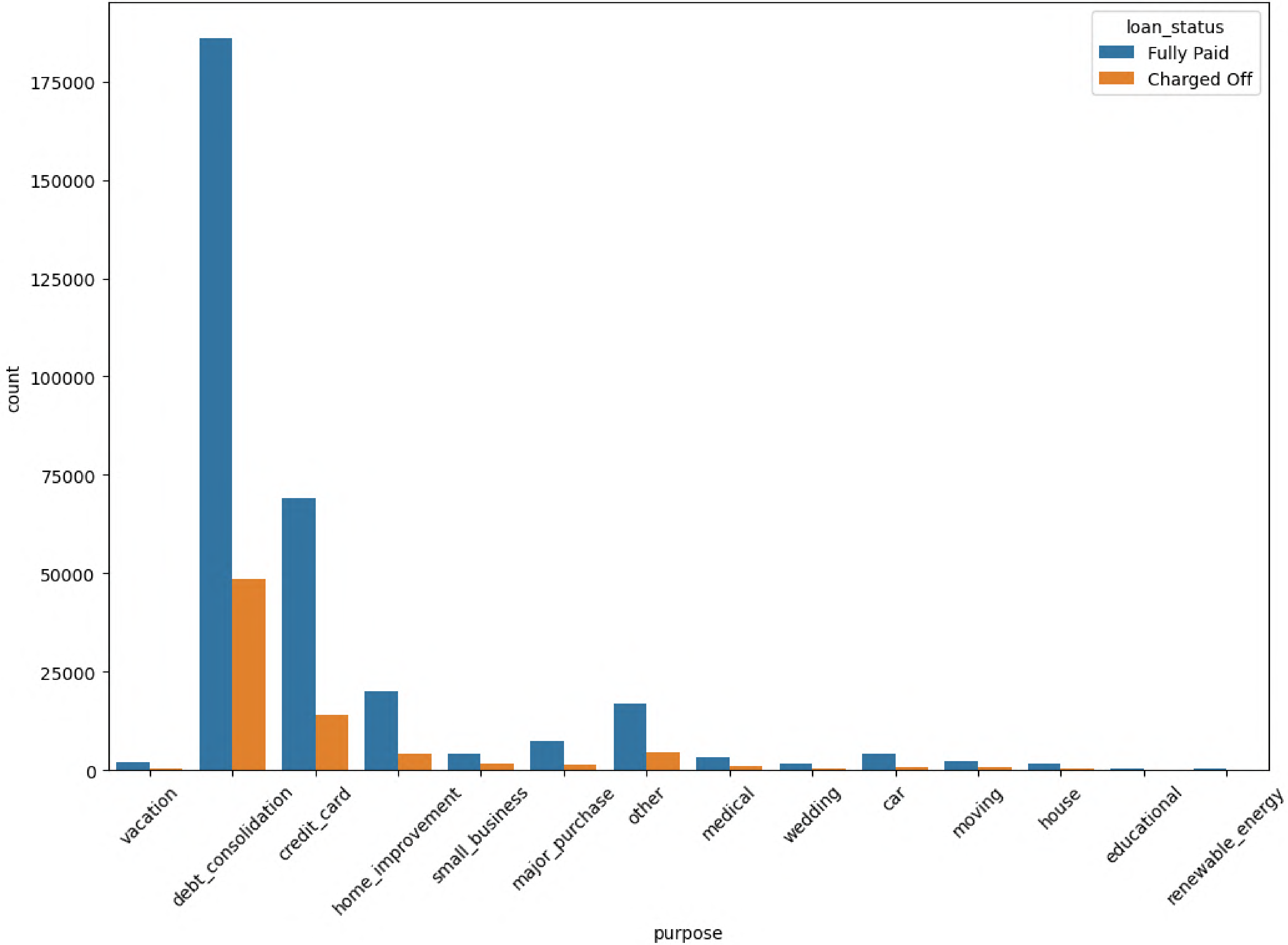
In [50]:
```
sns.countplot(data = df, x = df['verification_status'], hue = df['loan_status'])
```

Out[50]:
```
<AxesSubplot:xlabel='verification_status', ylabel='count'>
```



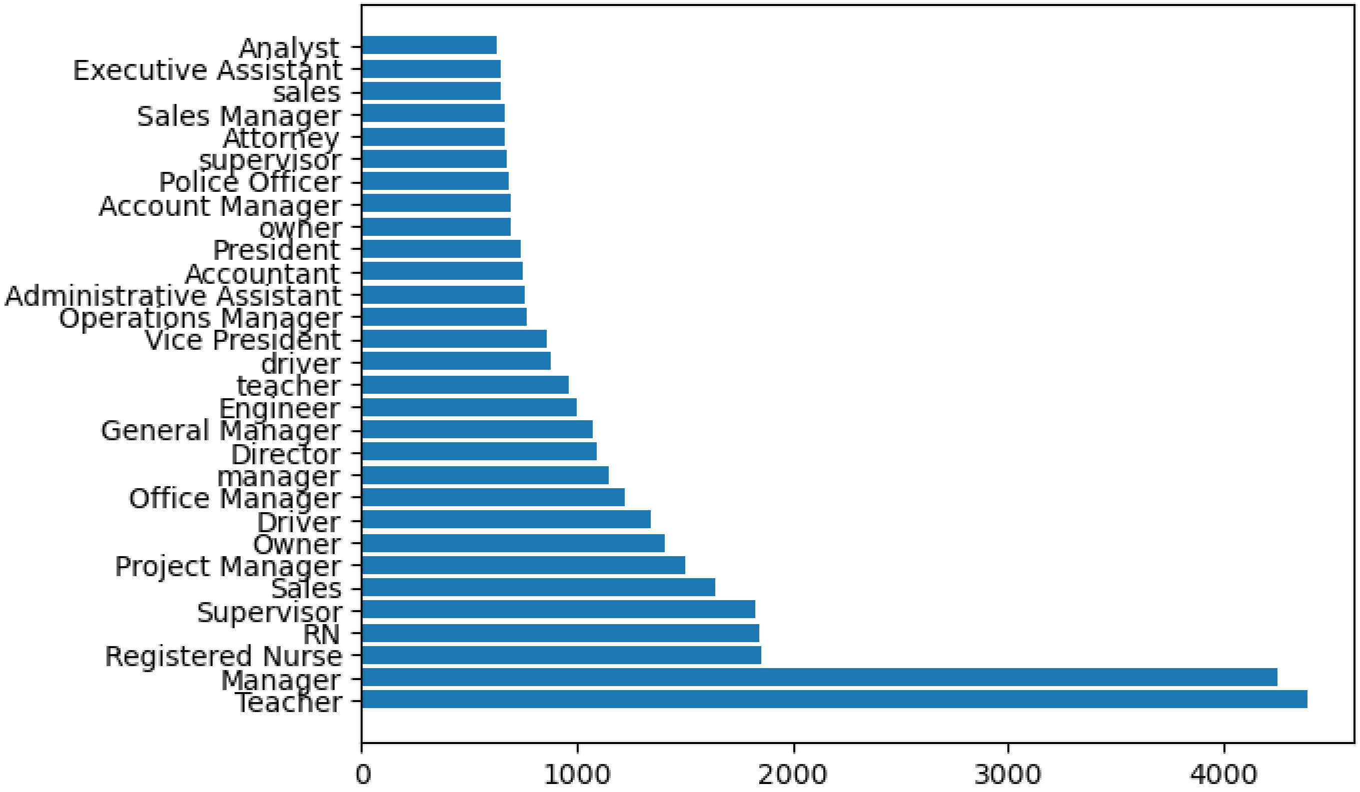
Verified borrowers are more likely to pay loans.

In [51]:
```
plt.figure(figsize=(12, 8))
sns.countplot(data = df, x = df['purpose'], hue = df['loan_status'])
plt.xticks (rotation = 45)
plt.show()
```


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

In [52]:
```
plt.barh(df.emp_title.value_counts()[:30].index, df.emp_title.value_counts()[:30])
```

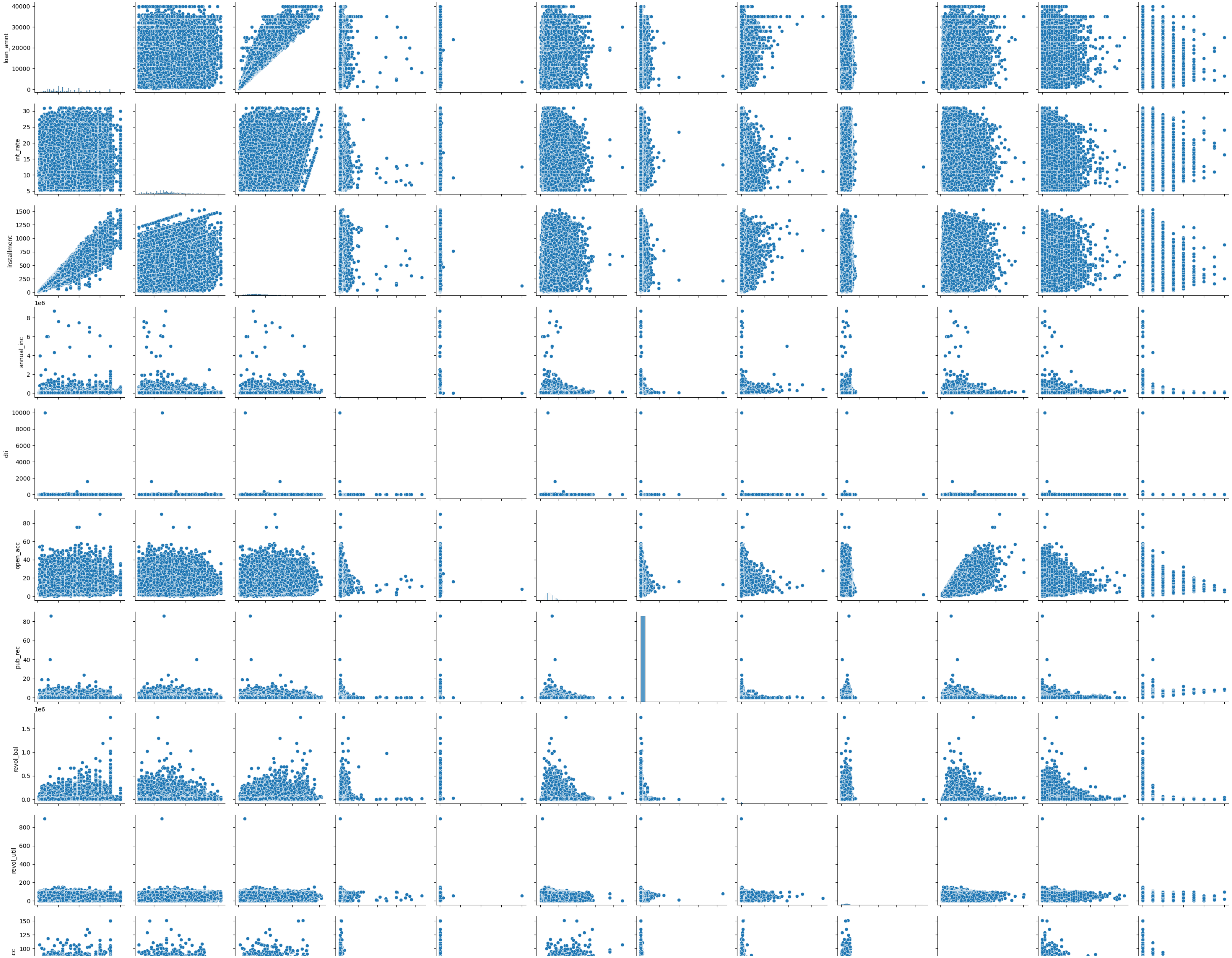
Out[52]:
```
<BarContainer object of 30 artists>
```

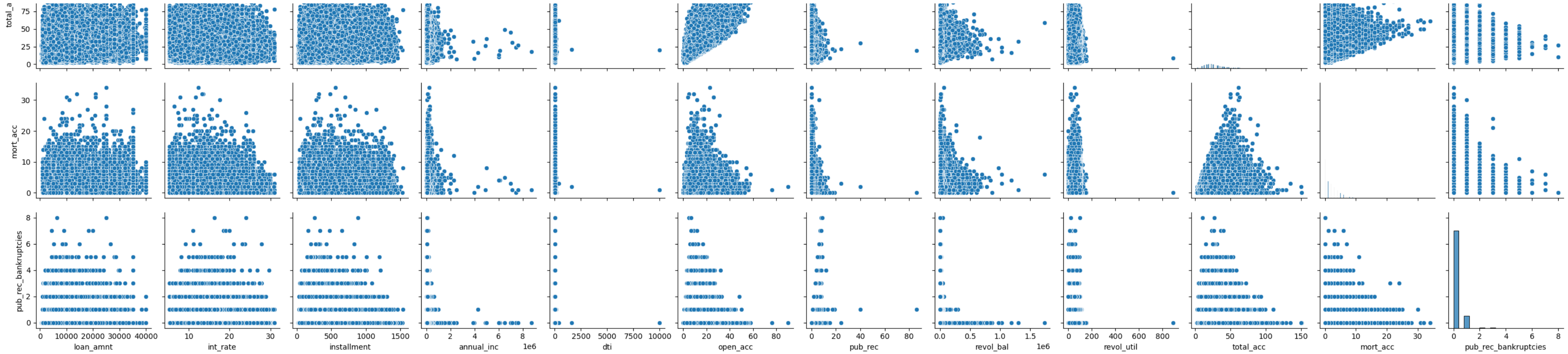


Teachers and Managers are more likely to afford loans.

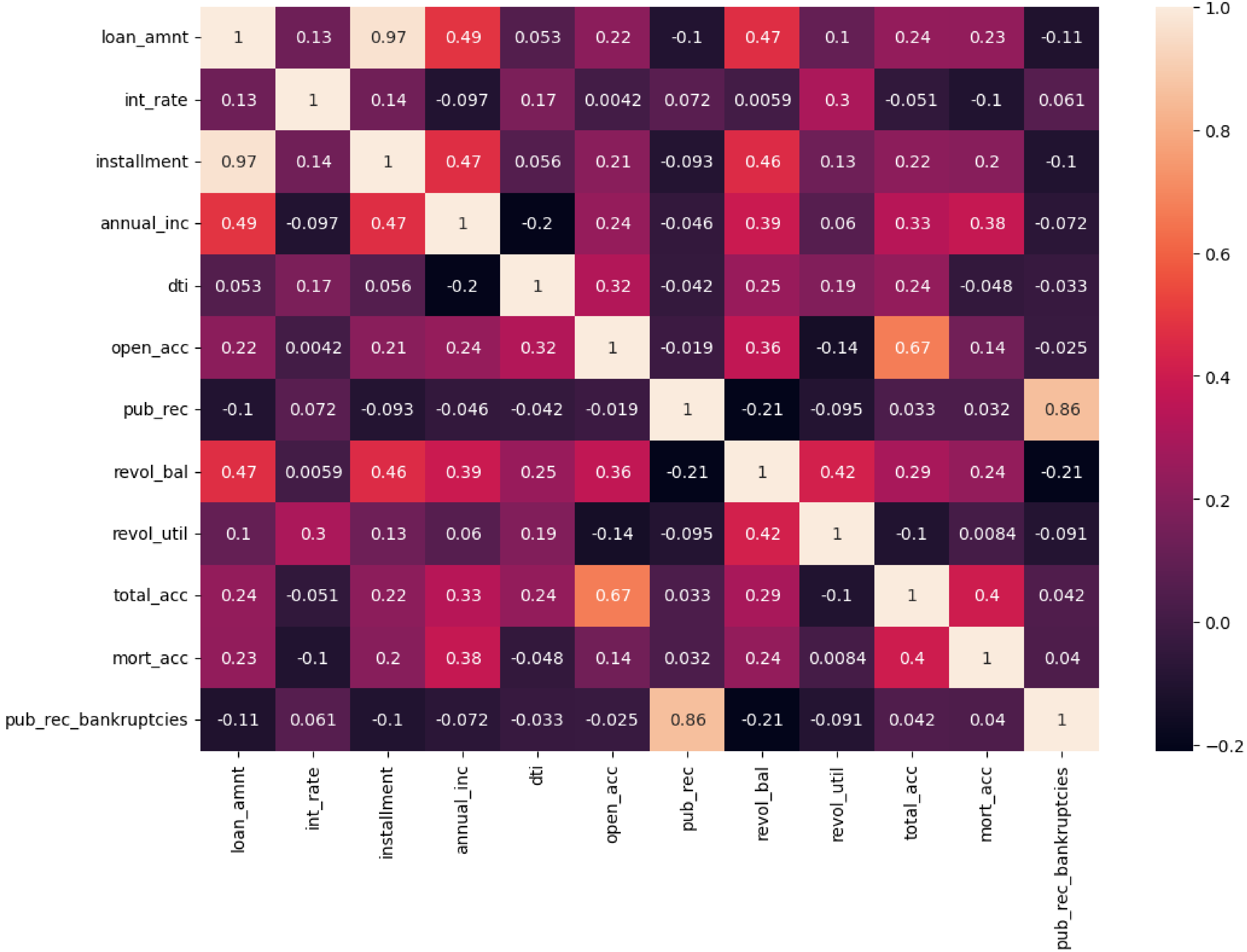
In [53]:
```
sns.pairplot(df)
```

Out[53]:
```
<seaborn.axisgrid.PairGrid at 0x2a9517a4af0>
```


loan_amnt
int_rate
installment
annual_inc
dti
open_acc
pub_rec
revol_bal
revol_util
cc



In [54]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(method='spearman'), annot=True)
plt.show()
```


loan_amnt
int_rate
installment
annual_inc
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
−0.2

Loan amount and installment amount are highly correlated.

In [55]:
```
df.groupby(by ='loan_status')['loan_amnt'].describe()
```

Out[55]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

Here, we can observe that if the loan amount is high, chances of default are higher.

Duplicate Values, Missing Values and Outlier Treatment

In [56]:
```
df.duplicated().sum()
```

Out[56]:
```
0
```

There are no duplicate values in the data.

In [57]:
```
def missing_df(data):
    total_missing_df = data.isna().sum().sort_values(ascending = False)
    percentage_missing_df = ((data.isna().sum()/len(data)*100)).sort_values(ascending = False)
    missingDF = pd.concat([total_missing_df, percentage_missing_df],axis = 1, keys=['Total', 'Percent'])
    return missingDF


missing_data = missing_df(df)
missing_data[missing_data["Total"]>0]
```

Out[57]:

	Total	Percent
mort_acc	37795	9.543469
emp_title	22927	5.789208
emp_length	18301	4.621115
title	1755	0.443148
pub_rec_bankruptcies	535	0.135091
revol_util	276	0.069692

In [58]:

```
from sklearn.impute import SimpleImputer
Imputer = SimpleImputer(strategy="most_frequent")
df["mort_acc"] = Imputer.fit_transform(df["mort_acc"].values.reshape(-1,1))
```

```
C:\Users\Home\anaconda3\lib\site-packages\sklearn\impute\_base.py:49: FutureWarning: Unlike other reduction functions (e.g. `ske
w`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and
the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode = stats.mode(array)
```

In [59]:

```
df.dropna(inplace=True)
```

In [60]:

```
missing_df(df)
```

Out[60]:

	Total	Percent
loan_amnt	0	0.0
title	0	0.0
pub_rec_bankruptcies	0	0.0
mort_acc	0	0.0
application_type	0	0.0
initial_list_status	0	0.0
total_acc	0	0.0
revol_util	0	0.0
revol_bal	0	0.0
pub_rec	0	0.0
open_acc	0	0.0
earliest_cr_line	0	0.0
dti	0	0.0
purpose	0	0.0
term	0	0.0
loan_status	0	0.0
issue_d	0	0.0
verification_status	0	0.0
annual_inc	0	0.0
home_ownership	0	0.0
emp_length	0	0.0
emp_title	0	0.0
sub_grade	0	0.0
grade	0	0.0

	Total	Percent
installment	0	0.0
int_rate	0	0.0
address	0	0.0

In [61]:
```
df.shape
```
Out[61]:
```
(370622, 27)
```

In [62]:
```
numerical_data = df.select_dtypes(include='number')
num_cols = numerical_data.columns
len(num_cols)
```
Out[62]:
```
12
```

In [63]:
```
for col in num_cols:
    mean = df[col].mean()
    std = df[col].std()

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

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

df.shape
```
Out[63]:
```
(338364, 27)
```

Feature Engineering

In [64]:
```
def pub_rec(num):
    if num == 0.0:
        return 0
    else:
        return 1

def mort_acc(num):
```

```
    if num == 0.0:
        return 0
    elif num >= 1.0:
        return 1
    else:
        return num

def pub_rec_bankruptcies(num):
    if num == 0.0:
        return 0
    elif num >= 1.0:
        return 1
    else:
        return num
```

In [65]:
```
df['pub_rec'] = df.pub_rec.apply(pub_rec)
df['mort_acc'] = df.mort_acc.apply(mort_acc)
df['pub_rec_bankruptcies'] = df.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

Data Preprocessing

In [66]:
```
#Convert employment length to numeric
d = {'10+ years':10, '4 years':4, '< 1 year':0,
     '6 years':6, '9 years':9,'2 years':2, '3 years':3,
     '8 years':8, '7 years':7, '5 years':5, '1 year':1}
df['emp_length']=df['emp_length'].replace(d)
```

In [67]:
```
#Convert term to numeric
t = {' 36 months' : 36, ' 60 months' : 60}
df['term'] = df['term'].replace(t)
```

In [68]:
```
#Converting initial listing status to numeric
ils = {'w' : 0, 'f' : 1}
df['initial_list_status'] = df['initial_list_status'].replace(ils)
```

In [69]:
```
df['zip_code'] = df.address.apply(lambda x: x[-5:])
df['zip_code'].value_counts(normalize=True)*100
```

Out[69]:
```
70466    14.365299
30723    14.290823
22690    14.254767
48052    14.142462
00813    11.603480
29597    11.532551
05113    11.525458
93700     2.772163
11650     2.769207
86630     2.743791
Name: zip_code, dtype: float64
```

In [70]:
```
df.drop(columns=['address', 'issue_d', 'emp_title', 'emp_length', 'title', 'sub_grade', 'earliest_cr_line'], axis = 1, inplace =
```

In [71]:
```
# Encoding Target Variable

df['loan_status']=df['loan_status'].map({'Fully Paid': 0, 'Charged Off':1}).astype(int)
```

In [72]:
```
dummies = ['purpose', 'zip_code', 'grade', 'verification_status', 'application_type', 'home_ownership']
df = pd.get_dummies(df, columns = dummies, drop_first = True)
```

In [73]:
```
df.head()
```

Out[73]:

	loan_amnt	term	int_rate	installment	annual_inc	loan_status	dti	open_acc	pub_rec	revol_bal	...	grade_G	verification_status_Source Verified	verificatio
0	10000.0	36	11.44	329.48	117000.0	0	26.24	16.0	0	36369.0	...	0	0	
1	8000.0	36	11.99	265.68	65000.0	0	22.05	17.0	0	20131.0	...	0	0	
2	15600.0	36	10.49	506.97	43057.0	0	12.79	13.0	0	11987.0	...	0	1	
3	7200.0	36	6.49	220.65	54000.0	0	2.60	6.0	0	5472.0	...	0	0	
4	24375.0	60	17.27	609.33	55000.0	1	33.95	13.0	0	24584.0	...	0	0	

5 rows × 52 columns

Preparing Data for Modeling

In [74]:
```
x = df.drop('loan_status', axis = 1)
y = df['loan_status']
```

In [75]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.20, stratify=y,random_state=42)
```

In [76]:
```
print(x_train.shape)
print(x_test.shape)
```

```
(270691, 51)
(67673, 51)
```

MinMaxScaler

In [77]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Logistic Regression

In [78]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(max_iter=1000)
logreg.fit(x_train, y_train)
```

Out[78]:
```
LogisticRegression(max_iter=1000)
```

In [79]:
```
y_pred = logreg.predict(x_test)
print('Accuracy : ', logreg.score(x_test, y_test))
```

```
Accuracy :  0.8906949595850635
```

In [80]:
```
from sklearn.metrics import(accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve, auc, plot_confusic
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Confusion Matrix

In [81]:
```
confusion_matrix = confusion_matrix(y_test, y_pred)
confusion_matrix
```

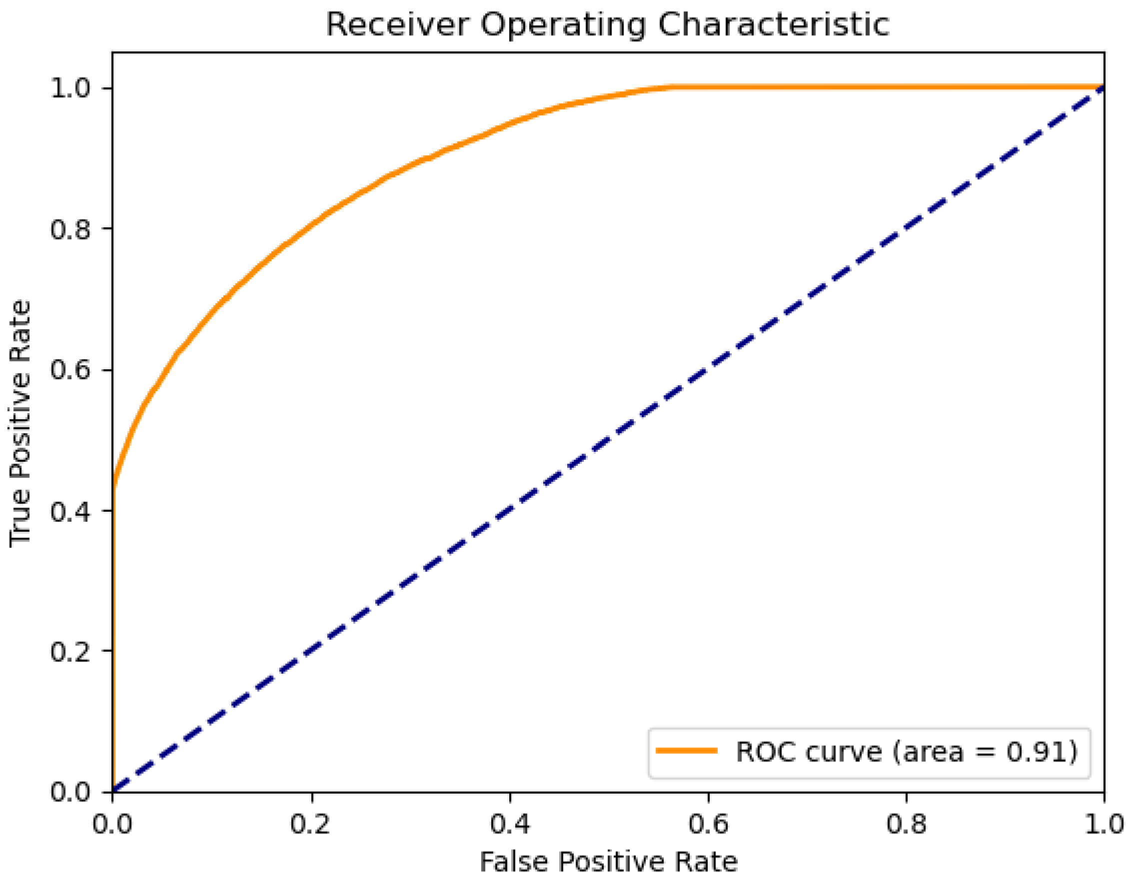
Out[81]:
```
array([[54322,   327],
       [ 7070,  5954]], dtype=int64)
```

Classification Report

In [82]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     54649
           1       0.95      0.46      0.62     13024

    accuracy                           0.89     67673
   macro avg       0.92      0.73      0.78     67673
weighted avg       0.90      0.89      0.87     67673
```

In [83]:
```
# Predict probabilities for the test set
probs = logreg.predict_proba(x_test)[:,1]

# Compute the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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.show()
```



- AUC of 0.91 signifies that the model is able to discriminate well between the positive and the negative class.
- But it is not a good measure for an imbalanced target variable because it may be high even when the classifier has a poor score on the minority class.
- This can happen when the classifier performs well on the majority class instances, which dominate the dataset. As a result, the AUC may appear high, but the model may not effectively identify the minority class instances.

Let's plot the Precision-Recall curve which is more suited for evaluation of imbalanced data.

Precision Recall Curve

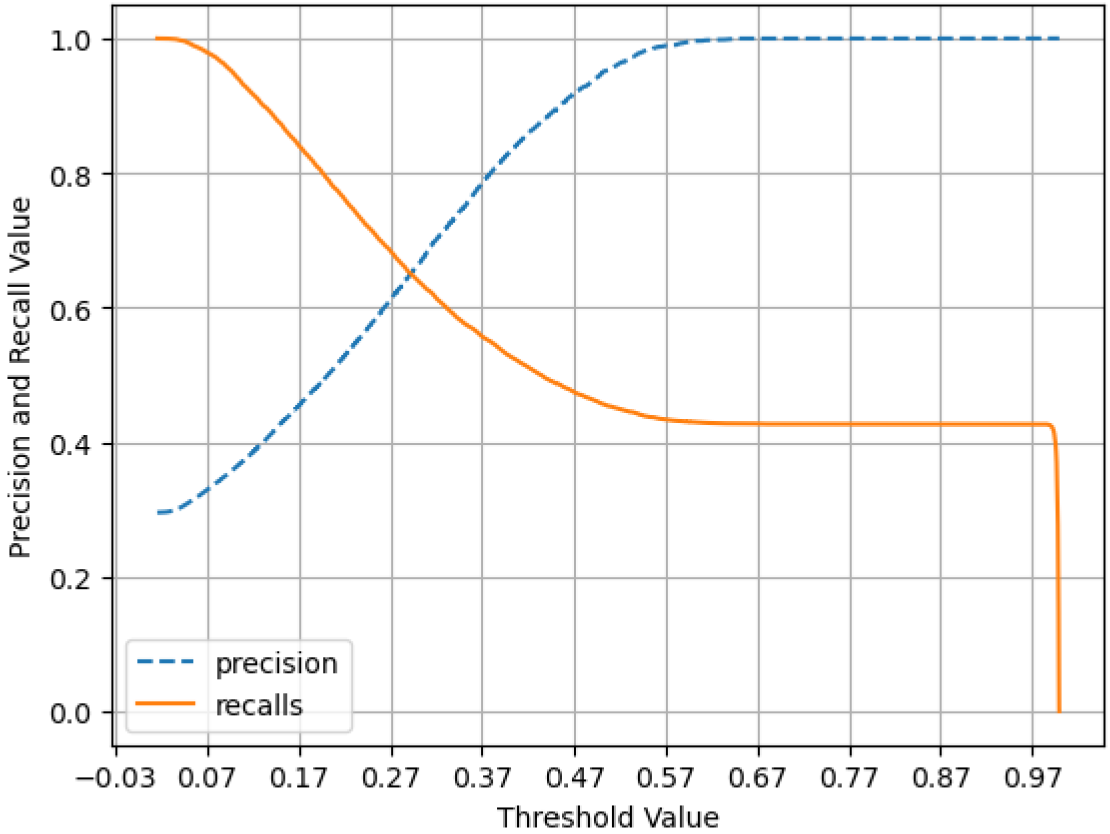
The Precision-Recall (PR) curve is another graphical representation commonly used to evaluate the performance of a binary classification model. It provides insights into the trade-off between precision and recall at various classification thresholds.

Precision represents the proportion of correctly predicted positive instances out of all instances predicted as positive. It focuses on the accuracy of positive predictions. Recall, also known as sensitivity or true positive rate, represents the proportion of correctly predicted positive instances out of all actual positive instances. It focuses on capturing all positive instances. Similar to the ROC curve, the PR curve is created by plotting recall on the x-axis and precision on the y-axis for different threshold values. The curve illustrates the relationship between precision and recall as the classification threshold changes.

A perfect classifier would have a precision of 1 and a recall of 1, resulting in a point at the top-right corner of the PR curve. Conversely, a random classifier would have a PR curve following the horizontal line defined by the ratio of positive instances in the dataset.

In [84]:

```
from sklearn.metrics import precision_recall_curve
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])
```



As expected, the area under precision recall curve is not as high. It is a decent model as the area is more than 0.5 (random model benchmark) but there is still scope for improvement.

Multicollinearity Check

In [85]:
```
vifs = []

for i in range(x_train.shape[1]):

    vifs.append((variance_inflation_factor(exog = x_train,
                                           exog_idx=i)))
vifs
```

Out[85]:
```
[225.4794708227029,
 9.425361899237933,
 55.71867162755407,
 202.74361738922,
 7.602809015958986,
 8.154766202362806,
 11.820653191919003,
 5.532694205519837,
 4.776964766738563,
 9.56658141434967,
 10.68016905908239,
 2.7366909622351403,
 3.7700697289049514,
 5.359027432094841,
 18.541837126589648,
 50.90445337926411,
 1.053010718789296,
 5.740343200826512,
 1.4368773690597005,
 2.8403079133878726,
 1.8598176057759601,
 1.615954782608101,
 5.415065956846516,
 1.0717854199068906,
 2.051150934507103,
 1.5316067656223655,
 1.4185093456162226,
 1.9907345705167858,
 1.2558499936611038,
 2.2324189057991104,
 1.9887971319213715,
 2.234439016418906,
 2.222154846658677,
 2.238832691009965,
 1.253373746571406,
 1.254843666755918,
 5.423985651304951,
 10.234011358224624,
 11.41858665599332,
 9.243761701496583,
 5.744642855600847,
 2.1701804963538103,
 2.1590663430835932,
 2.3008650458397644,
```

```
4373.2860773488055,
4.038575968826746,
2197.8949420436825,
1.3658952478912774,
2.377185518884185,
403.33415771994487,
1889.9079175673373]
```

```
In [86]: pd.DataFrame({ "coef_name : " : x.columns ,
                        "vif : ": np.around(vifs,2)})
```

Out[86]:

	coef_name :	vif :
0	loan_amnt	225.48
1	term	9.43
2	int_rate	55.72
3	installment	202.74
4	annual_inc	7.60
5	dti	8.15
6	open_acc	11.82
7	pub_rec	5.53
8	revol_bal	4.78
9	revol_util	9.57
10	total_acc	10.68
11	initial_list_status	2.74
12	mort_acc	3.77
13	pub_rec_bankruptcies	5.36
14	purpose_credit_card	18.54
15	purpose_debt_consolidation	50.90
16	purpose_educational	1.05
17	purpose_home_improvement	5.74
18	purpose_house	1.44
19	purpose_major_purchase	2.84
20	purpose_medical	1.86
21	purpose_moving	1.62
22	purpose_other	5.42
23	purpose_renewable_energy	1.07

	coef_name :	vif :
24	purpose_small_business	2.05
25	purpose_vacation	1.53
26	purpose_wedding	1.42
27	zip_code_05113	1.99
28	zip_code_11650	1.26
29	zip_code_22690	2.23
30	zip_code_29597	1.99
31	zip_code_30723	2.23
32	zip_code_48052	2.22
33	zip_code_70466	2.24
34	zip_code_86630	1.25
35	zip_code_93700	1.25
36	grade_B	5.42
37	grade_C	10.23
38	grade_D	11.42
39	grade_E	9.24
40	grade_F	5.74
41	grade_G	2.17
42	verification_status_Source Verified	2.16
43	verification_status_Verified	2.30
44	application_type_INDIVIDUAL	4373.29
45	application_type_JOINT	4.04
46	home_ownership_MORTGAGE	2197.89
47	home_ownership_NONE	1.37

	coef_name :	vif :
48	home_ownership_OTHER	2.38
49	home_ownership_OWN	403.33
50	home_ownership_RENT	1889.91

Feature Importance

In [87]:
```
feature_importance = pd.DataFrame(index = df.drop(["loan_status"],
                                              axis = 1).columns,
                                  data = logreg.coef_.ravel()).reset_index()
feature_importance
```

Out[87]:

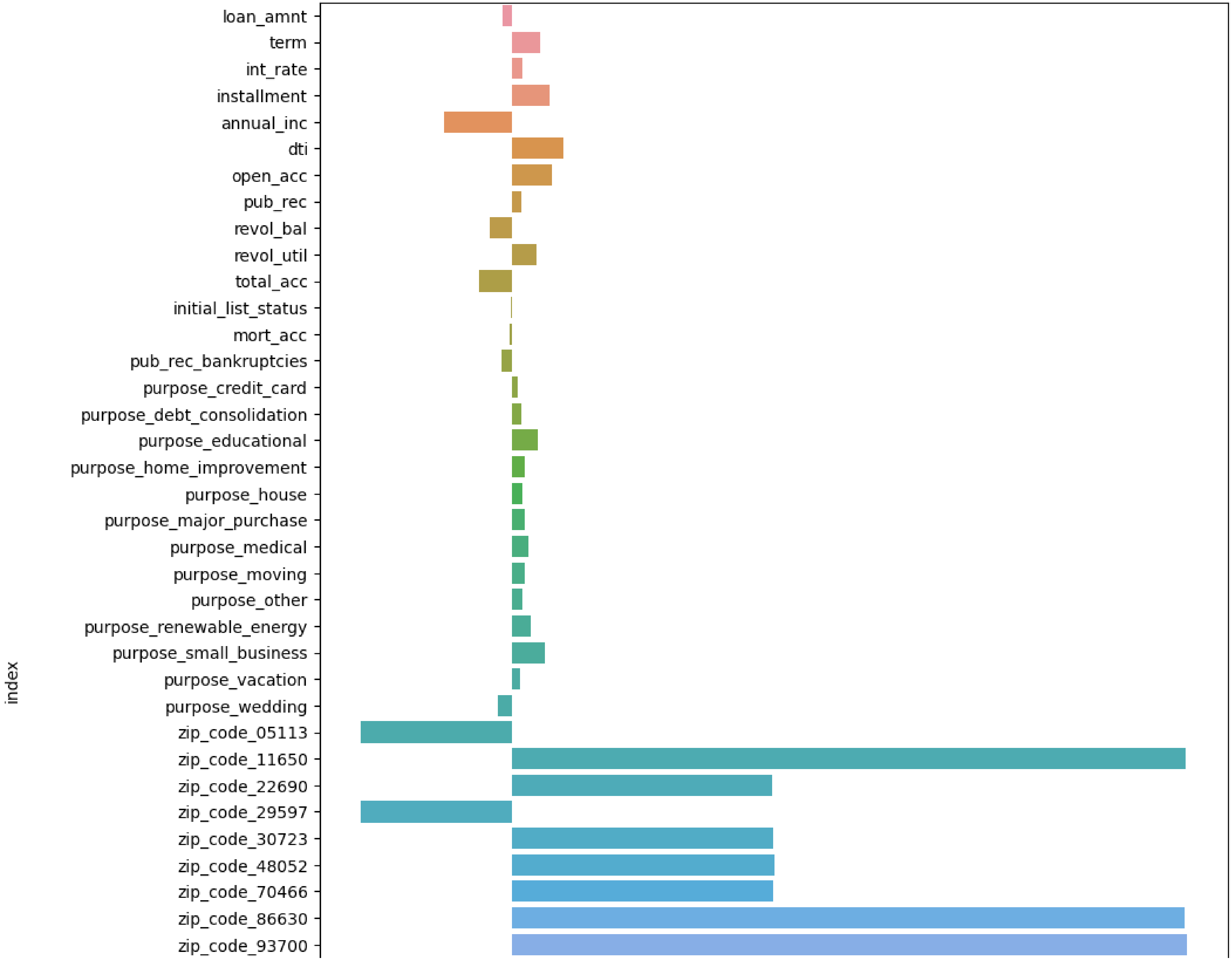
	index	0
0	loan_amnt	-0.173586
1	term	0.539073
2	int_rate	0.201938
3	installment	0.710599
4	annual_inc	-1.282681
5	dti	0.993002
6	open_acc	0.774550
7	pub_rec	0.193603
8	revol_bal	-0.413581
9	revol_util	0.468055
10	total_acc	-0.619208
11	initial_list_status	-0.022140
12	mort_acc	-0.034315
13	pub_rec_bankruptcies	-0.194417
14	purpose_credit_card	0.120761
15	purpose_debt_consolidation	0.195527
16	purpose_educational	0.504561
17	purpose_home_improvement	0.263403
18	purpose_house	0.211698
19	purpose_major_purchase	0.257423
20	purpose_medical	0.318343
21	purpose_moving	0.252790
22	purpose_other	0.215908
23	purpose_renewable_energy	0.361802

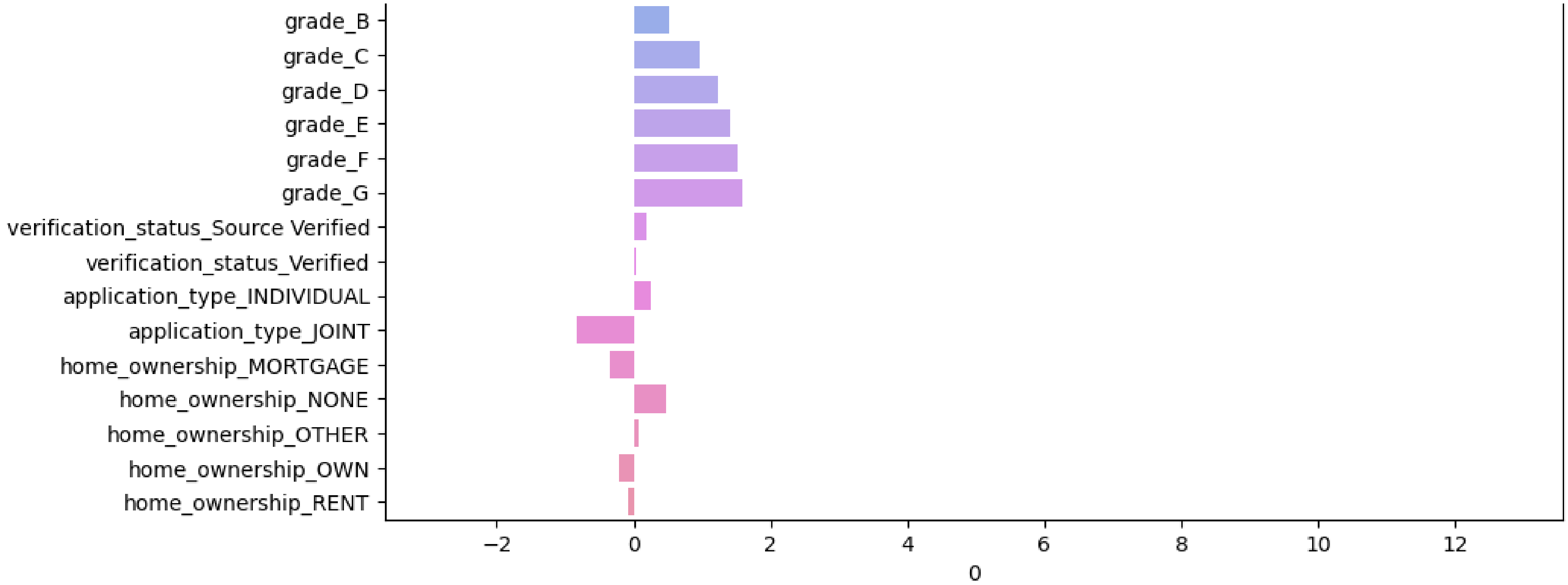
	index	0
24	purpose_small_business	0.621466
25	purpose_vacation	0.154104
26	purpose_wedding	-0.253782
27	zip_code_05113	-2.853648
28	zip_code_11650	12.781206
29	zip_code_22690	4.931077
30	zip_code_29597	-2.850273
31	zip_code_30723	4.948296
32	zip_code_48052	4.985105
33	zip_code_70466	4.948863
34	zip_code_86630	12.740321
35	zip_code_93700	12.784098
36	grade_B	0.517027
37	grade_C	0.971116
38	grade_D	1.239913
39	grade_E	1.415907
40	grade_F	1.527761
41	grade_G	1.584391
42	verification_status_Source Verified	0.193590
43	verification_status_Verified	0.030795
44	application_type_INDIVIDUAL	0.250398
45	application_type_JOINT	-0.824812
46	home_ownership_MORTGAGE	-0.336507
47	home_ownership_NONE	0.464522

	index	0
48	home_ownership_OTHER	0.071082
49	home_ownership_OWN	-0.212028
50	home_ownership_RENT	-0.079710

In [88]:
```
plt.figure(figsize=(10,15))
sns.barplot(y = feature_importance["index"],
            x =  feature_importance[0])
```

Out[88]: <AxesSubplot:xlabel='0', ylabel='index'>


loan_amnt
term
int_rate
installment
annual_inc
dti
open_acc
pub_rec
revol_bal
revol_util
total_acc
initial_list_status
mort_acc
pub_rec_bankruptcies
purpose_credit_card
purpose_debt_consolidation
purpose_educational
purpose_home_improvement
purpose_house
purpose_major_purchase
purpose_medical
purpose_moving
purpose_other
purpose_renewable_energy
purpose_small_business
purpose_vacation
purpose_wedding
zip_code_05113
zip_code_11650
zip_code_22690
zip_code_29597
zip_code_30723
zip_code_48052
zip_code_70466
zip_code_86630
zip_code_93700
index



Recommendations

- The optimal strategy to achieve the objective of balancing the risk of increasing NPAs by disbursing loans to defaulters with the opportunity to earn interest by disbursing loans to as many worthy customers as possible: maximise the F1 score along with the area under Precision Recall Curve (precision-recall trade-off)
- More complex classifiers like random forest would give better results compared to logistic regression because they are not restricted by the linearity of decision boundary.
- Since NPA is a real problem in the industry , Company should more investigate and check for the proof of assets. Since it was observed in probability plot, verified borrowers had higher probability of defaulters than non-verified.

Questionnaire

1. What percentage of customers have fully paid their Loan Amount?

- 80.75% people have fully paid their loans.

1. Comment about the correlation between Loan Amount and Installment features.

- Loan Amount and Installment are highly correlated.

1. The majority of people have home ownership as:

- Mortgage.

1. People with grades ‘A’ are more likely to fully pay their loan. (T/F)

- False. People with Grade B and C are more likely to fully pay their loans as observed in the countplot.

1. Name the top 2 afforded job titles.

- Teacher and Manager

1. Thinking from a bank's perspective, which metric should our primary focus be on: ROC AUC, Precision, Recall, F1 Score

- Focus on recall to avoid missing fraudulent transactions, even if it leads to more false positives requiring manual review. This will also help in fraud detection.

1. How does the gap in precision and recall affect the bank?

- By carefully analyzing the impact of the gap between precision and recall, banks can optimize their models for better financial performance and improved customer experience.

1. Which were the features that heavily affected the outcome?

- Purpose, DTI and Grade

1. Will the results be affected by geographical location? (Yes/No)

- Yes