LOAN_LENDING_CLUB_ANALYSIS_EDA_PROJECT_BY_

Lending Club Default Analysis

The analysis is divided into four main parts:

1. Data understanding

2. Data cleaning (cleaning missing values, removing redundant columns etc.)

3. Data Analysis

4. Recommendations

In [1]:
```
# load the necessary libraries for analysis

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# show all the details which is present in the dataset

pd.options.display.max_rows = 100
pd.options.display.max_columns = 111
```

In [3]:
```
%config InlineBackend.figure_format = 'retina'
```

In [4]:
```
# load and print the dataset

data = pd.read_csv("loan_csv.csv")
data
```

Out[4]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_ra
0	1077501	1296599	5000	5000	4975.0	36 months	10.65
1	1077430	1314167	2500	2500	2500.0	60 months	15.27
2	1077175	1313524	2400	2400	2400.0	36 months	15.96
3	1076863	1277178	10000	10000	10000.0	36 months	13.49
4	1075358	1311748	3000	3000	3000.0	60 months	12.69
...	...	...	...	...	...	...	
39712	92187	92174	2500	2500	1075.0	36 months	8.07
39713	90665	90607	8500	8500	875.0	36 months	10.28
39714	90395	90390	5000	5000	1325.0	36 months	8.07
39715	90376	89243	5000	5000	650.0	36 months	7.43
39716	87023	86999	7500	7500	800.0	36 months	13.75

39717 rows × 111 columns

Data Understanding

In [5]:
```
# first 5 rows of the dataset

data.head(5)
```

Out[5]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	i
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	
3	1076863	1277178	10000	10000	10000.0	36 months	13.49%	
4	1075358	1311748	3000	3000	3000.0	60 months	12.69%	

In [6]:
```
# last 5 rows of the dataset

data.tail(5)
```

Out[6]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate
39712	92187	92174	2500	2500	1075.0	36 months	8.07%
39713	90665	90607	8500	8500	875.0	36 months	10.28%
39714	90395	90390	5000	5000	1325.0	36 months	8.07%
39715	90376	89243	5000	5000	650.0	36 months	7.43%
39716	87023	86999	7500	7500	800.0	36 months	13.75%

In [7]:
```
# print the total no.of rows & columns

data.shape
```

Out[7]: (39717, 111)

In [8]:
```
# print the information of all the columns

data.info(verbose=True, show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 111 columns):
 #    Column                          Non-Null Count  Dtype
---   ------                          --------------  -----
 0    id                              39717 non-null  int64
 1    member_id                       39717 non-null  int64
 2    loan_amnt                       39717 non-null  int64
 3    funded_amnt                     39717 non-null  int64
 4    funded_amnt_inv                 39717 non-null  float64
 5    term                            39717 non-null  object
 6    int_rate                        39717 non-null  object
 7    installment                     39717 non-null  float64
 8    grade                           39717 non-null  object
 9    sub_grade                       39717 non-null  object
 10   emp_title                       37258 non-null  object
 11   emp_length                      38642 non-null  object
 12   home_ownership                  39717 non-null  object
 13   annual_inc                      39717 non-null  float64
 14   verification_status             39717 non-null  object
 15   issue_d                         39717 non-null  object
 16   loan_status                     39717 non-null  object
 17   pymnt_plan                      39717 non-null  object
 18   url                             39717 non-null  object
 19   desc                            26565 non-null  object
 20   purpose                         39717 non-null  object
 21   title                           39706 non-null  object
 22   zip_code                        39717 non-null  object
 23   addr_state                      39717 non-null  object
 24   dti                             39717 non-null  float64
 25   delinq_2yrs                     39717 non-null  int64
 26   earliest_cr_line                39717 non-null  object
 27   inq_last_6mths                  39717 non-null  int64
 28   mths_since_last_delinq          14035 non-null  float64
 29   mths_since_last_record          2786 non-null   float64
 30   open_acc                        39717 non-null  int64
 31   pub_rec                         39717 non-null  int64
 32   revol_bal                       39717 non-null  int64
 33   revol_util                      39667 non-null  object
 34   total_acc                       39717 non-null  int64
 35   initial_list_status             39717 non-null  object
 36   out_prncp                       39717 non-null  float64
 37   out_prncp_inv                   39717 non-null  float64
 38   total_pymnt                     39717 non-null  float64
 39   total_pymnt_inv                 39717 non-null  float64
 40   total_rec_prncp                 39717 non-null  float64
 41   total_rec_int                   39717 non-null  float64
 42   total_rec_late_fee              39717 non-null  float64
 43   recoveries                      39717 non-null  float64
 44   collection_recovery_fee         39717 non-null  float64
 45   last_pymnt_d                    39646 non-null  object
 46   last_pymnt_amnt                 39717 non-null  float64
 47   next_pymnt_d                    1140 non-null   object
 48   last_credit_pull_d              39715 non-null  object
 49   collections_12_mths_ex_med      39661 non-null  float64
 50   mths_since_last_major_derog     0 non-null      float64
 51   policy_code                     39717 non-null  int64
 52   application_type                39717 non-null  object
 53   annual_inc_joint                0 non-null      float64
 54   dti_joint                       0 non-null      float64
```

```
 55   verification_status_joint       0 non-null      float64
 56   acc_now_delinq                  39717 non-null  int64
 57   tot_coll_amt                    0 non-null      float64
 58   tot_cur_bal                     0 non-null      float64
 59   open_acc_6m                     0 non-null      float64
 60   open_il_6m                      0 non-null      float64
 61   open_il_12m                     0 non-null      float64
 62   open_il_24m                     0 non-null      float64
 63   mths_since_rcnt_il              0 non-null      float64
 64   total_bal_il                    0 non-null      float64
 65   il_util                         0 non-null      float64
 66   open_rv_12m                     0 non-null      float64
 67   open_rv_24m                     0 non-null      float64
 68   max_bal_bc                      0 non-null      float64
 69   all_util                        0 non-null      float64
 70   total_rev_hi_lim                0 non-null      float64
 71   inq_fi                          0 non-null      float64
 72   total_cu_tl                     0 non-null      float64
 73   inq_last_12m                    0 non-null      float64
 74   acc_open_past_24mths            0 non-null      float64
 75   avg_cur_bal                     0 non-null      float64
 76   bc_open_to_buy                  0 non-null      float64
 77   bc_util                         0 non-null      float64
 78   chargeoff_within_12_mths        39661 non-null  float64
 79   delinq_amnt                     39717 non-null  int64
 80   mo_sin_old_il_acct              0 non-null      float64
 81   mo_sin_old_rev_tl_op            0 non-null      float64
 82   mo_sin_rcnt_rev_tl_op           0 non-null      float64
 83   mo_sin_rcnt_tl                  0 non-null      float64
 84   mort_acc                        0 non-null      float64
 85   mths_since_recent_bc            0 non-null      float64
 86   mths_since_recent_bc_dlq        0 non-null      float64
 87   mths_since_recent_inq           0 non-null      float64
 88   mths_since_recent_revol_delinq  0 non-null      float64
 89   num_accts_ever_120_pd           0 non-null      float64
 90   num_actv_bc_tl                  0 non-null      float64
 91   num_actv_rev_tl                 0 non-null      float64
 92   num_bc_sats                     0 non-null      float64
 93   num_bc_tl                       0 non-null      float64
 94   num_il_tl                       0 non-null      float64
 95   num_op_rev_tl                   0 non-null      float64
 96   num_rev_accts                   0 non-null      float64
 97   num_rev_tl_bal_gt_0             0 non-null      float64
 98   num_sats                        0 non-null      float64
 99   num_tl_120dpd_2m                0 non-null      float64
 100  num_tl_30dpd                    0 non-null      float64
 101  num_tl_90g_dpd_24m              0 non-null      float64
 102  num_tl_op_past_12m              0 non-null      float64
 103  pct_tl_nvr_dlq                  0 non-null      float64
 104  percent_bc_gt_75                0 non-null      float64
 105  pub_rec_bankruptcies            39020 non-null  float64
 106  tax_liens                       39678 non-null  float64
 107  tot_hi_cred_lim                 0 non-null      float64
 108  total_bal_ex_mort               0 non-null      float64
 109  total_bc_limit                  0 non-null      float64
 110  total_il_high_credit_limit      0 non-null      float64
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB
```

In [9]:
```
# print overall statistics about the dataset

data.describe(include="all").T
```

Out[9]:

	count	unique	top	freq	mean	std	
id	39717.0	NaN	NaN	NaN	683131.91306	210694.132915	547
member_id	39717.0	NaN	NaN	NaN	850463.559408	265678.307421	706
loan_amnt	39717.0	NaN	NaN	NaN	11219.443815	7456.670694	5
funded_amnt	39717.0	NaN	NaN	NaN	10947.713196	7187.23867	5
funded_amnt_inv	39717.0	NaN	NaN	NaN	10397.448868	7128.450439	
...	...	...	...	...	...	...	
tax_liens	39678.0	NaN	NaN	NaN	0.0	0.0	
tot_hi_cred_lim	0.0	NaN	NaN	NaN	NaN	NaN	
total_bal_ex_mort	0.0	NaN	NaN	NaN	NaN	NaN	
total_bc_limit	0.0	NaN	NaN	NaN	NaN	NaN	
total_il_high_credit_limit	0.0	NaN	NaN	NaN	NaN	NaN	

111 rows × 11 columns

Data Cleaning

In [10]:
```
# finding null values of each column

missing_info = missing_info = pd.DataFrame(data.isnull().sum().sort_values()).re

# rename the heading and columns of the missing_info data

missing_info.rename(columns={'index':'col_name',0:'null_counts'},inplace=True)
missing_info.tail(100)
```

Out[10]:

	col_name	null_counts
11	total_pymnt	0
12	total_rec_int	0
13	total_rec_late_fee	0
14	recoveries	0
15	collection_recovery_fee	0
16	last_pymnt_amnt	0
17	policy_code	0
18	application_type	0
19	acc_now_delinq	0
20	delinq_amnt	0
21	total_pymnt_inv	0
22	dti	0
23	total_rec_prncp	0
24	zip_code	0
25	member_id	0
26	loan_amnt	0
27	addr_state	0
28	funded_amnt_inv	0
29	term	0
30	int_rate	0
31	installment	0
32	grade	0
33	sub_grade	0
34	home_ownership	0
35	annual_inc	0
36	funded_amnt	0
37	issue_d	0
38	purpose	0
39	verification_status	0
40	loan_status	0
41	pymnt_plan	0
42	url	0
43	last_credit_pull_d	2

	col_name	null_counts
44	title	11
45	tax_liens	39
46	revol_util	50
47	collections_12_mths_ex_med	56
48	chargeoff_within_12_mths	56
49	last_pymnt_d	71
50	pub_rec_bankruptcies	697
51	emp_length	1075
52	emp_title	2459
53	desc	13152
54	mths_since_last_delinq	25682
55	mths_since_last_record	36931
56	next_pymnt_d	38577
57	num_bc_sats	39717
58	mths_since_recent_bc	39717
59	mths_since_recent_bc_dlq	39717
60	mths_since_recent_inq	39717
61	mths_since_recent_revol_delinq	39717
62	num_accts_ever_120_pd	39717
63	num_actv_bc_tl	39717
64	num_actv_rev_tl	39717
65	mort_acc	39717
66	num_bc_tl	39717
67	num_tl_op_past_12m	39717
68	num_op_rev_tl	39717
69	num_rev_accts	39717
70	num_rev_tl_bal_gt_0	39717
71	num_sats	39717
72	num_tl_120dpd_2m	39717
73	num_tl_30dpd	39717
74	num_tl_90g_dpd_24m	39717
75	pct_tl_nvr_dlq	39717
76	percent_bc_gt_75	39717

	col_name	null_counts
77	tot_hi_cred_lim	39717
78	total_bal_ex_mort	39717
79	mo_sin_rcnt_tl	39717
80	num_il_tl	39717
81	mo_sin_rcnt_rev_tl_op	39717
82	verification_status_joint	39717
83	mo_sin_old_il_acct	39717
84	mths_since_last_major_derog	39717
85	annual_inc_joint	39717
86	dti_joint	39717
87	total_bc_limit	39717
88	tot_coll_amt	39717
89	tot_cur_bal	39717
90	open_acc_6m	39717
91	open_il_6m	39717
92	open_il_12m	39717
93	open_il_24m	39717
94	mths_since_rcnt_il	39717
95	total_bal_il	39717
96	il_util	39717
97	open_rv_12m	39717
98	open_rv_24m	39717
99	max_bal_bc	39717
100	all_util	39717
101	total_rev_hi_lim	39717
102	inq_fi	39717
103	total_cu_tl	39717
104	inq_last_12m	39717
105	acc_open_past_24mths	39717
106	avg_cur_bal	39717
107	bc_open_to_buy	39717
108	bc_util	39717
109	mo_sin_old_rev_tl_op	39717

	col_name	null_counts
110	total_il_high_credit_limit	39717

In [11]:
```
# again check the shape of the missing_column dataset

missing_info.shape
```

Out[11]:
```
(111, 2)
```

In [12]:
```
# calculate the missing percentage of the missing column and import in excel fil

missing_info["missing_percentage"] = missing_info["null_counts"] / data.shape[0]

# export in the excel file

missing_info.to_excel("missing_info_vivek_chauhan.xlsx")

missing_info["missing_percentage"]
```

Out[12]:
```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
       ...
106    100.0
107    100.0
108    100.0
109    100.0
110    100.0
Name: missing_percentage, Length: 111, dtype: float64
```

HANDLE THE MISSING VALUES BY DROP THE COLUMNS THAT CONTAIN MORE THAN 60 % MISSING VALUES OR NULL VALUES.

In [13]:
```
# make a seperate list those columns names that contains more than 60% null data

miss_col = missing_info[missing_info['missing_percentage']>=60]['col_name'].to_l
len(miss_col) # no of columns that contains null values
```

Out[13]:
```
57
```

In [14]:
```
# remove column that contains more than 60% of the dataset

new_data = data.drop(columns = miss_col,axis=1)
new_data
```

Out[14]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_ra
0	1077501	1296599	5000	5000	4975.0	36 months	10.6
1	1077430	1314167	2500	2500	2500.0	60 months	15.2
2	1077175	1313524	2400	2400	2400.0	36 months	15.9
3	1076863	1277178	10000	10000	10000.0	36 months	13.4
4	1075358	1311748	3000	3000	3000.0	60 months	12.6
...	...	...	...	...	...	...	
39712	92187	92174	2500	2500	1075.0	36 months	8.0
39713	90665	90607	8500	8500	875.0	36 months	10.2
39714	90395	90390	5000	5000	1325.0	36 months	8.0
39715	90376	89243	5000	5000	650.0	36 months	7.4
39716	87023	86999	7500	7500	800.0	36 months	13.7

39717 rows × 54 columns

In [15]:
```
# print all the column names which is present in the dataset
```

```
new_data.columns
```

Out[15]:
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title',
       'emp_length', 'home_ownership', 'annual_inc', 'verification_status',
       'issue_d', 'loan_status', 'pymnt_plan', 'url', 'desc', 'purpose',
       'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs',
       'earliest_cr_line', 'inq_last_6mths', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'out_prncp', 'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv',
       'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee', 'recoveries',
       'collection_recovery_fee', 'last_pymnt_d', 'last_pymnt_amnt',
       'last_credit_pull_d', 'collections_12_mths_ex_med', 'policy_code',
       'application_type', 'acc_now_delinq', 'chargeoff_within_12_mths',
       'delinq_amnt', 'pub_rec_bankruptcies', 'tax_liens'],
      dtype='object')
```

In [16]:
```
# recheck which columns contains null values

new_data.isnull().sum().sort_values(ascending=False)
```

Out[16]:
```
desc                          13152
emp_title                      2459
emp_length                     1075
pub_rec_bankruptcies            697
last_pymnt_d                     71
collections_12_mths_ex_med       56
chargeoff_within_12_mths         56
revol_util                       50
tax_liens                        39
title                            11
last_credit_pull_d                2
total_rec_prncp                   0
out_prncp                         0
initial_list_status               0
out_prncp_inv                     0
total_acc                         0
total_pymnt                       0
total_pymnt_inv                   0
collection_recovery_fee           0
total_rec_int                     0
total_rec_late_fee                0
recoveries                        0
pub_rec                           0
last_pymnt_amnt                   0
policy_code                       0
application_type                  0
acc_now_delinq                    0
delinq_amnt                       0
revol_bal                         0
id                                0
open_acc                          0
member_id                         0
loan_amnt                         0
funded_amnt                       0
funded_amnt_inv                   0
term                              0
int_rate                          0
installment                       0
grade                             0
sub_grade                         0
home_ownership                    0
annual_inc                        0
verification_status               0
issue_d                           0
loan_status                       0
pymnt_plan                        0
url                               0
purpose                           0
zip_code                          0
addr_state                        0
dti                               0
delinq_2yrs                       0
earliest_cr_line                  0
inq_last_6mths                    0
dtype: int64
```

In [17]:
```
# filterout columns that contains null values

new_data.isnull().sum().sort_values(ascending=False).head(11)
```

Out[17]:
```
desc                          13152
emp_title                      2459
emp_length                     1075
pub_rec_bankruptcies            697
last_pymnt_d                     71
collections_12_mths_ex_med       56
chargeoff_within_12_mths         56
revol_util                       50
tax_liens                        39
title                            11
last_credit_pull_d                2
dtype: int64
```

In [18]:
```
# create a seperate list of column names that contains null values

null_col = new_data.isnull().sum().sort_values(ascending=False).head(11).index.t
null_col
```

Out[18]:
```
['desc',
 'emp_title',
 'emp_length',
 'pub_rec_bankruptcies',
 'last_pymnt_d',
 'collections_12_mths_ex_med',
 'chargeoff_within_12_mths',
 'revol_util',
 'tax_liens',
 'title',
 'last_credit_pull_d']
```

In [19]:
```
# check the null_col one by one so we can identify which column is usefull for a

# new_data.groupby("desc").size() # remove the column
# new_data.groupby("emp_title").size() # remove the column
# new_data.groupby("emp_length").size() # usefull column we can identify which e
# new_data.groupby("pub_rec_bankruptcies").size() # usefull column we can identi
# new_data.groupby("last_pymnt_d").size() # remove the column
# new_data.groupby("collections_12_mths_ex_med").size() # remove the column we d
# new_data.groupby("chargeoff_within_12_mths").size() # remove the column
# new_data.groupby("revol_util").size() # remove the column we dont need the uti
# new_data.groupby("tax_liens").size() # remove the column
# new_data.groupby("title").size() # remove the column we dont need the title of
# new_data.groupby("last_credit_pull_d").size() # remove the column we dont need
```

In [20]:
```
# so remove the unnecessory column for better analysis

new_data = new_data.drop(labels=['desc','emp_title','last_pymnt_d','collections_
```

In [21]:
```
# print the new_data with dropped columns

new_data
```

Out[21]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_ra
0	1077501	1296599	5000	5000	4975.0	36 months	10.65
1	1077430	1314167	2500	2500	2500.0	60 months	15.27
2	1077175	1313524	2400	2400	2400.0	36 months	15.96
3	1076863	1277178	10000	10000	10000.0	36 months	13.49
4	1075358	1311748	3000	3000	3000.0	60 months	12.69
...	...	...	...	...	...	...	
39712	92187	92174	2500	2500	1075.0	36 months	8.07
39713	90665	90607	8500	8500	875.0	36 months	10.28
39714	90395	90390	5000	5000	1325.0	36 months	8.07
39715	90376	89243	5000	5000	650.0	36 months	7.43
39716	87023	86999	7500	7500	800.0	36 months	13.75

39717 rows × 45 columns

In [22]:
```
# check which one is best fit for fill the missing values in the column emp_leng

print(new_data.groupby('emp_length').size()) # here most of the people emp_lengt
print("the mode of the emp_length",new_data['emp_length'].mode())
```

```
emp_length
1 year       3240
10+ years    8879
2 years      4388
3 years      4095
4 years      3436
5 years      3282
6 years      2229
7 years      1773
8 years      1479
9 years      1258
< 1 year     4583
dtype: int64
the mode of the emp_length 0    10+ years
Name: emp_length, dtype: object
```

In [23]:
```
# fill missing values in the emp_length column with mode
```

```
new_data['emp_length'] = new_data['emp_length'].fillna(new_data['emp_length'].mo
new_data
```

Out[23]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_ra
0	1077501	1296599	5000	5000	4975.0	36 months	10.6
1	1077430	1314167	2500	2500	2500.0	60 months	15.2
2	1077175	1313524	2400	2400	2400.0	36 months	15.9
3	1076863	1277178	10000	10000	10000.0	36 months	13.4
4	1075358	1311748	3000	3000	3000.0	60 months	12.6
...	...	...	...	...	...	...	
39712	92187	92174	2500	2500	1075.0	36 months	8.0
39713	90665	90607	8500	8500	875.0	36 months	10.2
39714	90395	90390	5000	5000	1325.0	36 months	8.0
39715	90376	89243	5000	5000	650.0	36 months	7.4
39716	87023	86999	7500	7500	800.0	36 months	13.7

39717 rows × 45 columns

In [24]:
```
# check which one is best fit for fill the missing values in the column pub_rec_

print(new_data.groupby('pub_rec_bankruptcies').size()) # here is the 3 counts of
print("the mode of the pub_rec_bankruptcies",new_data['pub_rec_bankruptcies'].mo
```

```
pub_rec_bankruptcies
0.0    37339
1.0     1674
2.0        7
dtype: int64
the mode of the pub_rec_bankruptcies 0    0.0
Name: pub_rec_bankruptcies, dtype: float64
```

In [25]:
```
# fill missing values in the pub_rec_bankruptcies column with mode

new_data['pub_rec_bankruptcies'] = new_data['pub_rec_bankruptcies'].fillna(new_d
new_data
```

Out[25]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_ra
0	1077501	1296599	5000	5000	4975.0	36 months	10.65
1	1077430	1314167	2500	2500	2500.0	60 months	15.27
2	1077175	1313524	2400	2400	2400.0	36 months	15.96
3	1076863	1277178	10000	10000	10000.0	36 months	13.49
4	1075358	1311748	3000	3000	3000.0	60 months	12.69
...	...	...	...	...	...	...	
39712	92187	92174	2500	2500	1075.0	36 months	8.07
39713	90665	90607	8500	8500	875.0	36 months	10.28
39714	90395	90390	5000	5000	1325.0	36 months	8.07
39715	90376	89243	5000	5000	650.0	36 months	7.43
39716	87023	86999	7500	7500	800.0	36 months	13.75

39717 rows × 45 columns

In [26]:
```
# summarise number of missing values again

100*(new_data.isnull().sum()/len(new_data.index))
```

Out[26]:
```
id                          0.0
member_id                   0.0
loan_amnt                   0.0
funded_amnt                 0.0
funded_amnt_inv             0.0
term                        0.0
int_rate                    0.0
installment                 0.0
grade                       0.0
sub_grade                   0.0
emp_length                  0.0
home_ownership              0.0
annual_inc                  0.0
verification_status         0.0
issue_d                     0.0
loan_status                 0.0
pymnt_plan                  0.0
url                         0.0
purpose                     0.0
zip_code                    0.0
addr_state                  0.0
dti                         0.0
delinq_2yrs                 0.0
earliest_cr_line            0.0
inq_last_6mths              0.0
open_acc                    0.0
pub_rec                     0.0
revol_bal                   0.0
total_acc                   0.0
initial_list_status         0.0
out_prncp                   0.0
out_prncp_inv               0.0
total_pymnt                 0.0
total_pymnt_inv             0.0
total_rec_prncp             0.0
total_rec_int               0.0
total_rec_late_fee          0.0
recoveries                  0.0
collection_recovery_fee     0.0
last_pymnt_amnt             0.0
policy_code                 0.0
application_type            0.0
acc_now_delinq              0.0
delinq_amnt                 0.0
pub_rec_bankruptcies        0.0
dtype: float64
```

In [27]:
```
# again count the length of the columns

len(new_data.columns)
```

Out[27]:
```
45
```

In [28]:
```
# The column int_rate is character type, let's convert it to float

new_data['int_rate'] = new_data['int_rate'].apply(lambda x: pd.to_numeric(x.spli
```

In [29]:
```
# checking the data types
```

```
new_data.dtypes
```

Out[29]:

```
id                          int64
member_id                   int64
loan_amnt                   int64
funded_amnt                 int64
funded_amnt_inv           float64
term                       object
int_rate                  float64
installment               float64
grade                      object
sub_grade                  object
emp_length                 object
home_ownership             object
annual_inc                float64
verification_status        object
issue_d                    object
loan_status                object
pymnt_plan                 object
url                        object
purpose                    object
zip_code                   object
addr_state                 object
dti                       float64
delinq_2yrs                 int64
earliest_cr_line           object
inq_last_6mths              int64
open_acc                    int64
pub_rec                     int64
revol_bal                   int64
total_acc                   int64
initial_list_status        object
out_prncp                 float64
out_prncp_inv             float64
total_pymnt               float64
total_pymnt_inv           float64
total_rec_prncp           float64
total_rec_int             float64
total_rec_late_fee        float64
recoveries                float64
collection_recovery_fee   float64
last_pymnt_amnt           float64
policy_code                 int64
application_type           object
acc_now_delinq              int64
delinq_amnt                 int64
pub_rec_bankruptcies      float64
dtype: object
```

In [30]:
```
# extract the numerical part from the string data

# first, let's drop the missing values from the column (otherwise the regex code
new_data = new_data[~new_data['emp_length'].isnull()]

# using regular expression to extract numeric values from the string
import re
new_data['emp_length'] = new_data['emp_length'].apply(lambda x: re.findall('\d+'

# convert to numeric
new_data['emp_length'] = new_data['emp_length'].apply(lambda x: pd.to_numeric(x)
```

In [31]:
```
# let's count the employee length values

new_data["emp_length"].value_counts()
```

Out[31]:
```
emp_length
10    9954
1     7823
2     4388
3     4095
4     3436
5     3282
6     2229
7     1773
8     1479
9     1258
Name: count, dtype: int64
```

Data Analysis

The objective is to identify predictors of default so that at the time of loan application, we can use those variables for approval/rejection of the loan.

There are broadly three types of variables –

1. those which are related to the applicant (demographic variables such as age, occupation, employment details etc.),

Loan characteristics (amount of loan, interest rate, purpose of loan etc.) and

2. Customer behavior variables (those which are generated after the loan is approved such as delinquent 2 years, revolving balance, next payment date etc.).

Now, the customer behavior variables are not available at the time of loan application, and thus they cannot be used as predictors for credit approval.

The ones marked 'current' are neither fully paid not defaulted, so get rid of the current loans. Also, tag the other two values as 0 or 1 to make your analysis simple and clean.

In [32]:
```
behaviour_var =  [
  "delinq_2yrs",
```

```
  "earliest_cr_line",
  "inq_last_6mths",
  "open_acc",
  "pub_rec",
  "revol_bal",
  "total_acc",
  "out_prncp",
  "out_prncp_inv",
  "total_pymnt",
  "total_pymnt_inv",
  "total_rec_prncp",
  "total_rec_int",
  "total_rec_late_fee",
  "recoveries",
  "collection_recovery_fee",
  "last_pymnt_amnt",
  "application_type"]
behaviour_var
```

Out[32]:
```
['delinq_2yrs',
 'earliest_cr_line',
 'inq_last_6mths',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'total_acc',
 'out_prncp',
 'out_prncp_inv',
 'total_pymnt',
 'total_pymnt_inv',
 'total_rec_prncp',
 'total_rec_int',
 'total_rec_late_fee',
 'recoveries',
 'collection_recovery_fee',
 'last_pymnt_amnt',
 'application_type']
```

In [33]:
```
# let's now remove the behaviour variables from analysis

new_data = new_data.drop(behaviour_var, axis=1)
```

In [34]:
```
# also, we will not be able to use the variables zip code, address, state etc.
# the variable 'title' is derived from the variable 'purpose'
# thus let get rid of all these variables as well

new_data = new_data.drop(['url', 'zip_code', 'addr_state'], axis=1)
```

In [35]:
```
# count the category of the loan status data

new_data["loan_status"].value_counts() # here we see the int64 data type so we n
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    39717 non-null  int64  
 1   member_id             39717 non-null  int64  
 2   loan_amnt             39717 non-null  int64  
 3   funded_amnt           39717 non-null  int64  
 4   funded_amnt_inv       39717 non-null  float64
 5   term                  39717 non-null  object 
 6   int_rate              39717 non-null  float64
 7   installment           39717 non-null  float64
 8   grade                 39717 non-null  object 
 9   sub_grade             39717 non-null  object 
 10  emp_length            39717 non-null  int64  
 11  home_ownership        39717 non-null  object 
 12  annual_inc            39717 non-null  float64
 13  verification_status   39717 non-null  object 
 14  issue_d               39717 non-null  object 
 15  loan_status           39717 non-null  object 
 16  pymnt_plan            39717 non-null  object 
 17  purpose               39717 non-null  object 
 18  dti                   39717 non-null  float64
 19  initial_list_status   39717 non-null  object 
 20  policy_code           39717 non-null  int64  
 21  acc_now_delinq        39717 non-null  int64  
 22  delinq_amnt           39717 non-null  int64  
 23  pub_rec_bankruptcies  39717 non-null  float64
dtypes: float64(6), int64(8), object(10)
memory usage: 7.3+ MB
```

In [36]:
```
# let's check the column one by one and remove that column is unnecessory

# new_data["pymnt_plan"].value_counts() # remove the column
# new_data["initial_list_status"].value_counts() # remove the column
# new_data["policy_code"].value_counts() # remove the column
# new_data["acc_now_delinq"].value_counts() # remove the column
# new_data["delinq_amnt"].value_counts() # remove the column
```

In [37]:
```
# remove the unnecessory columns for better analysis

new_data = new_data.drop(["pymnt_plan","initial_list_status","policy_code","acc_
```

Clean Data Analysis

In [38]:
```
# again print the dataset here starts our analysis

new_data
```

Out[38]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_ra
0	1077501	1296599	5000	5000	4975.0	36 months	10.
1	1077430	1314167	2500	2500	2500.0	60 months	15.
2	1077175	1313524	2400	2400	2400.0	36 months	15.
3	1076863	1277178	10000	10000	10000.0	36 months	13.
4	1075358	1311748	3000	3000	3000.0	60 months	12.
...	...	...	...	...	...	...	
39712	92187	92174	2500	2500	1075.0	36 months	8.
39713	90665	90607	8500	8500	875.0	36 months	10.
39714	90395	90390	5000	5000	1325.0	36 months	8.
39715	90376	89243	5000	5000	650.0	36 months	7.
39716	87023	86999	7500	7500	800.0	36 months	13.

39717 rows × 19 columns

In [39]:
```
# print all the columns of the dataset

new_data.columns
```

Out[39]:
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'pub_rec_bankruptcies'],
      dtype='object')
```

In [40]:
```
# print the shape of our dataset

new_data.shape
```

Out[40]:
```
(39717, 19)
```

In [41]:
```
# print top10 data from the dataset

new_data.head(10)
```

Out[41]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	i
0	1077501	1296599	5000	5000	4975.0	36 months	10.65	
1	1077430	1314167	2500	2500	2500.0	60 months	15.27	
2	1077175	1313524	2400	2400	2400.0	36 months	15.96	
3	1076863	1277178	10000	10000	10000.0	36 months	13.49	
4	1075358	1311748	3000	3000	3000.0	60 months	12.69	
5	1075269	1311441	5000	5000	5000.0	36 months	7.90	
6	1069639	1304742	7000	7000	7000.0	60 months	15.96	
7	1072053	1288686	3000	3000	3000.0	36 months	18.64	
8	1071795	1306957	5600	5600	5600.0	60 months	21.28	
9	1071570	1306721	5375	5375	5350.0	60 months	12.69	

In [42]:
```
# print last10 data from the dataset

new_data.tail(10)
```

Out[42]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate
39707	92666	92661	5000	5000	525.0	36 months	9.33
39708	92552	92542	5000	5000	375.0	36 months	9.96
39709	92533	92529	5000	5000	675.0	36 months	11.22
39710	92507	92502	5000	5000	250.0	36 months	7.43
39711	92402	92390	5000	5000	700.0	36 months	8.70
39712	92187	92174	2500	2500	1075.0	36 months	8.07
39713	90665	90607	8500	8500	875.0	36 months	10.28
39714	90395	90390	5000	5000	1325.0	36 months	8.07
39715	90376	89243	5000	5000	650.0	36 months	7.43
39716	87023	86999	7500	7500	800.0	36 months	13.75

In [43]:
```
# print overall statistics of our dataset

new_data.describe(include = "all").T
```

Out[43]:

	count	unique	top	freq	mean	
id	39717.0	NaN	NaN	NaN	683131.91306	210694
member_id	39717.0	NaN	NaN	NaN	850463.559408	265678
loan_amnt	39717.0	NaN	NaN	NaN	11219.443815	7456
funded_amnt	39717.0	NaN	NaN	NaN	10947.713196	718
funded_amnt_inv	39717.0	NaN	NaN	NaN	10397.448868	7128
term	39717	2	36 months	29096	NaN	
int_rate	39717.0	NaN	NaN	NaN	12.021177	3
installment	39717.0	NaN	NaN	NaN	324.561922	208
grade	39717	7	B	12020	NaN	
sub_grade	39717	35	B3	2917	NaN	
emp_length	39717.0	NaN	NaN	NaN	5.224891	
home_ownership	39717	5	RENT	18899	NaN	
annual_inc	39717.0	NaN	NaN	NaN	68968.926377	6379
verification_status	39717	3	Not Verified	16921	NaN	
issue_d	39717	55	Dec-11	2260	NaN	
loan_status	39717	3	Fully Paid	32950	NaN	
purpose	39717	14	debt_consolidation	18641	NaN	
dti	39717.0	NaN	NaN	NaN	13.31513	6
pub_rec_bankruptcies	39717.0	NaN	NaN	NaN	0.042501	0

In [44]:
```
# print information of our dataset

new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    39717 non-null  int64  
 1   member_id             39717 non-null  int64  
 2   loan_amnt             39717 non-null  int64  
 3   funded_amnt           39717 non-null  int64  
 4   funded_amnt_inv       39717 non-null  float64
 5   term                  39717 non-null  object 
 6   int_rate              39717 non-null  float64
 7   installment           39717 non-null  float64
 8   grade                 39717 non-null  object 
 9   sub_grade             39717 non-null  object 
 10  emp_length            39717 non-null  int64  
 11  home_ownership        39717 non-null  object 
 12  annual_inc            39717 non-null  float64
 13  verification_status   39717 non-null  object 
 14  issue_d               39717 non-null  object 
 15  loan_status           39717 non-null  object 
 16  purpose               39717 non-null  object 
 17  dti                   39717 non-null  float64
 18  pub_rec_bankruptcies  39717 non-null  float64
dtypes: float64(6), int64(5), object(8)
memory usage: 5.8+ MB
```

In [45]:
```
# change the load_status datatype

new_data['loan_status'] = new_data['loan_status'].astype('category')
new_data["loan_status"].value_counts()
```

Out[45]:
```
loan_status
Fully Paid     32950
Charged Off     5627
Current         1140
Name: count, dtype: int64
```

"Here, you can see that the number of Fully Paid customers is higher compared to those with charge-offs and current loan status".

Most customers pay their loans on time, but a few do not pay on or before the due date, so they are considered charged off. We have removed customers with a current loan status from the analysis because we cannot predict whether they will pay on time or not."

In [46]:
```
# filtering only fully paid or charged-off

new_data = new_data[new_data['loan_status'] != 'Current']
```

```
new_data['loan_status'] = new_data['loan_status'].apply(lambda x: 0 if x=='Fully

# converting loan_status to integer type
new_data['loan_status'] = new_data['loan_status'].apply(lambda x: pd.to_numeric(

# summarising the values
new_data['loan_status'].value_counts()
```

Out[46]:
```
loan_status
0    32950
1     5627
Name: count, dtype: int64
```

Univariate Analysis

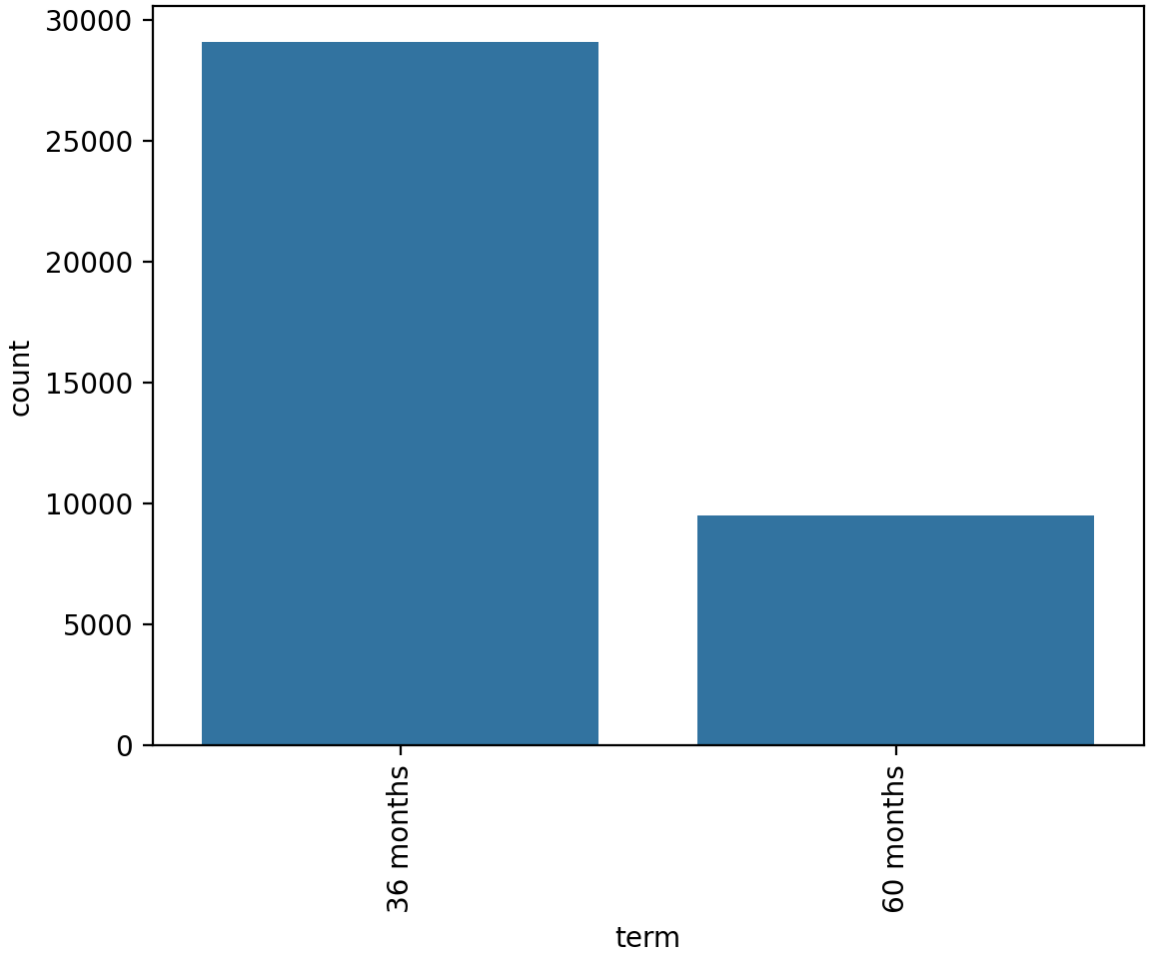
In [163…
```
# check the ratio of the defaulters in the loan_status
# defaulters means those who are not pay thire installments/loan in due time for

defaulters_percentage = round(np.mean(new_data['loan_status']), 2) * 100  # roun
percentage = round(defaulters_percentage,2)
percentage
```

Out[163…
```
14.0
```

Aprox 14 % of the customers are defaulters.

In [48]:
```
# lets define a function to plot new_data

def plot_count(cat_var):
    sns.countplot(x=cat_var,data=new_data)
    plt.xticks(rotation="vertical")
    plt.show()
```

In [49]:
```
# let's count the term wise loan appicants

plot_count("term")
```



Among the applicants — including those with charge-offs as well as those who have fully paid — the 36-month loan term appears more frequently.

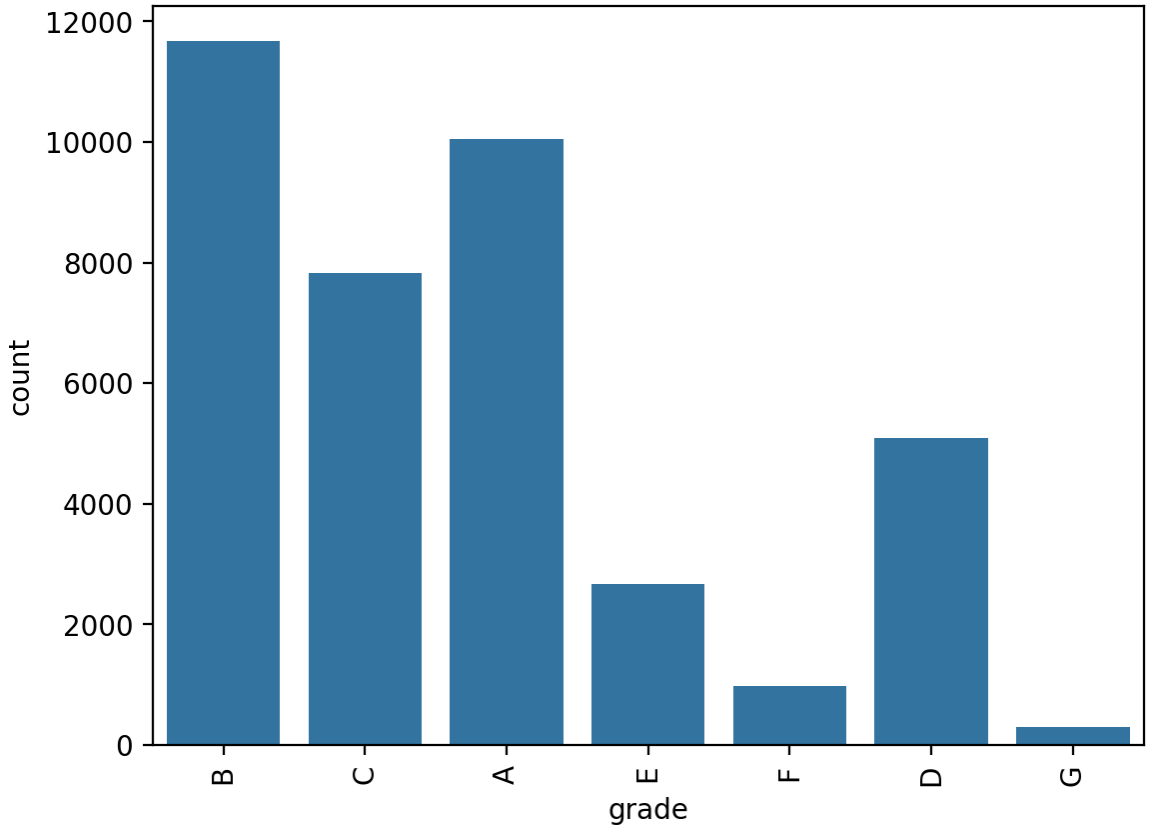
In [50]:
```
new_data["loan_status"].value_counts()
```

Out[50]:
```
loan_status
0    32950
1     5627
Name: count, dtype: int64
```

In [51]:
```
new_data["term"].value_counts()
```

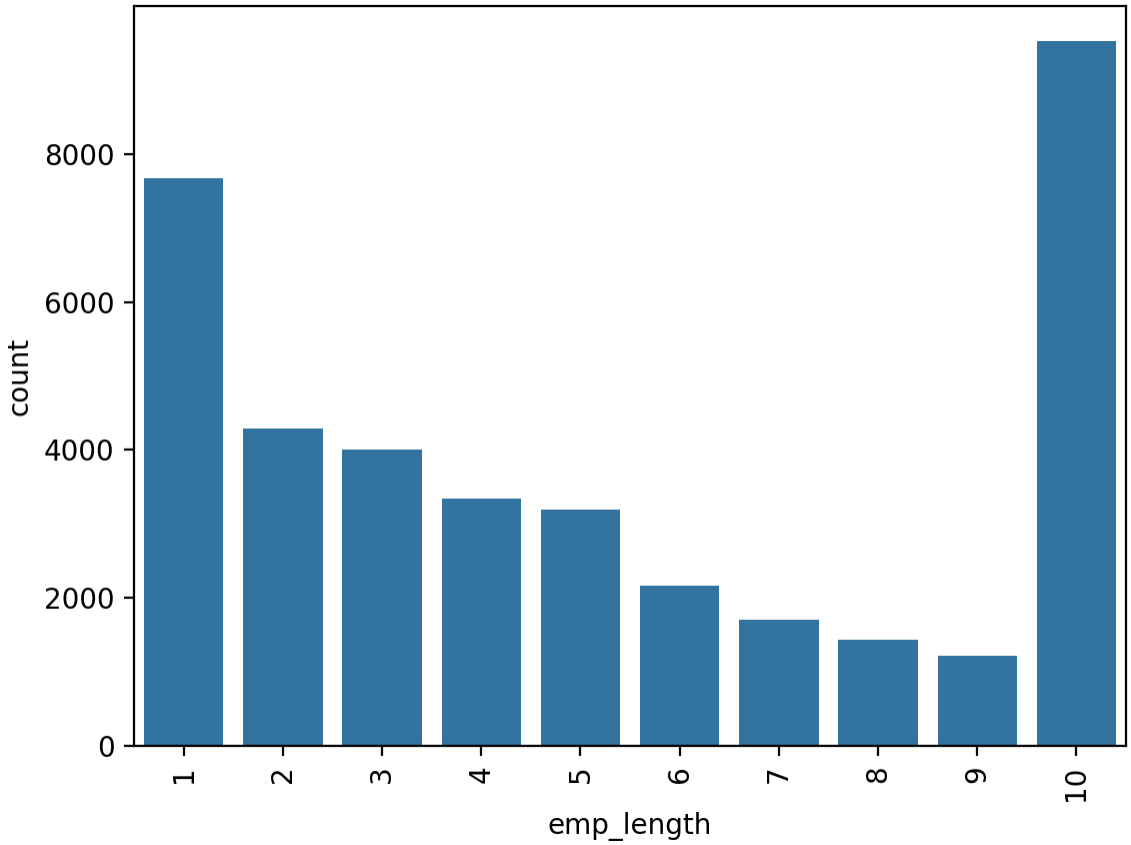
Out[51]:
```
term
36 months    29096
60 months     9481
Name: count, dtype: int64
```

In [52]:
```
# let's count the grade wise loan appicants

plot_count("grade")
```



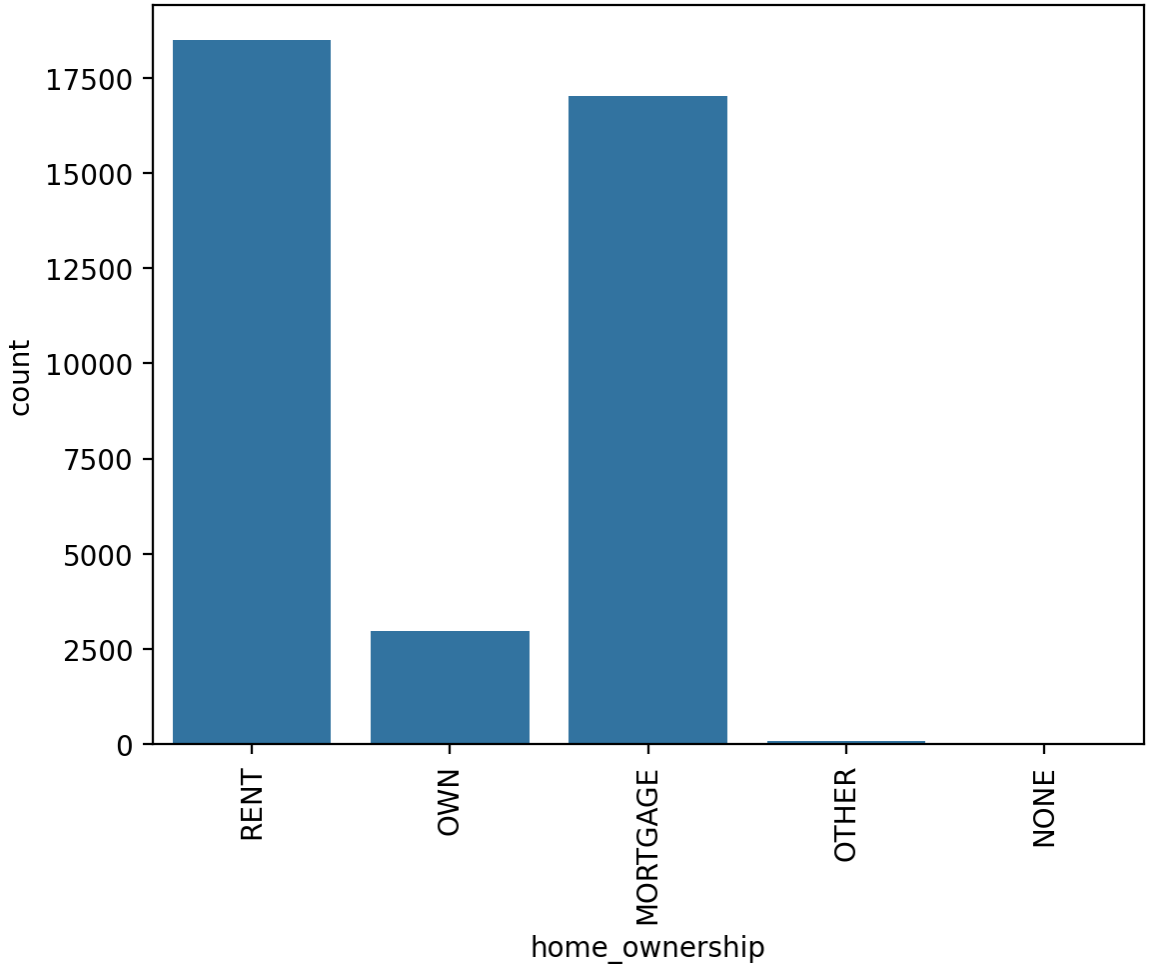
The majority of the applicants fall under Grade A, B, and C categories.

In [53]:
```
# let's count the emp_length wise loan appicants

plot_count("emp_length")
```



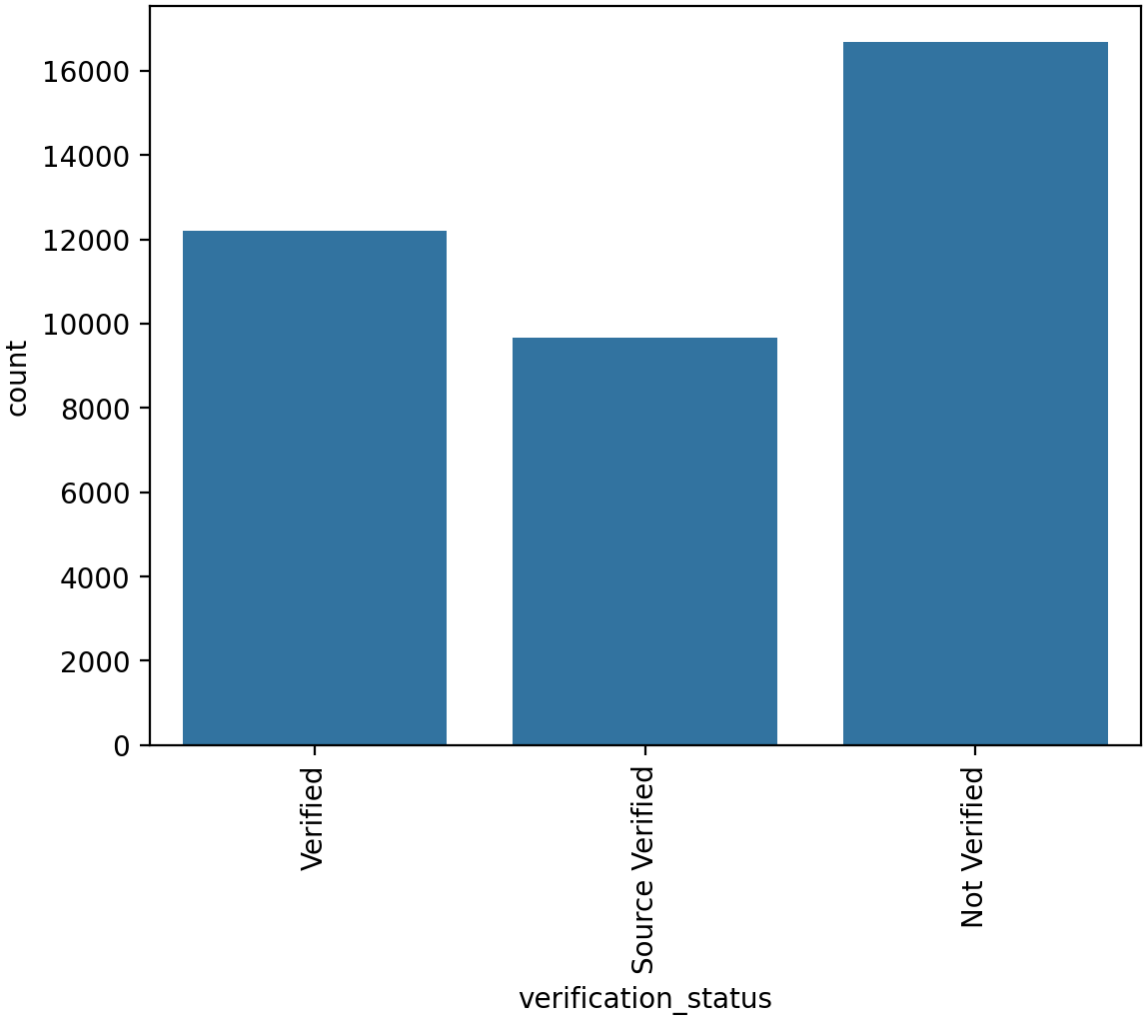
Most applicants have an employment length of 1, 2, or 10 years.

In [54]:
```
# let's count the home ownership wise loan appicants

plot_count("home_ownership")
```



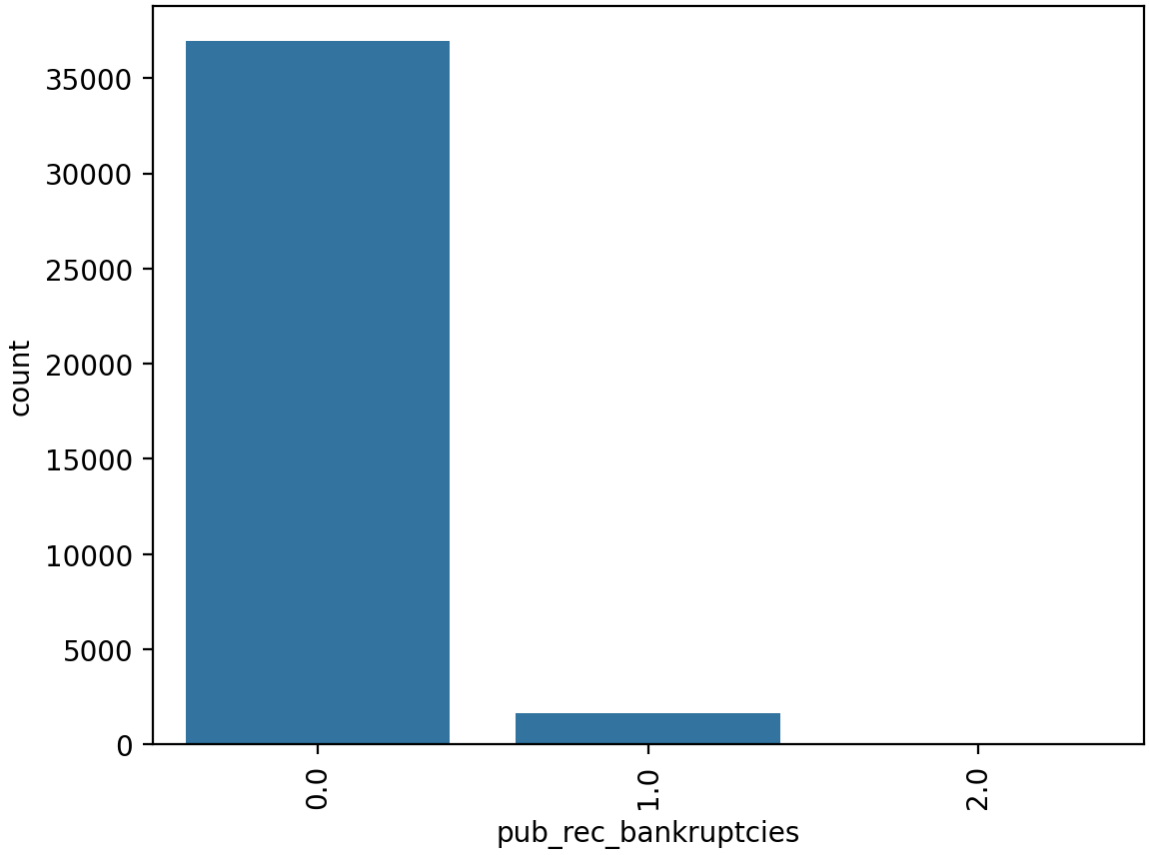
The majority of the applicants either rent their homes or have a mortgage.

In [55]:
```
# let's count the verification status wise loan appicants

plot_count("verification_status")
```



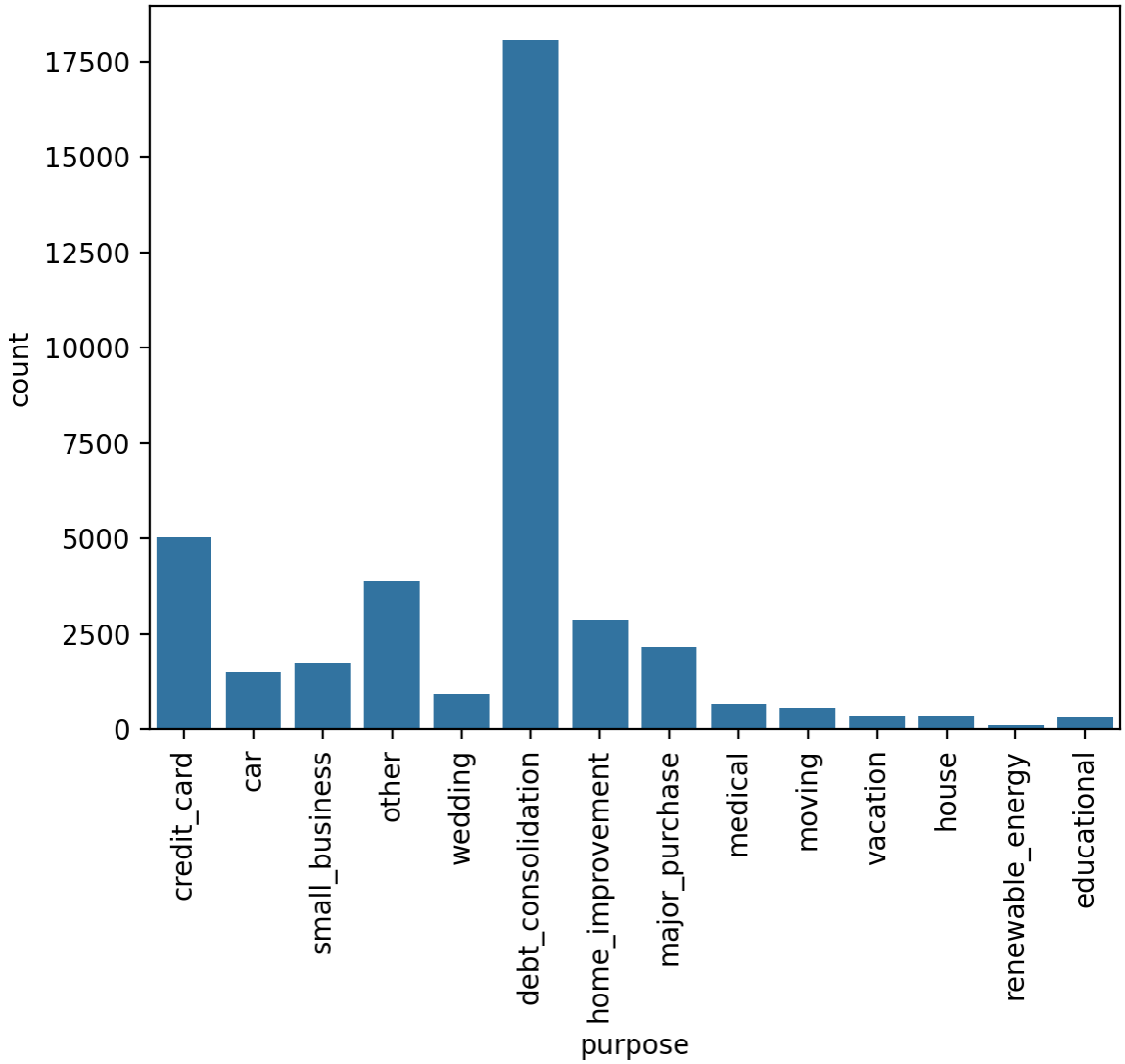
A large number of applicants have not been income verified.

In [56]:
```
# let's count the dti wise loan appicants

plot_count("pub_rec_bankruptcies")
```



Most applicants have no public bankruptcy records.

In [57]:
```
# let's count the emp_length wise loan appicants

plot_count("purpose")
```

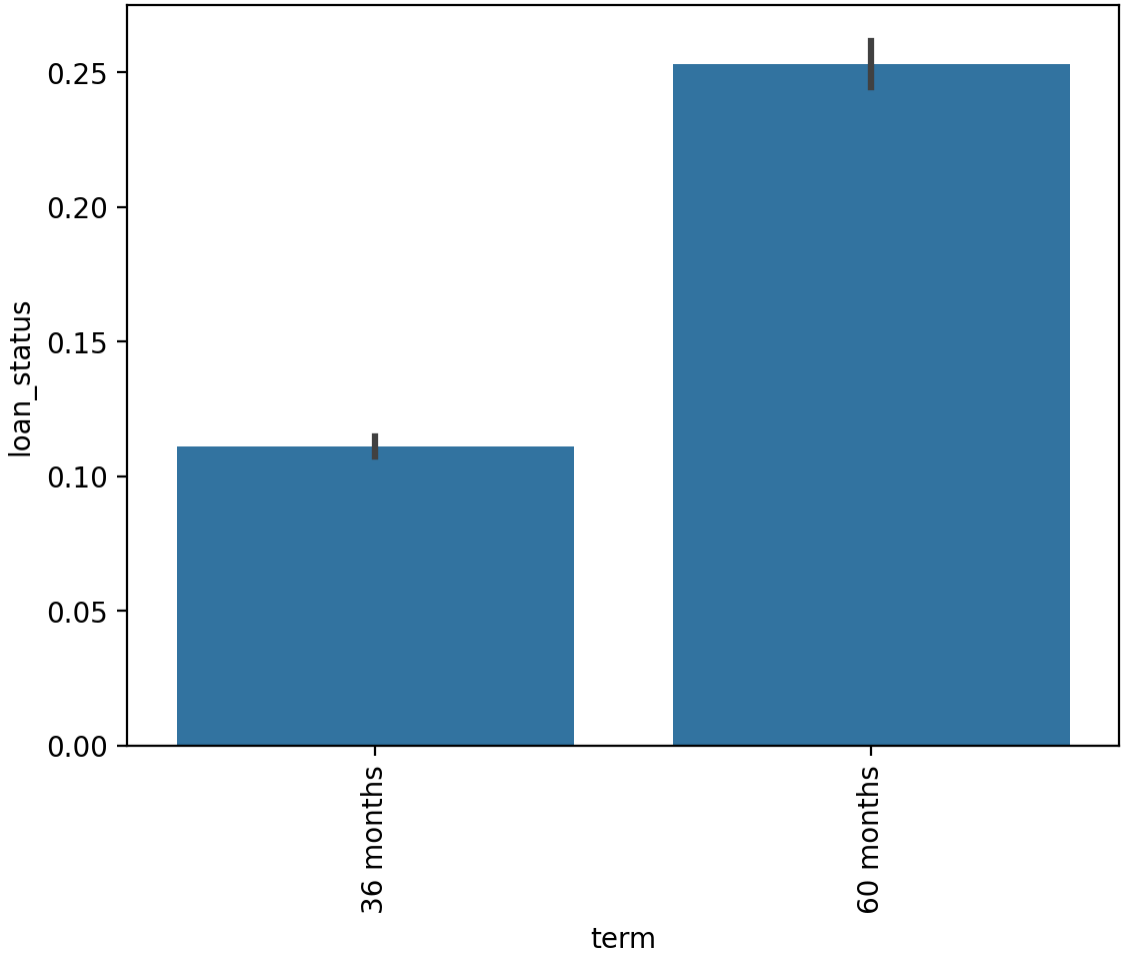


Debt consolidation is the most common reason for loan applications.

Bi-Variate Analysis

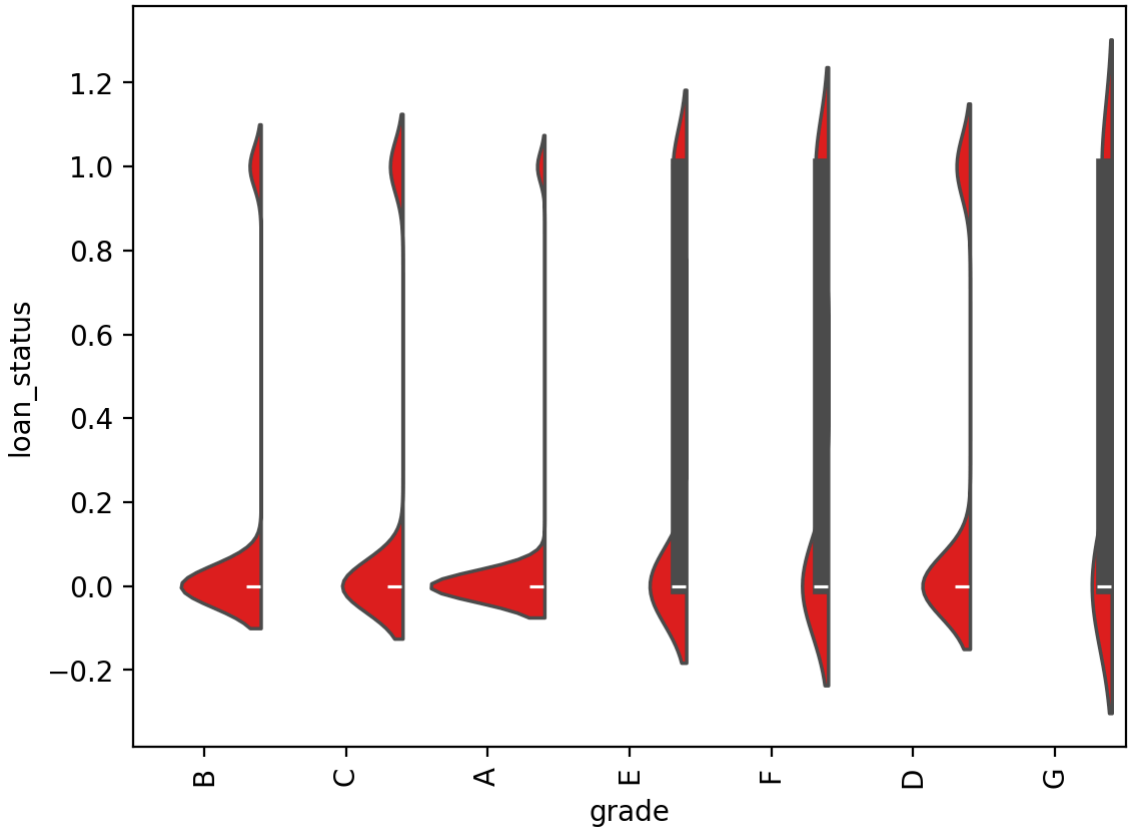
In [58]:
```
# lets define a function to plot loan_status across categorical variables

def plot_cat(cat_var):
    sns.barplot(x=cat_var, y="loan_status",data=new_data)
    plt.xticks(rotation="vertical")
    plt.show()

def plot_cat2(cat_var2):
    sns.violinplot(x= cat_var2,y="loan_status",data=new_data,color="red",split=T
    plt.xticks(rotation="vertical")
    plt.show()
```

In [59]:
```
# compare default rates across grade of loan

plot_cat("term")
```



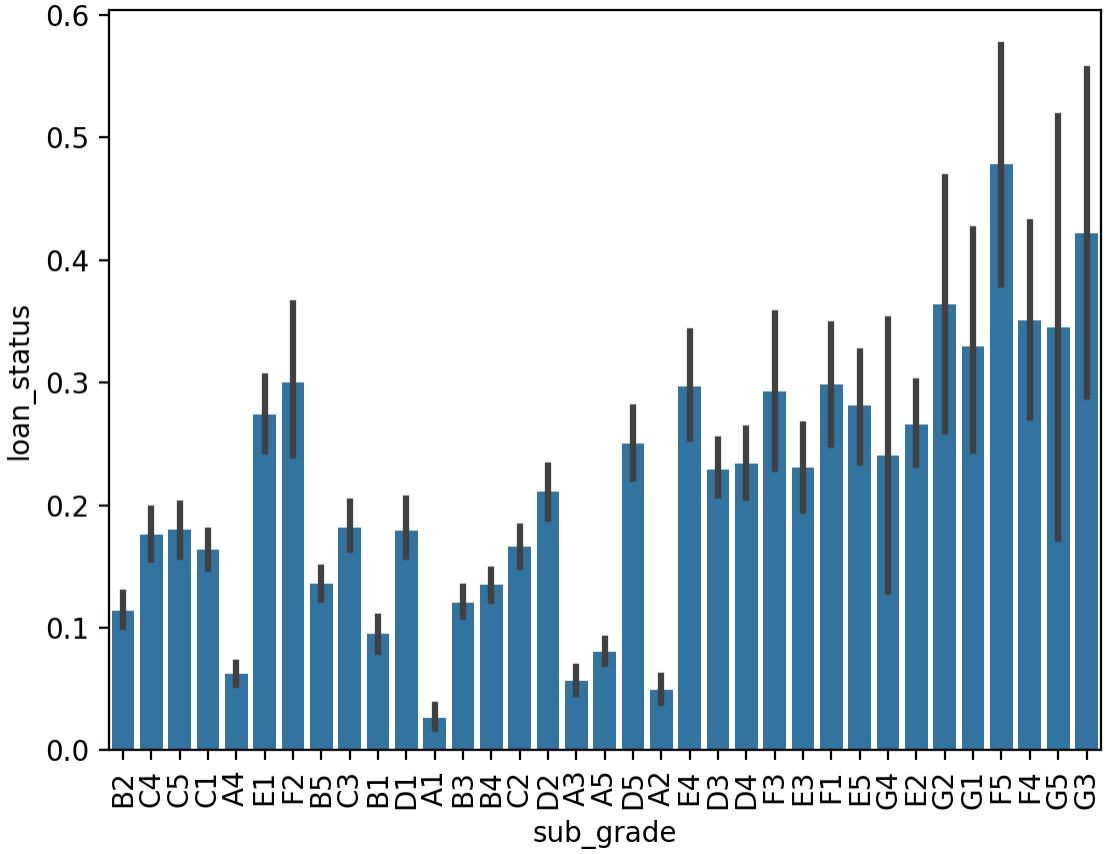
Most of the defaulters among the majority of applicants had selected a 60-month loan term.

In [60]:
```
# compare default rates across grade of loan

plot_cat2("grade")
```



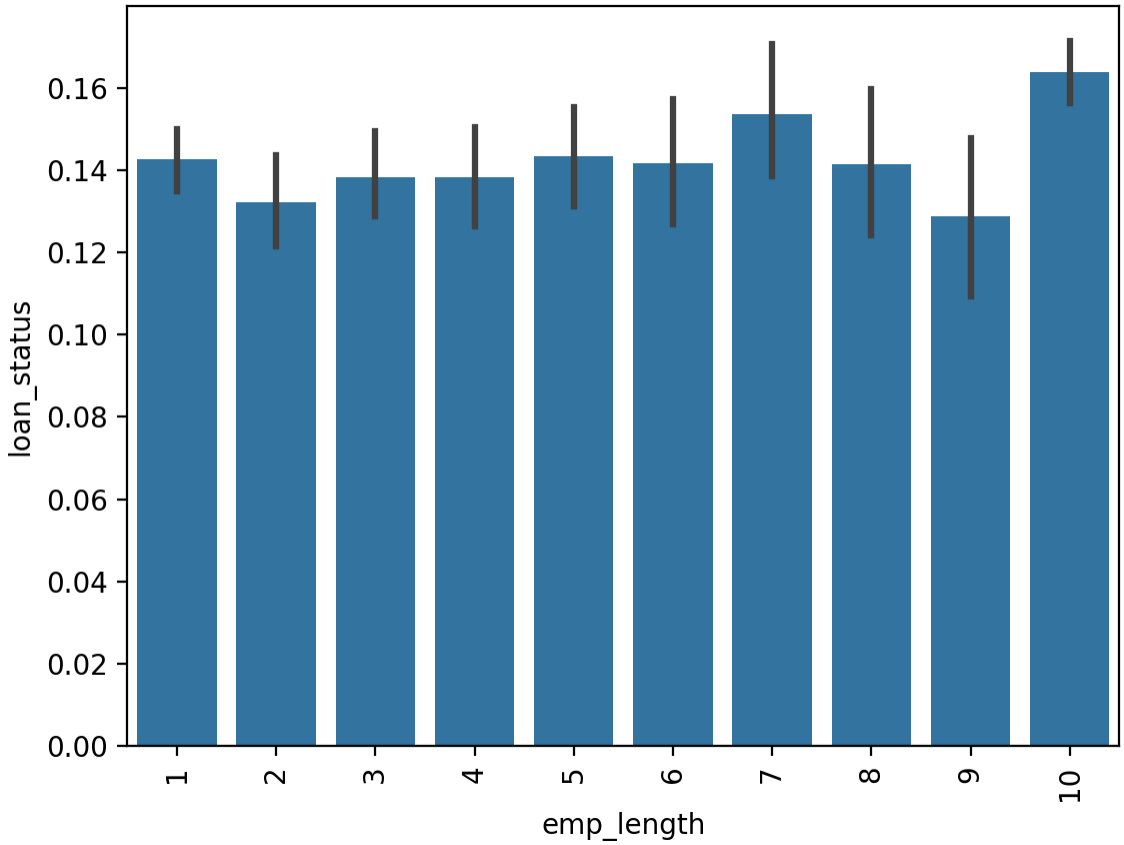
Applicants with Grade E,F & G were more likely to default, despite being in the majority category.

In [61]:
```
# compare default rates across grade of loan

plot_cat("sub_grade") # here we dont consider their subgrade for the analysis pa
```



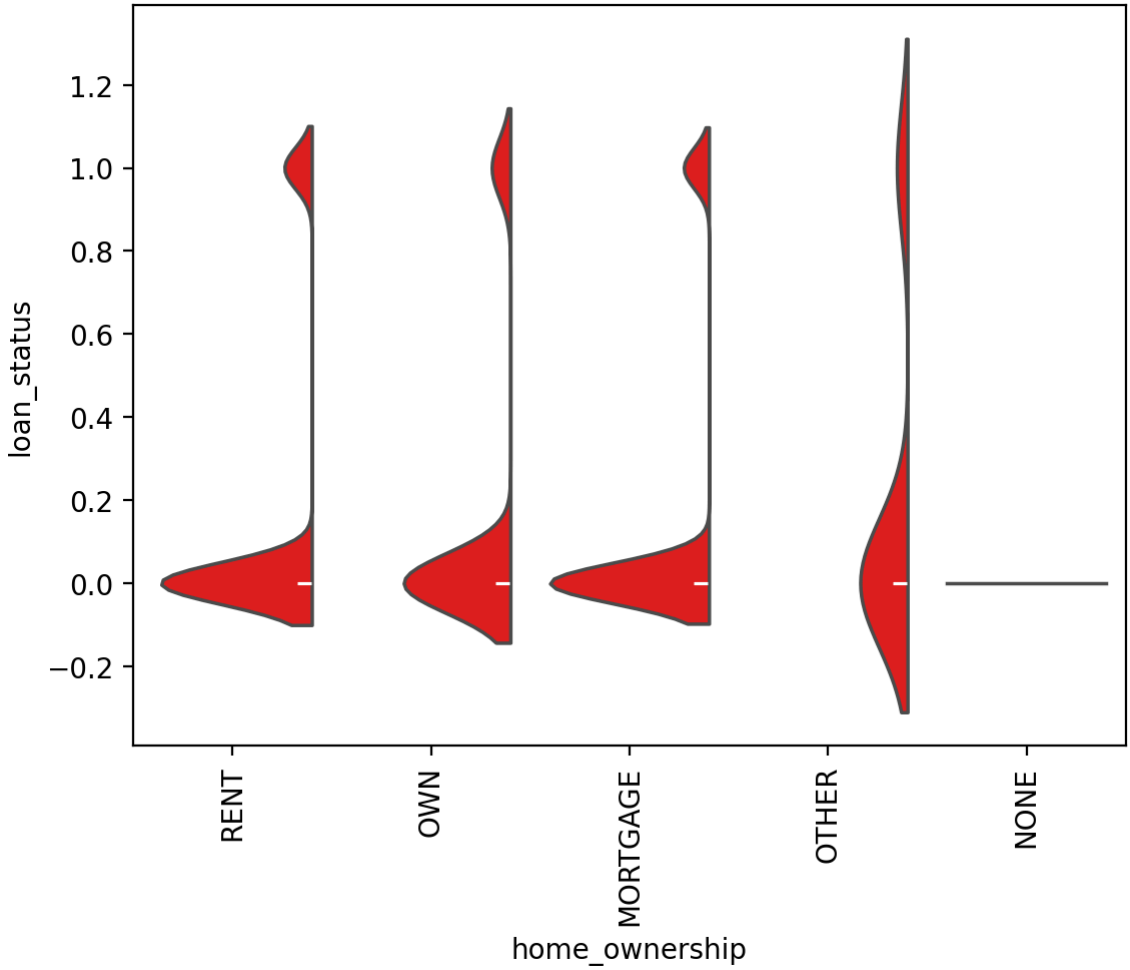
Sub-grades F5 & G3 saw a notable number of defaults among the most common applicants.

In [62]:
```
# compare default rates across grade of loan

plot_cat("emp_length")
```



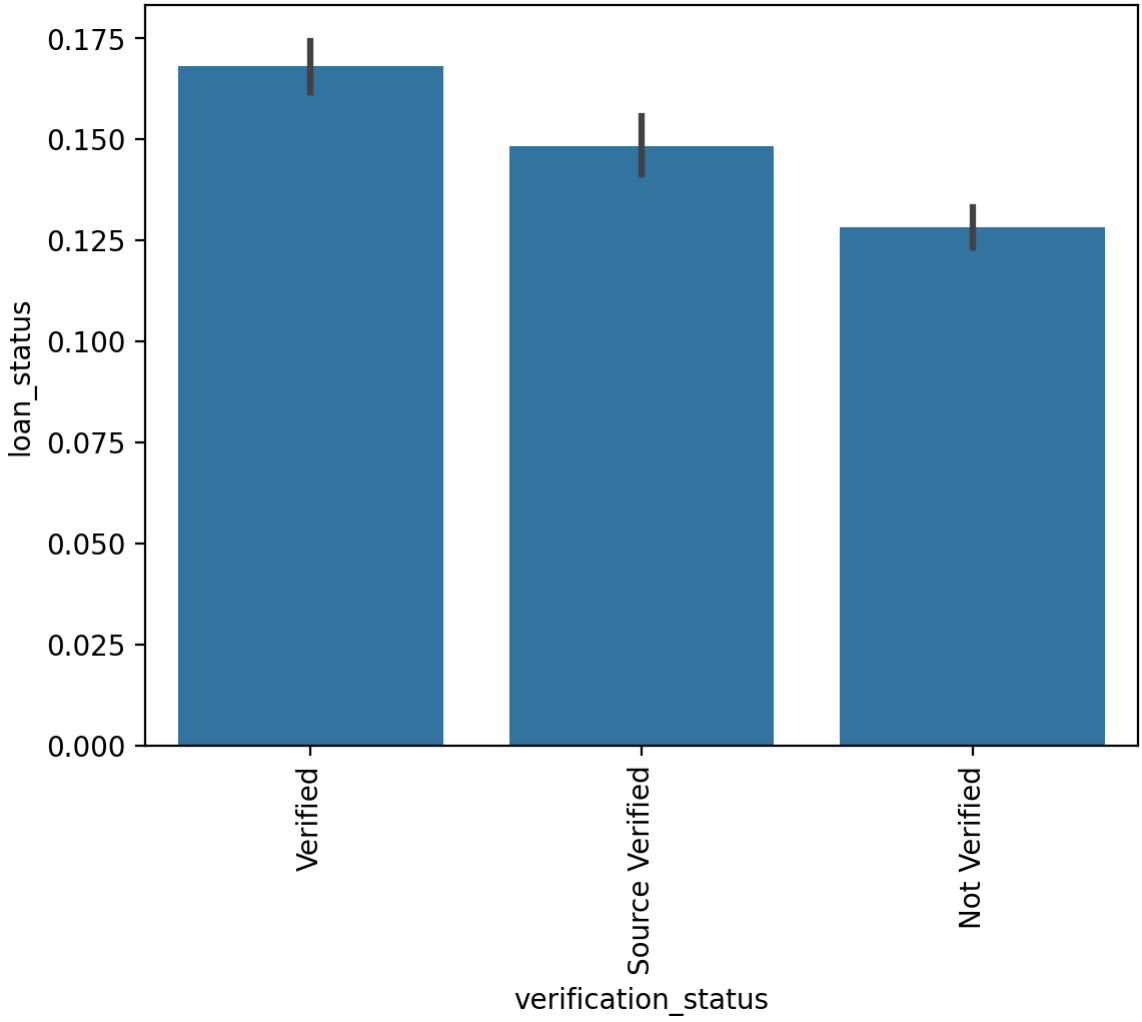
Individuals with 7 & 10 years Means Experts and Fresher of employment were among the key defaulters in the dataset.

In [63]:
```
# compare default rates across grade of loan

plot_cat2("home_ownership")
```



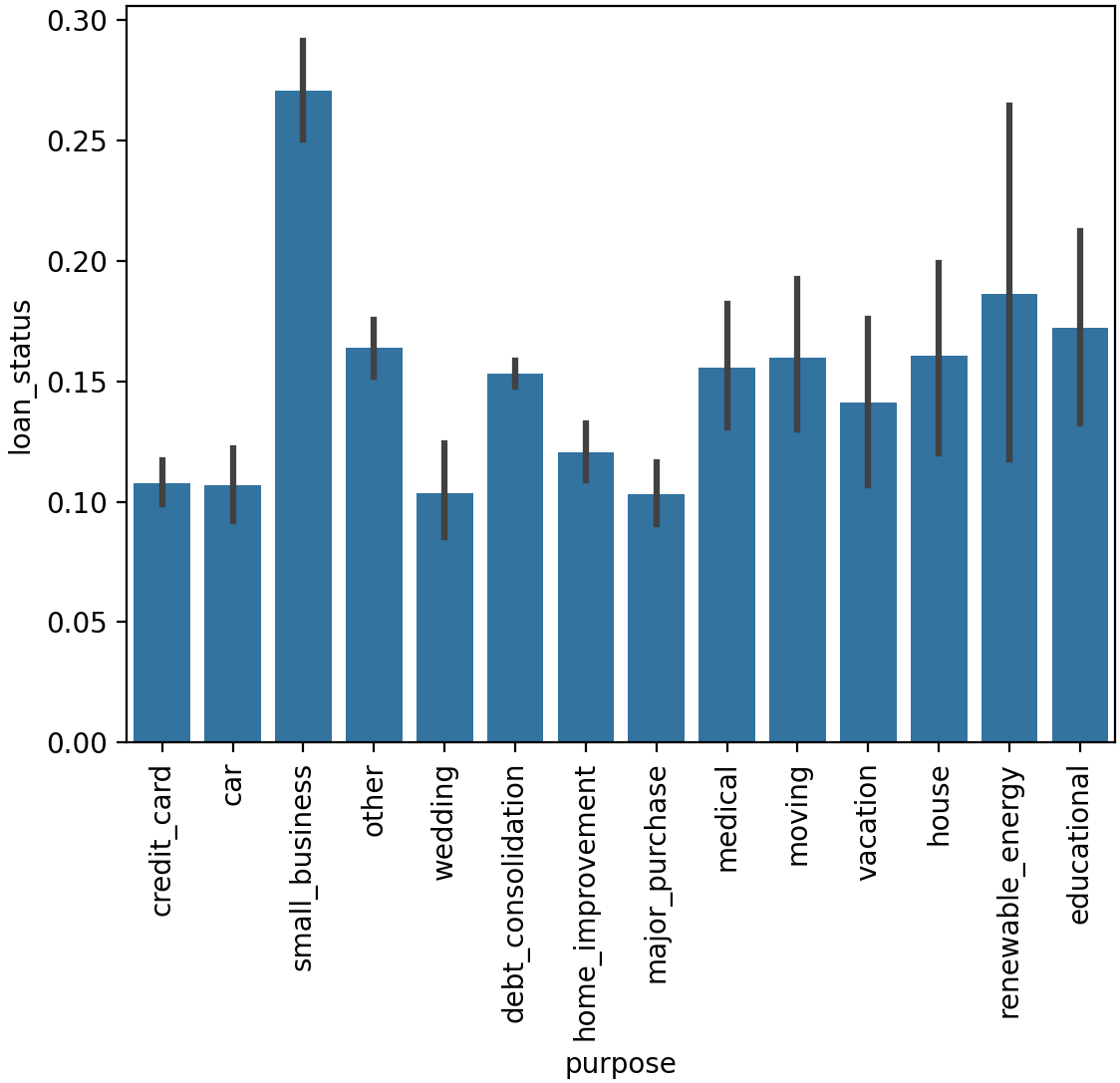
Most applicants had a other on their home and get defaults.

In [64]:
```
# compare default rates across grade of loan

plot_cat("verification_status")
```



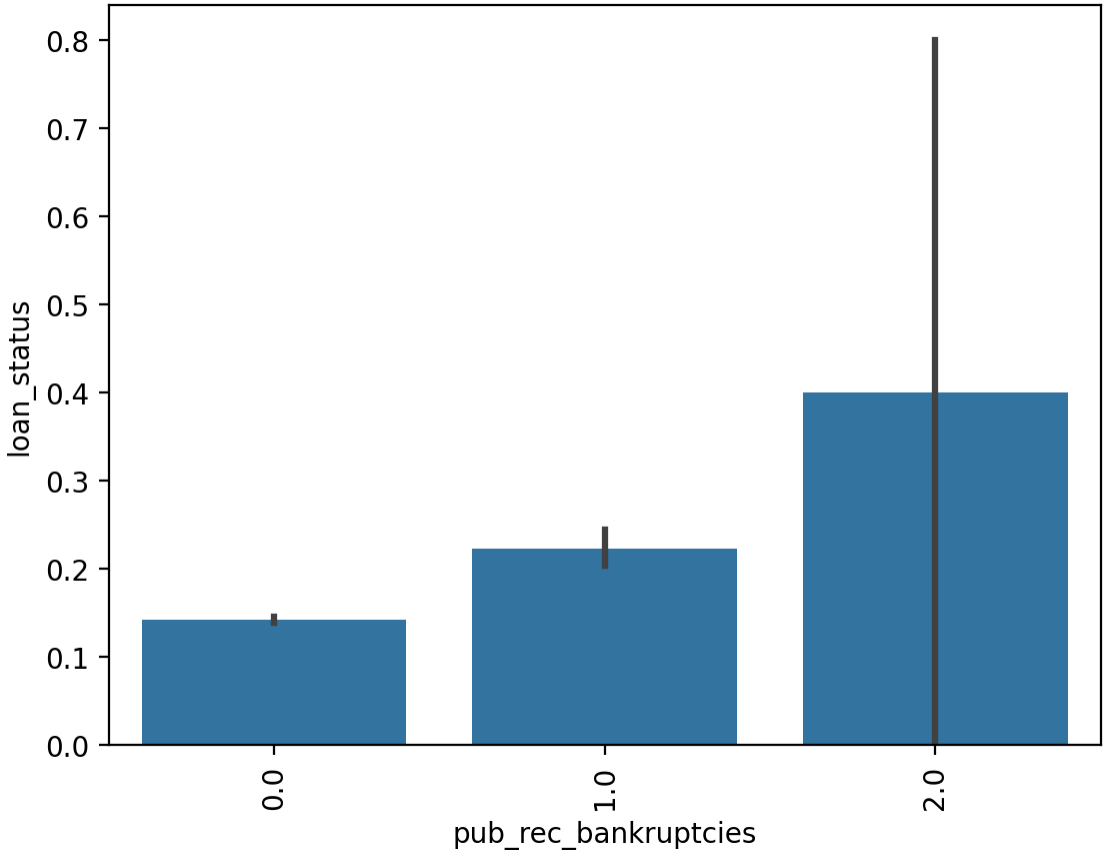
Applicants whose income was verified were more likely to default.

In [65]:
```
# compare default rates across grade of loan

plot_cat("purpose")
```



Most of the applicants get Defaults were higher among those who borrowed for small business purpose.

In [66]:
```
# compare default rates across grade of loan

plot_cat("pub_rec_bankruptcies")
```



Interestingly, many defaulters had 1 & 2 public record of bankruptcy.

In [67]:
```
# let's also observe the distribution of loans across years
# first lets convert the year column into datetime and then extract year and mon
new_data['issue_d'].head()
```

Out[67]:
```
0    Dec-11
1    Dec-11
2    Dec-11
3    Dec-11
5    Dec-11
Name: issue_d, dtype: object
```

In [68]:
```
from datetime import datetime
new_data['issue_d'] = new_data['issue_d'].apply(lambda x: datetime.strptime(x, '
```

In [69]:
```
# extracting month and year from issue_date
new_data['month'] = new_data['issue_d'].apply(lambda x: x.month)
new_data['year'] = new_data['issue_d'].apply(lambda x: x.year)
```

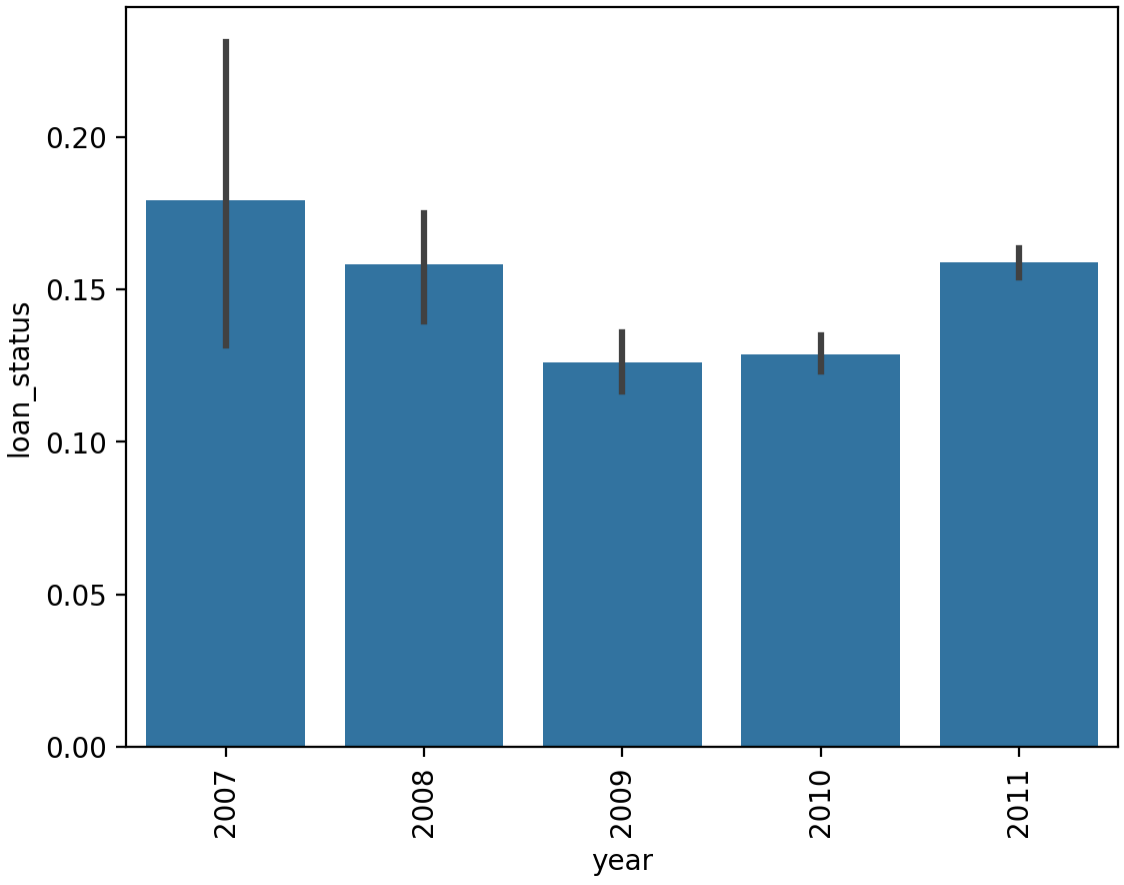
In [70]:
```
# let's first observe the number of loans granted across years
new_data.groupby('year').year.count()
```

Out[70]:
```
year
2007      251
2008     1562
2009     4716
2010    11532
2011    20516
Name: year, dtype: int64
```

In [71]:
```
# number of loans across months
new_data.groupby('month').month.count()
```

Out[71]:
```
month
1     2379
2     2358
3     2691
4     2831
5     2919
6     3180
7     3351
8     3388
9     3498
10    3761
11    4006
12    4215
Name: month, dtype: int64
```

In [72]:
```
# lets compare the default rates across years
# the default rate had suddenly increased in 2011, inspite of reducing from 2008
plot_cat('year')
```



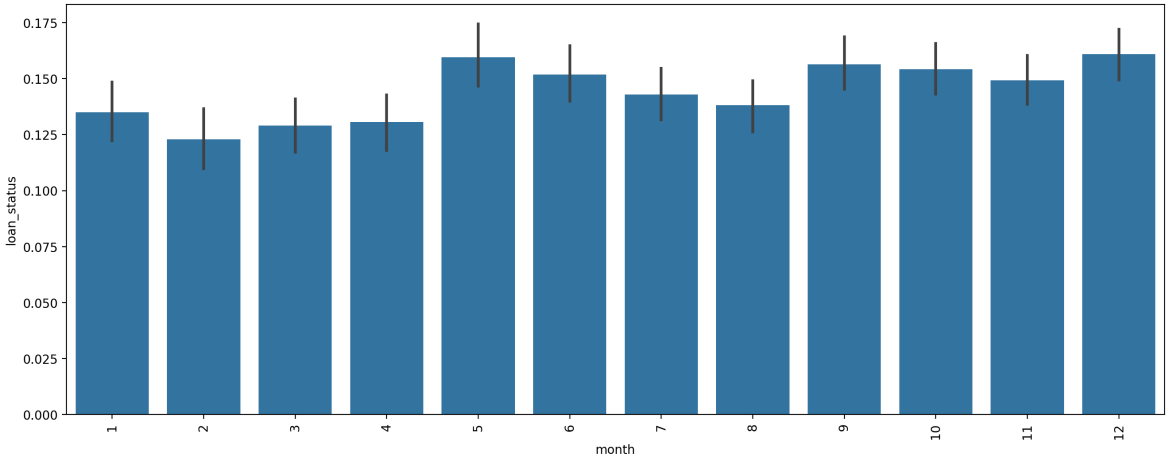
Most of the applicants are fail to pay their loan on or before due time in the year of 2007 & 2008.

2008 to 2009 Year by Year defaulters is low.

Again 2010 applicants get as defaulters are higher by year.

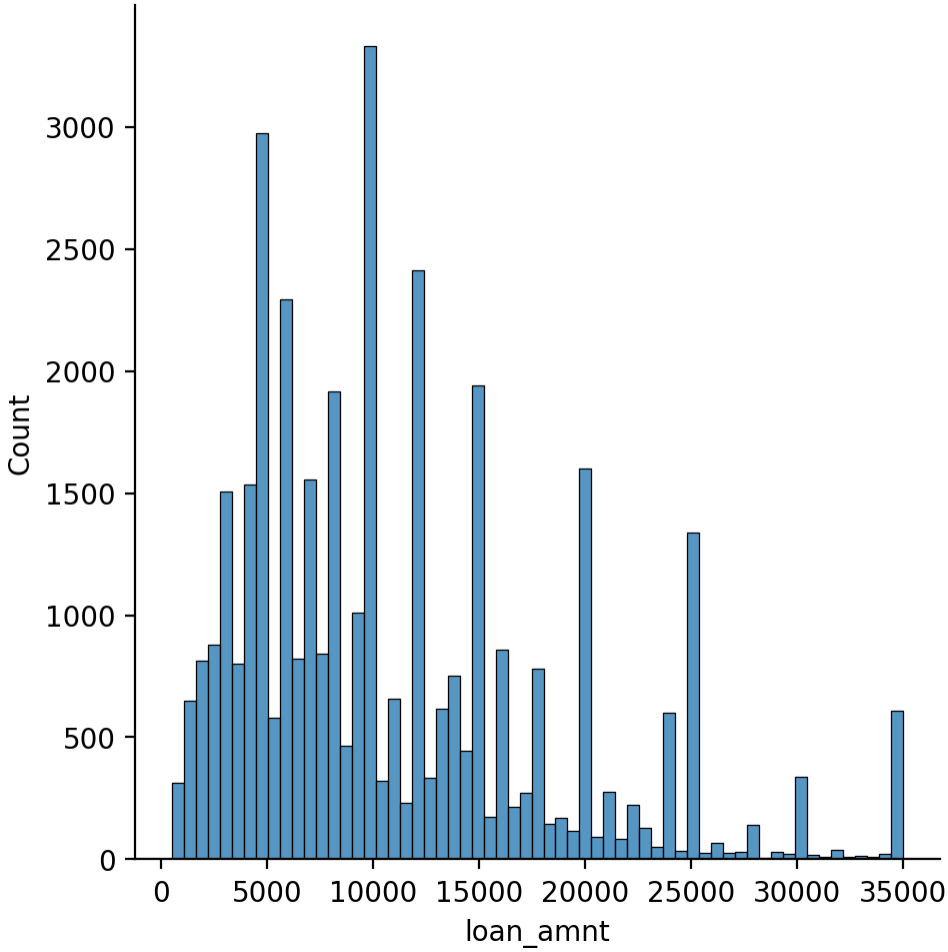
Defaults were common among loans issued in the years 2007 and 2008.

In [73]:
```
# comparing default rates across months: not much variation across months
plt.figure(figsize=(16, 6))
plot_cat('month')
```



A considerable number of defaults occurred in most of the applicants for loans issued in the month of May.

In [74]:
```
# loan amount: the median loan amount is around 10,000
sns.displot(new_data['loan_amnt'])
plt.show()
```



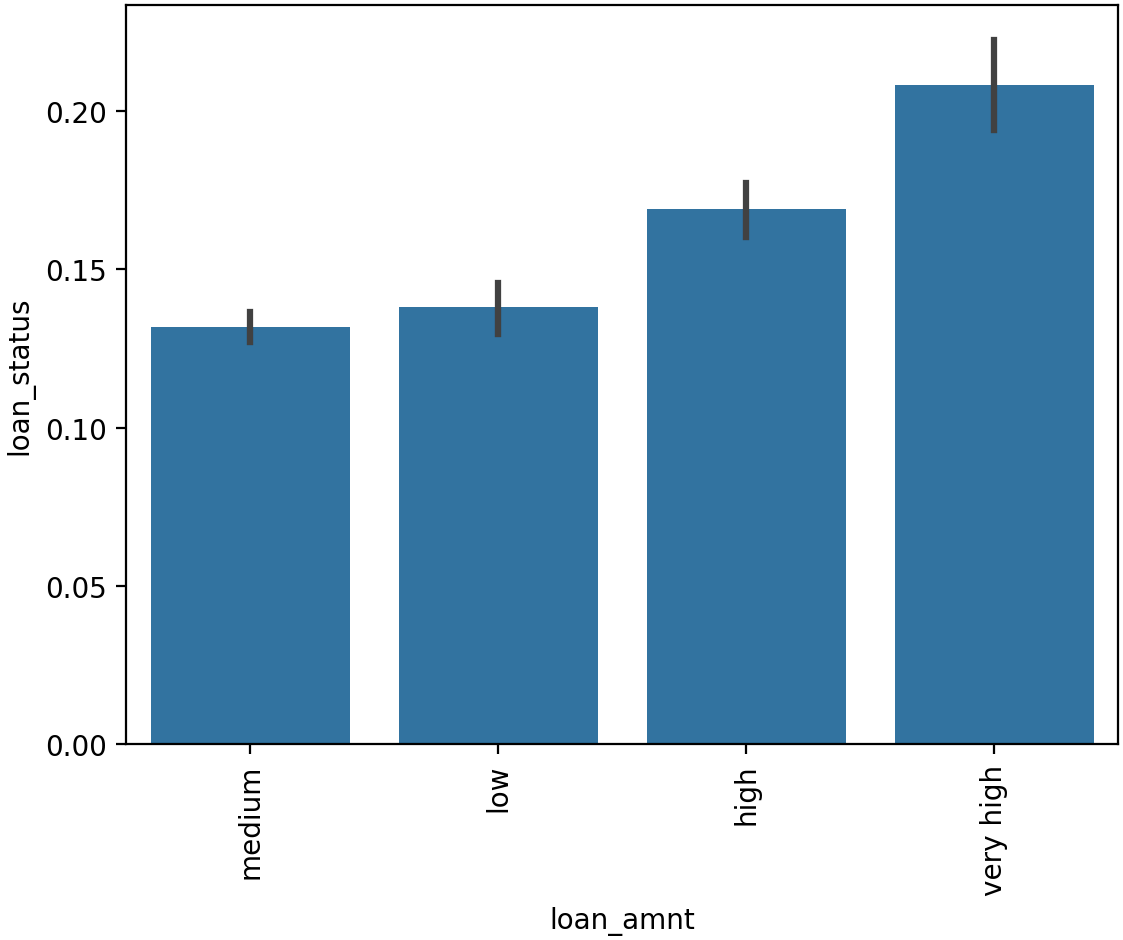
Most of the applicants wants to take a loan amount in the range 5000 to 15000 rupees.

In [75]:
```
# categorise the loan amount

def loan_amount(n):
    if n < 5000:
        return 'low'
    elif n >=5000 and n < 15000:
        return 'medium'
    elif n >= 15000 and n < 25000:
        return 'high'
    else:
        return 'very high'

new_data['loan_amnt'] = new_data['loan_amnt'].apply(lambda x: loan_amount(x))
```

In [76]:
```
# caunt the loan amount category
new_data['loan_amnt'].value_counts()
```

Out[76]:
```
loan_amnt
medium       20675
high          7696
low           7444
very high     2762
Name: count, dtype: int64
```

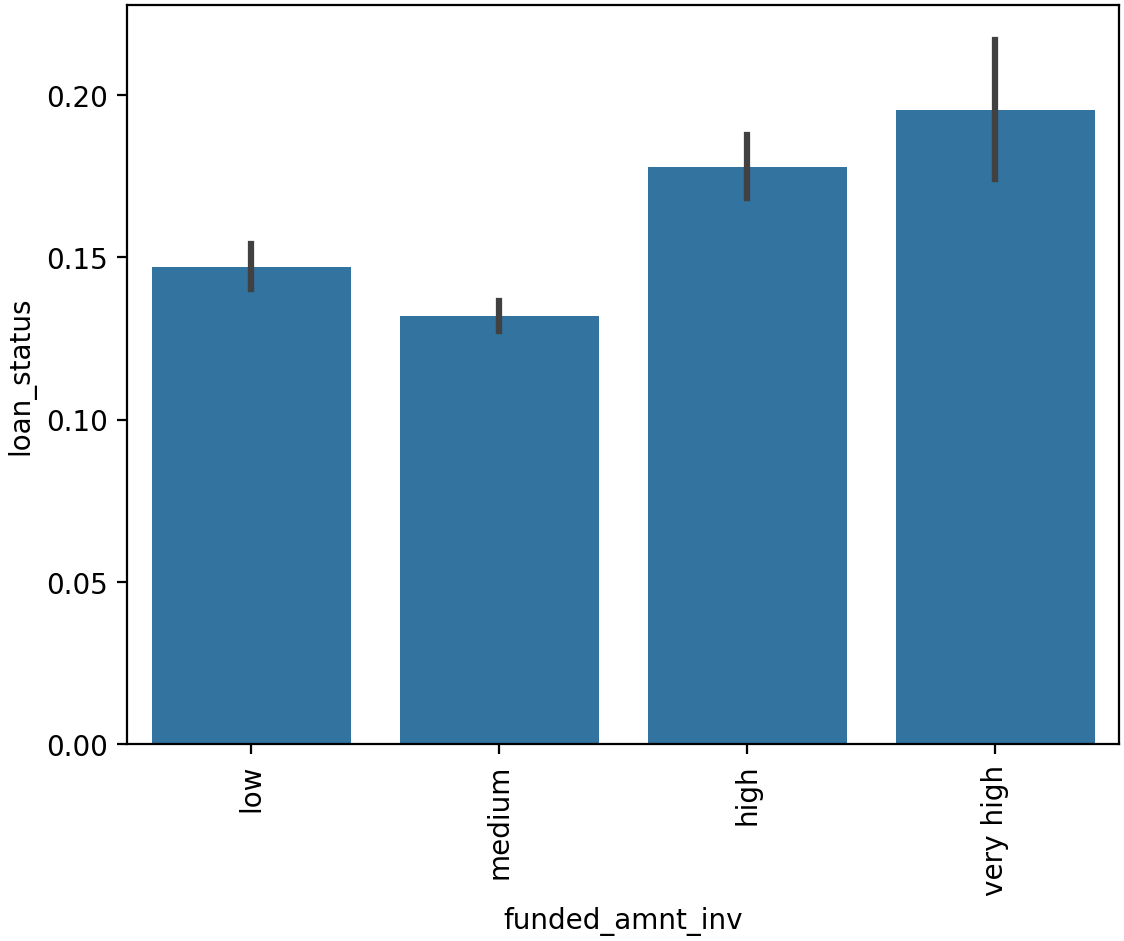
In [77]:
```
# let's compare the default rates across loan amount type
# higher the loan amount, higher the default rate
plot_cat('loan_amnt')
```



Most of the applicants get Defaults often had taken loans in the very high amount range.

In [78]:
```
# let's also convert funded amount invested to bins
new_data['funded_amnt_inv'] = new_data['funded_amnt_inv'].apply(lambda x: loan_a
```

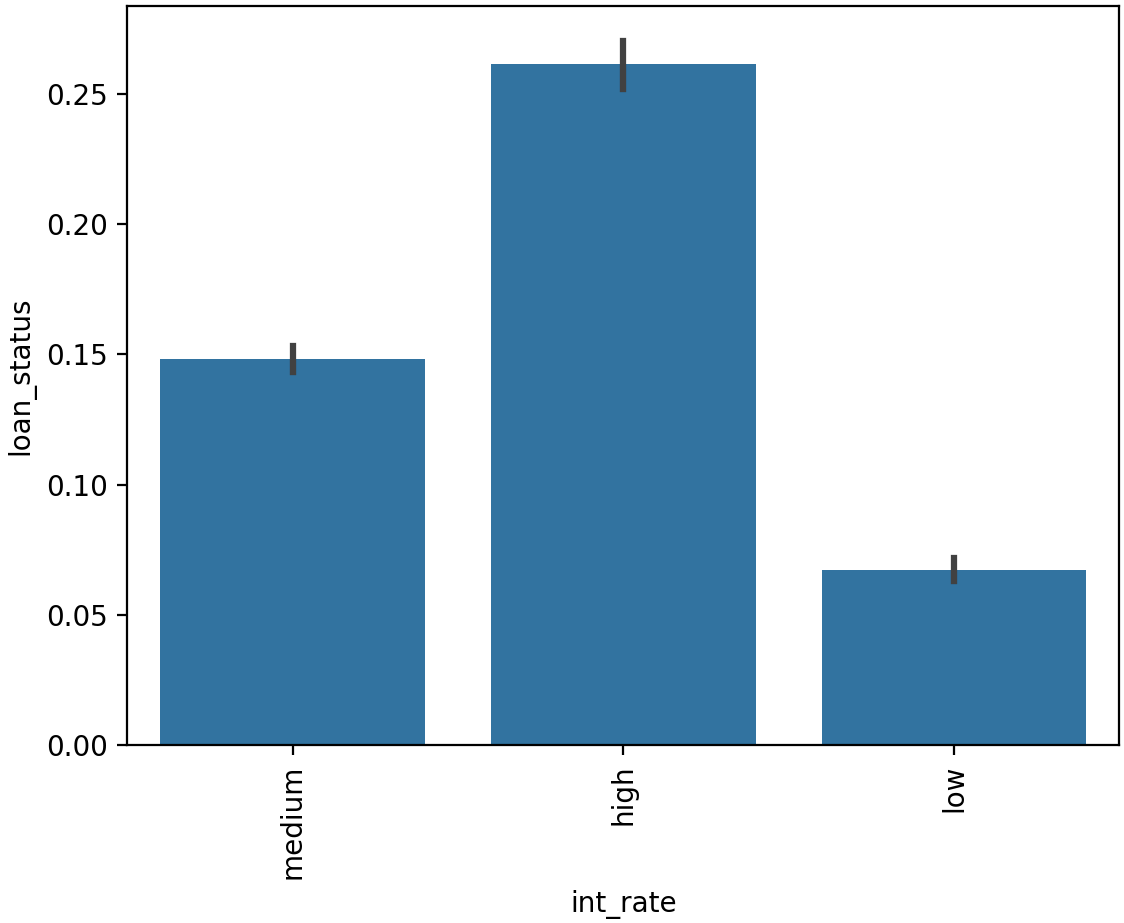
In [79]:
```
# funded amount invested
plot_cat('funded_amnt_inv')
```



Most of the applicants get defaults Defaults were frequent in loans with a very high level of funded amount.

In [80]:
```
# lets also convert interest rate to low, medium, high
# binning loan amount

def int_rate(n):
    if n <= 10:
        return 'low'
    elif n > 10 and n <=15:
        return 'medium'
    else:
        return 'high'


new_data['int_rate'] = new_data['int_rate'].apply(lambda x: int_rate(x))
```

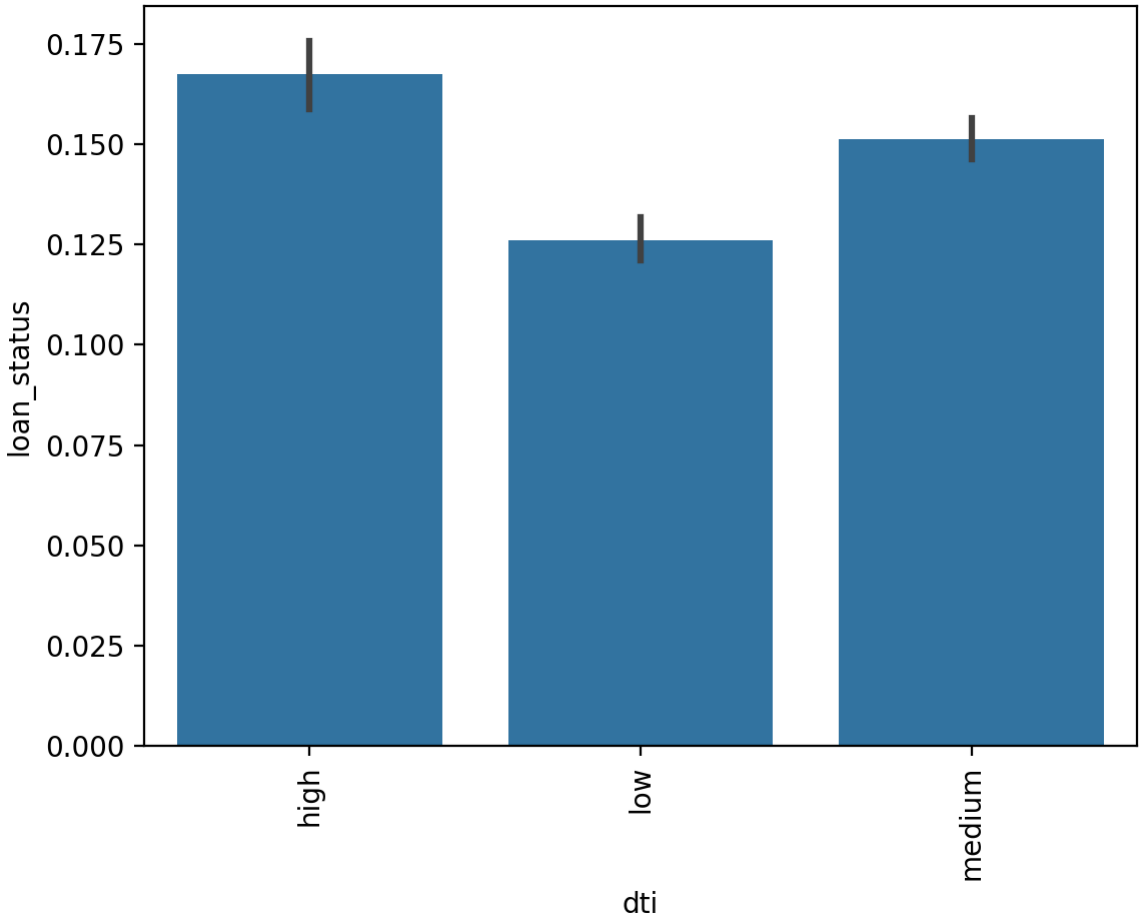
In [81]:
```
# loan status by loan rates
plot_cat('int_rate')
```



Surprisingly, applicants with high interest rate & medium interest rates also defaulted.

In [82]:
```
# debt to income ratio that means whose income vs interest ratio is high/low/ver

def dti(n):
    if n <= 10:
        return 'low'
    elif n > 10 and n <=20:
        return 'medium'
    else:
        return 'high'


new_data['dti'] = new_data['dti'].apply(lambda x: dti(x))
```

In [83]:
```
# compare the debt vs loan_status

plot_cat("dti")
```

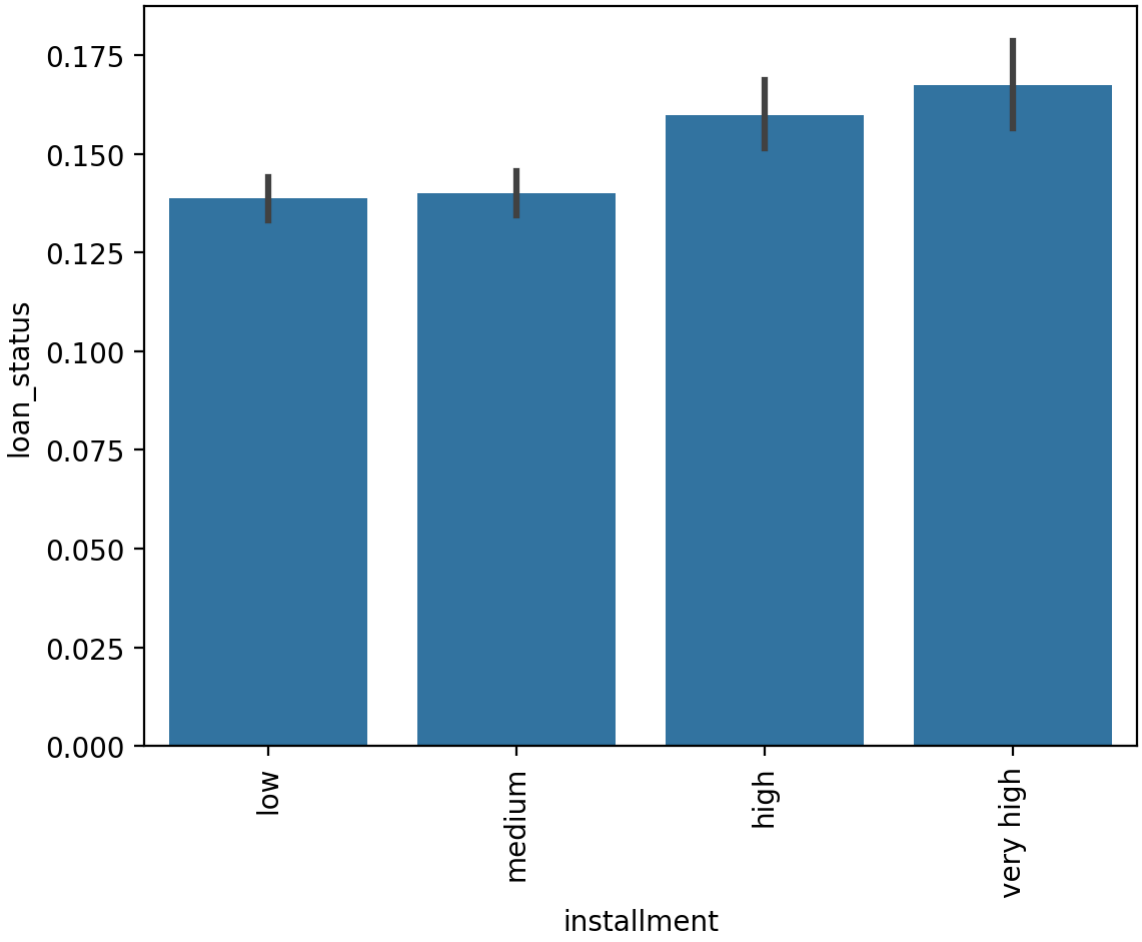


Even most of the applicants with a high Debt-to-Income (DTI) ratio showed defaults.

In [84]:
```
# funded amount

def funded_amount(n):
    if n <= 5000:
        return 'low'
    elif n > 5000 and n <=15000:
        return 'medium'
    else:
        return 'high'

new_data['funded_amnt'] = new_data['funded_amnt'].apply(lambda x: funded_amount(
```

In [85]:
```
# installment

def installment(n):
    if n <= 200:
        return 'low'
    elif n > 200 and n <=400:
        return 'medium'
    elif n > 400 and n <=600:
        return 'high'
    else:
        return 'very high'

new_data['installment'] = new_data['installment'].apply(lambda x: installment(x)
```

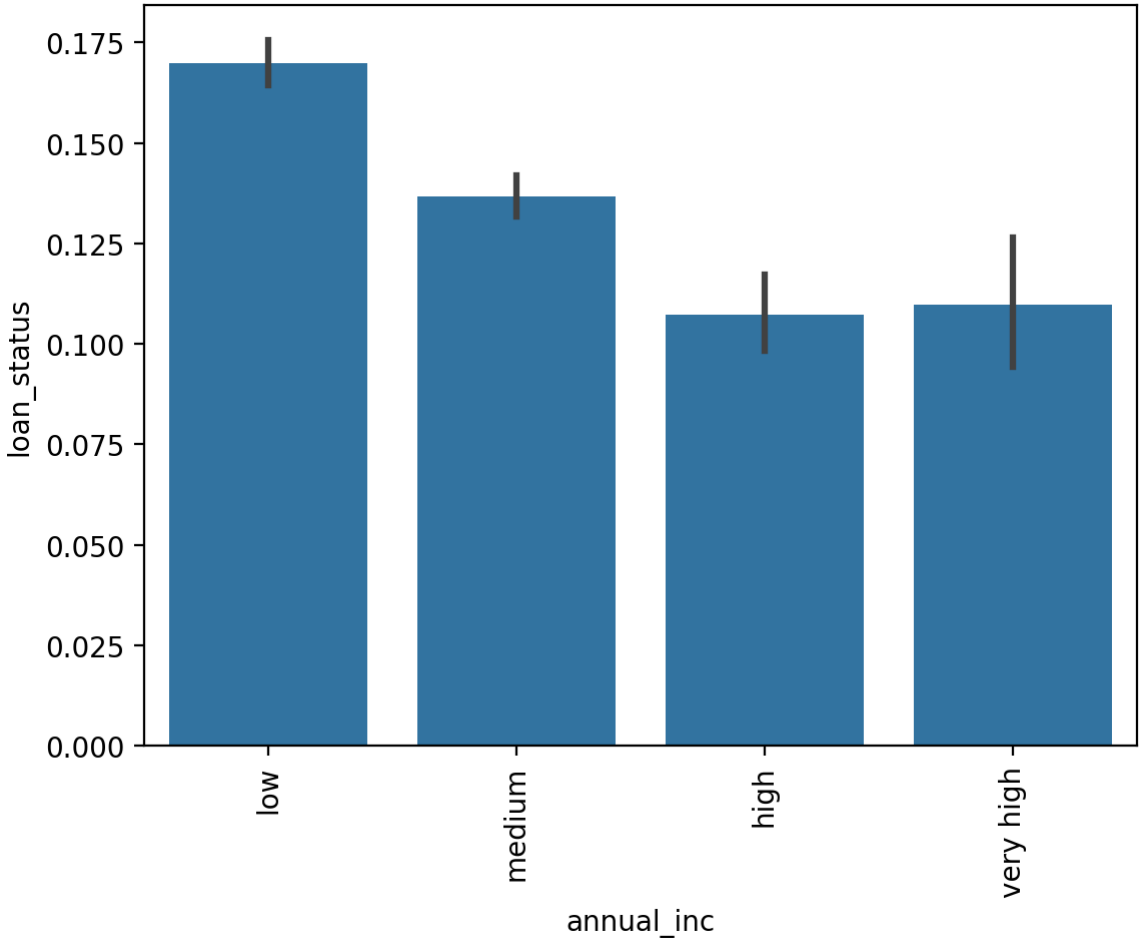
In [86]:
```
# compare the default loan rates across the installments
# check the installment wise loan status

plot_cat("installment")
```



Even most of the applicants with a very high installments showed higher defaults.

In [87]:
```
# annual income

def annual_income(n):
    if n <= 50000:
        return 'low'
    elif n > 50000 and n <=100000:
        return 'medium'
    elif n > 100000 and n <=150000:
        return 'high'
    else:
        return 'very high'

new_data['annual_inc'] = new_data['annual_inc'].apply(lambda x: annual_income(x))
```

In [88]:
```
# check the income wise default rates
plot_cat("annual_inc")
```



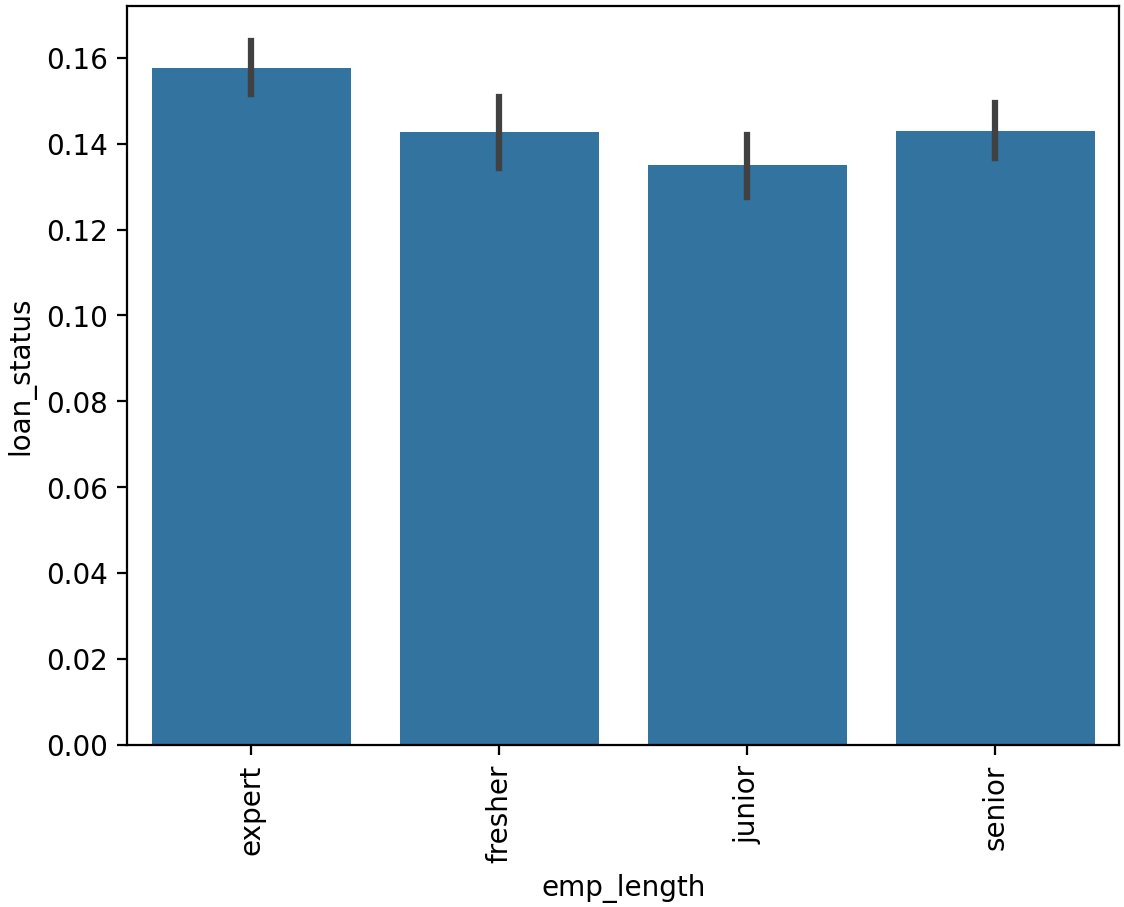
Even most of the applicants with a Low annual income showed higher defaults.

In [89]:
```
new_data["emp_length"].isnull().sum()
```

Out[89]:
```
0
```

In [90]:
```
# employee length numerical column to category column

def emp_length(n):
    if n <= 1:
        return 'fresher'
    elif n > 1 and n <=3:
        return 'junior'
    elif n > 3 and n <=7:
        return 'senior'
    else:
        return 'expert'

new_data['emp_length'] = new_data['emp_length'].apply(lambda x: emp_length(x))
```

In [91]:
```
# let's check which category of the employee length is default

plot_cat("emp_length")
```



In the most of the Applicants the Defaults were more noticeable among expert-level employees.

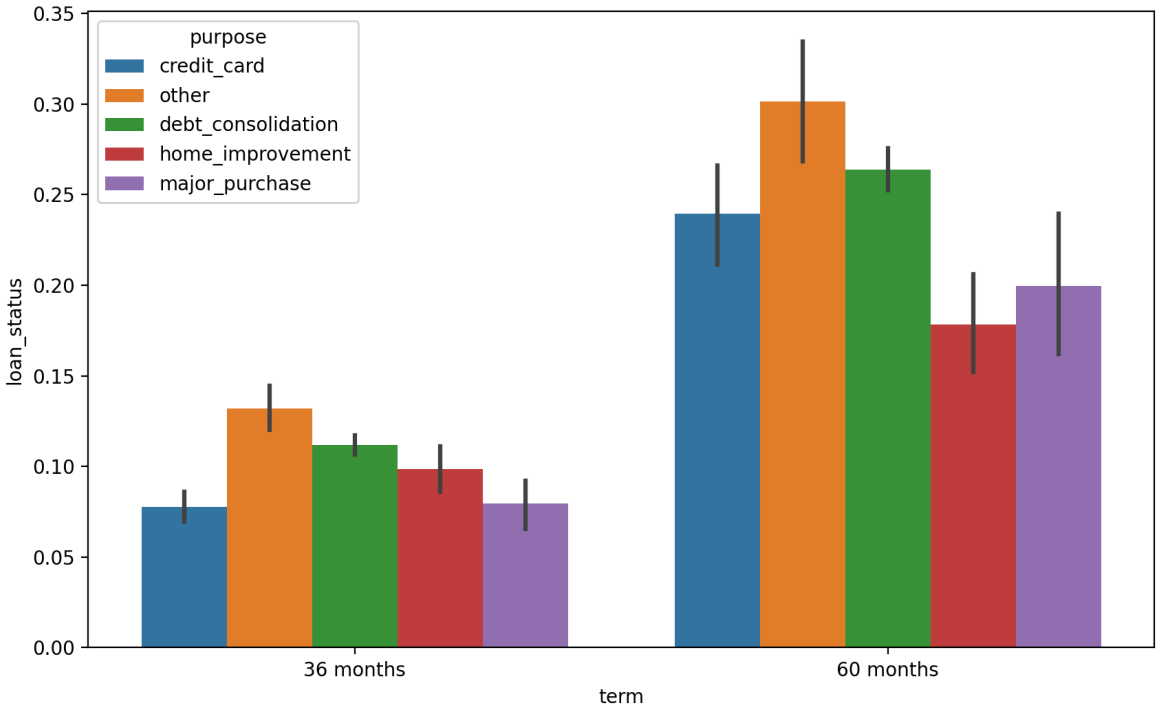
In [92]:
```
# now i filterout the 5 major purposes for loan appicants

main_purposes = ["credit_card","debt_consolidation","home_improvement","major_pu
new_data = new_data[new_data['purpose'].isin(main_purposes)]
new_data["purpose"].value_counts()
```

Out[92]:
```
purpose
debt_consolidation    18055
credit_card            5027
other                  3865
home_improvement       2875
major_purchase         2150
Name: count, dtype: int64
```

Here we can see the major loan purpose is debt_consolidation,credit-card,other,home imporovement,major-purchases.

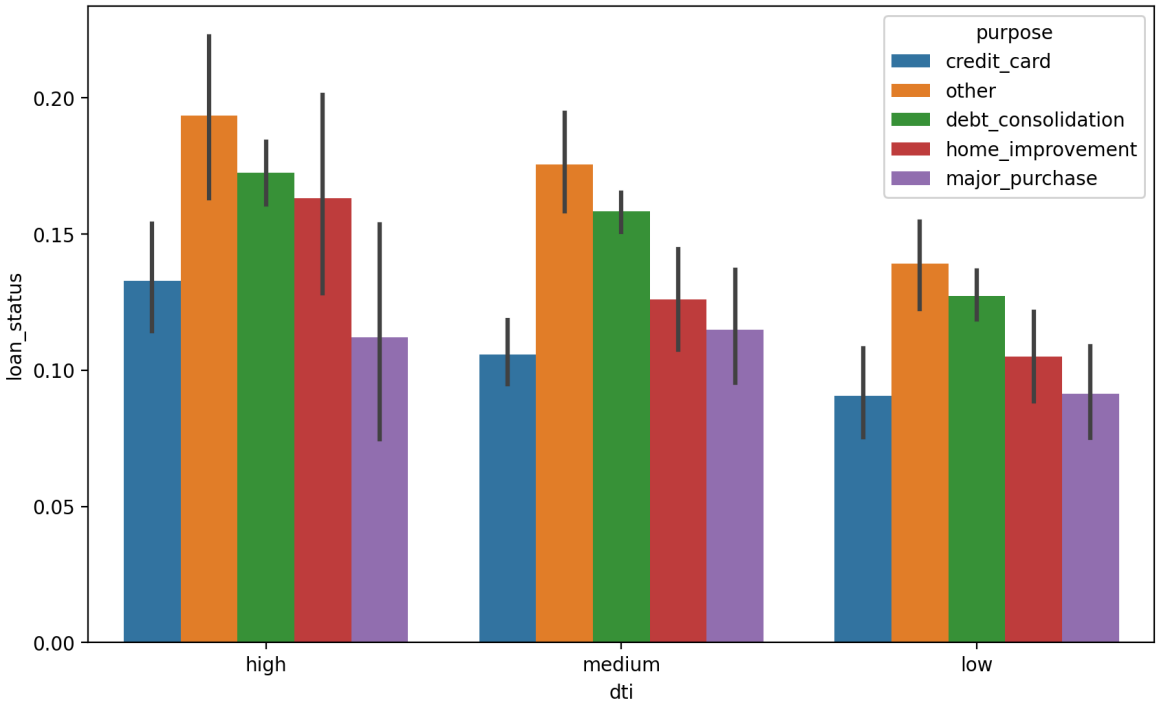
Multi-variate Analysis

In [93]:
```
# now we compare the default rates across the loan purpose and term wise

plt.figure(figsize=[10, 6])
sns.barplot(x='term', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



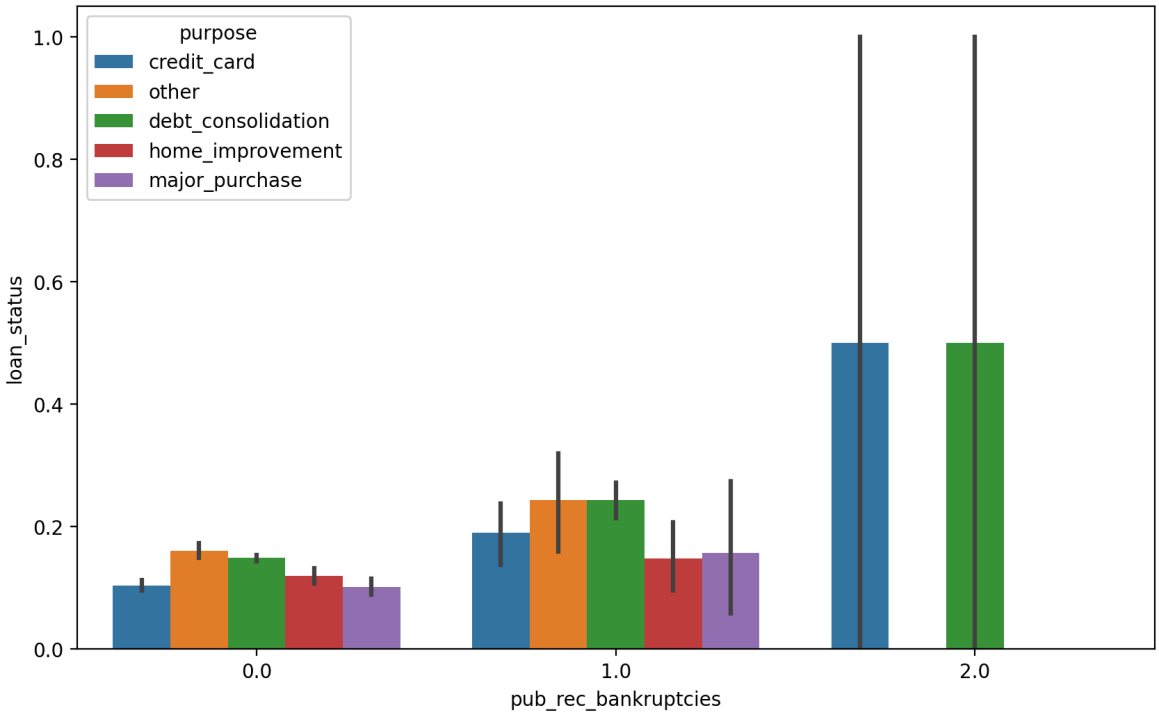
Applicants with a 60-month term and other and debt consolidation purpose had more defaults, as seen by higher values at loan status 1.

In [94]:
```
# now we compare the default rates across the loan purpose and dti wise

plt.figure(figsize=[10, 6])
sns.barplot(x='dti', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



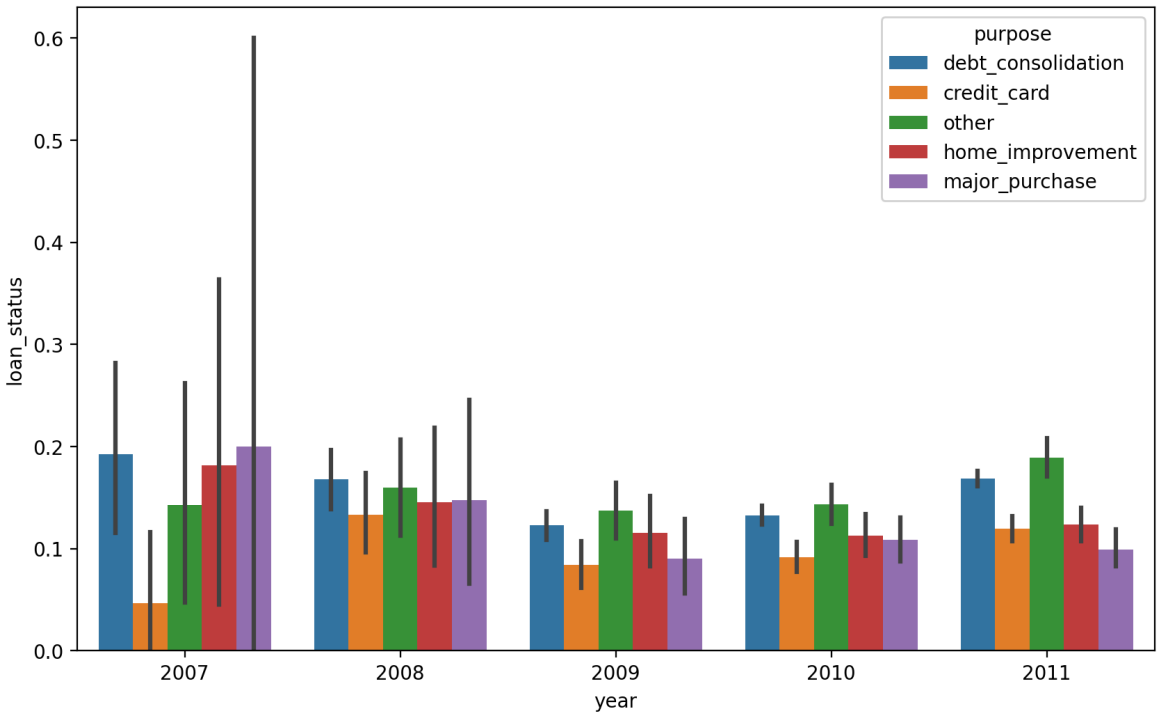
Even applicants with a high DTI ratio defaulted more frequently when the loan purpose was other,debt consolidation.

In [95]:
```
# now we compare the default rates across the loan purpose and bankruptcies wise

plt.figure(figsize=[10, 6])
sns.barplot(x='pub_rec_bankruptcies', y="loan_status", hue='purpose', data=new_d
plt.show()
```



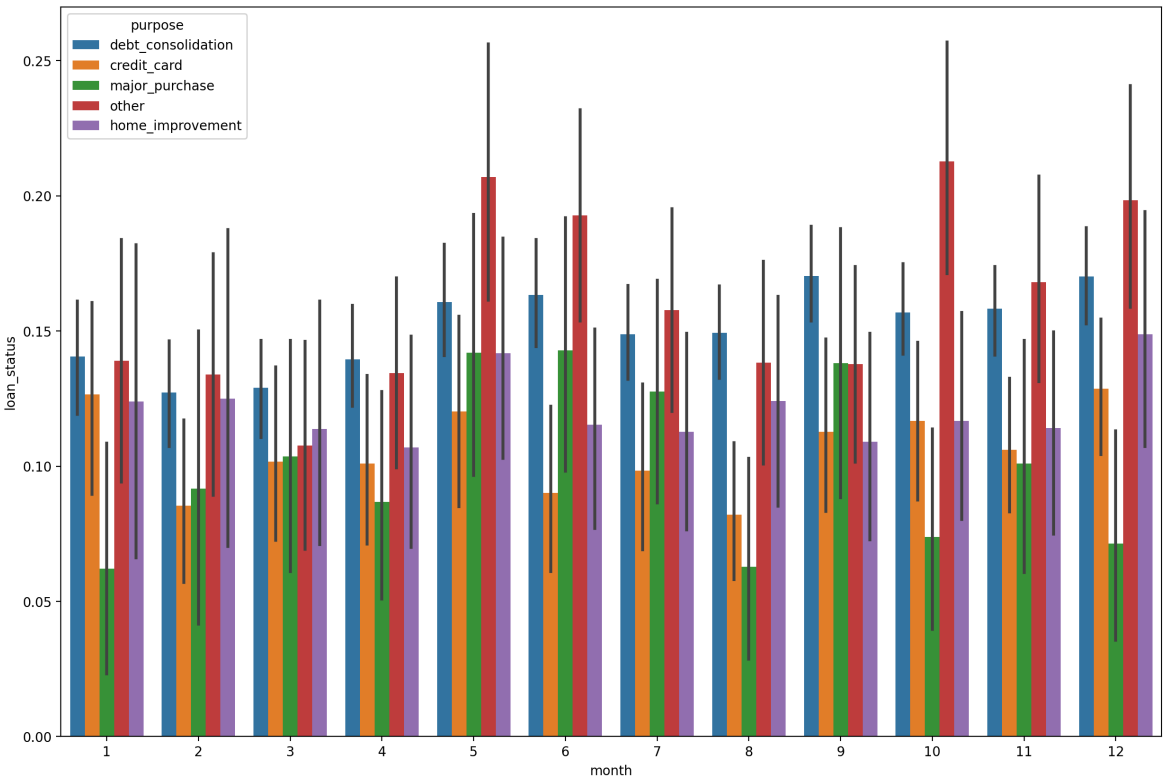
Most of the defaulters have 0 & 1 bankruptcies.

In [96]:
```
# now we compare the default rates across the loan purpose and year wise

plt.figure(figsize=[10, 6])
sns.barplot(x='year', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



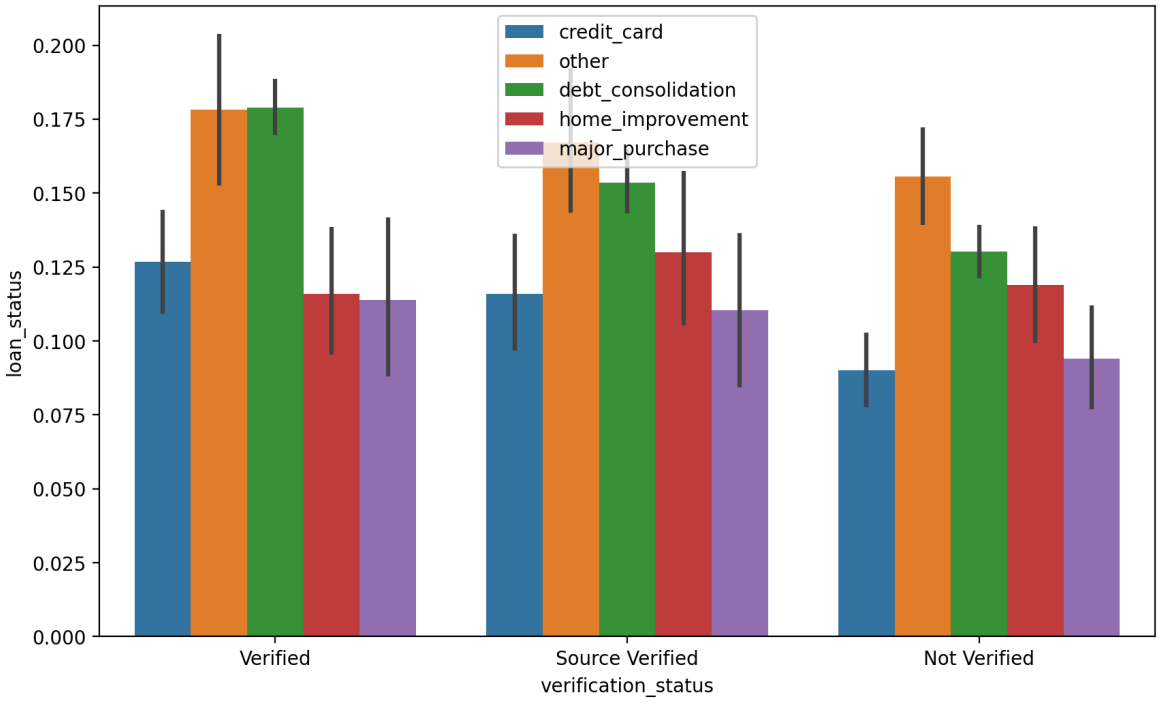
Most of the Applicants get Defaults for debt consolidation and major purchase loans were more common in the year 2007

In [97]:
```
# now we compare the default rates across the loan purpose and month wise

plt.figure(figsize=[15, 10])
sns.barplot(x='month', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



Most of the applicants get a Loans taken for major purchases in the month of May and October & December had higher charge-offs for the purpose of others.

In [98]:
```
# now we compare the default rates across the loan purpose and verification_stat

plt.figure(figsize=[10, 6])
sns.barplot(x='verification_status', y="loan_status", hue='purpose', data=new_da
plt.legend(loc='upper center')
plt.show()
```

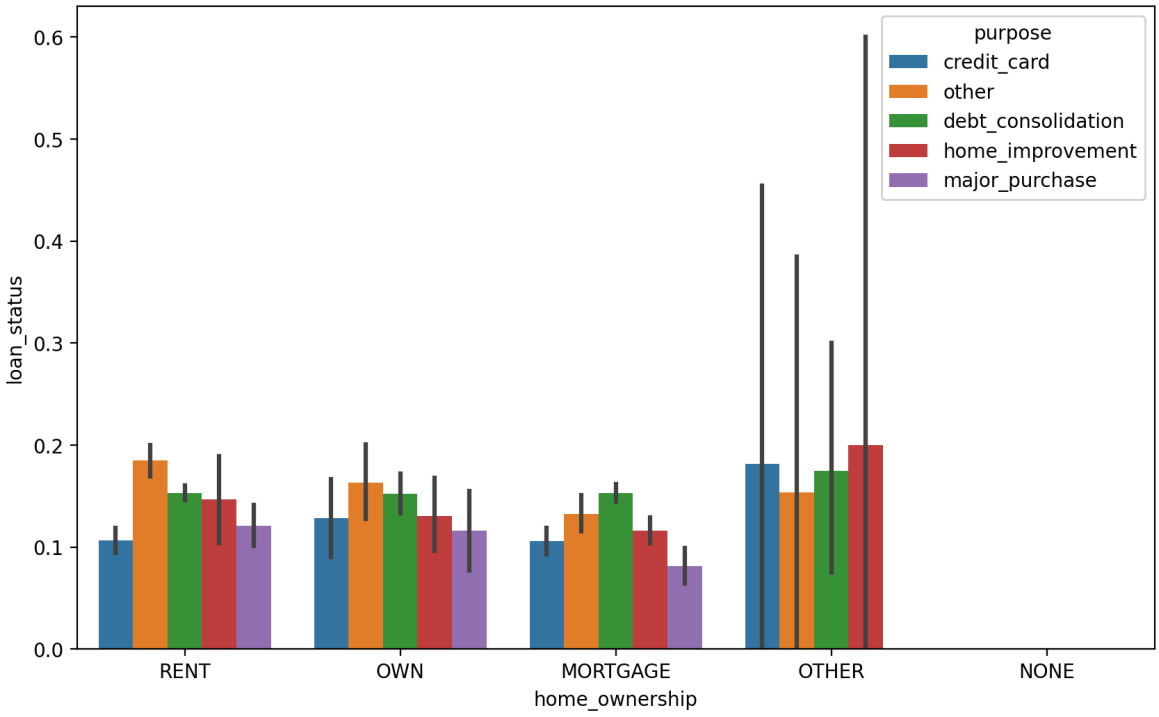


Verified applicants borrowing for other & debt consolidation purposes get defaulted more than other status.

In [99]:
```
new_data.columns
```

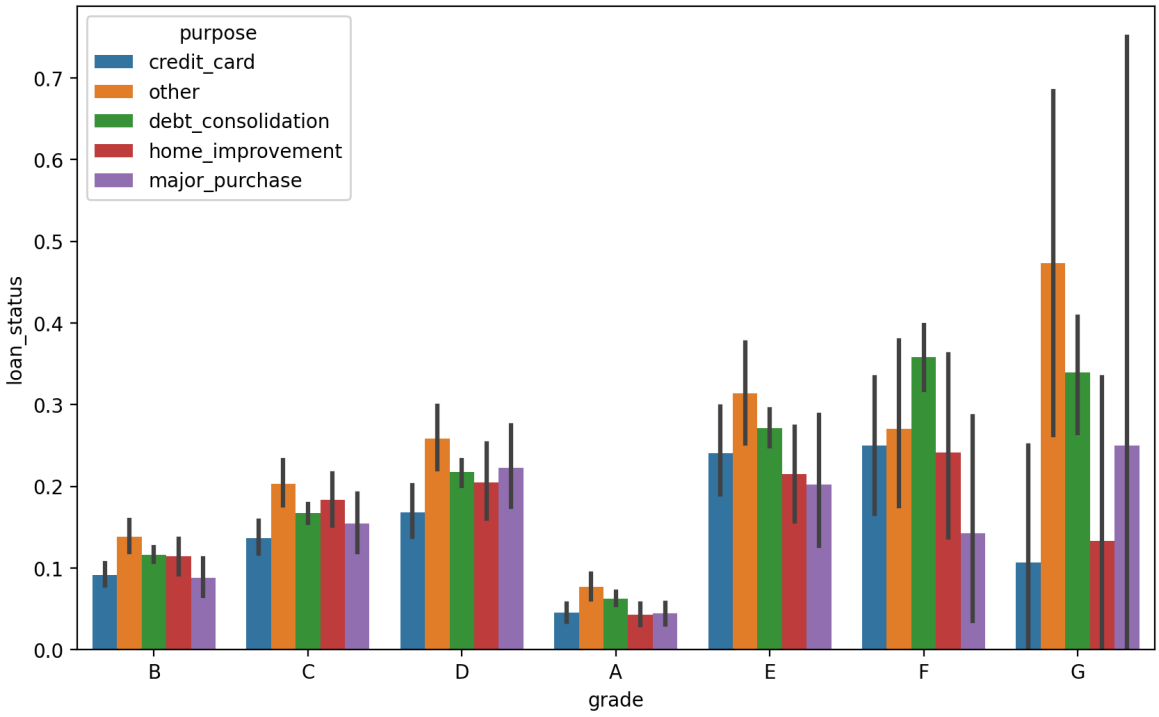
Out[99]:
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'pub_rec_bankruptcies', 'month',
       'year'],
      dtype='object')
```

In [100…
```
# now we compare the default rates across the loan purpose and 'home_ownership'

plt.figure(figsize=[10, 6])
sns.barplot(x='home_ownership', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



Other home holders who took loans for credit card & debt consolidation and home improvement showed increased defaults.

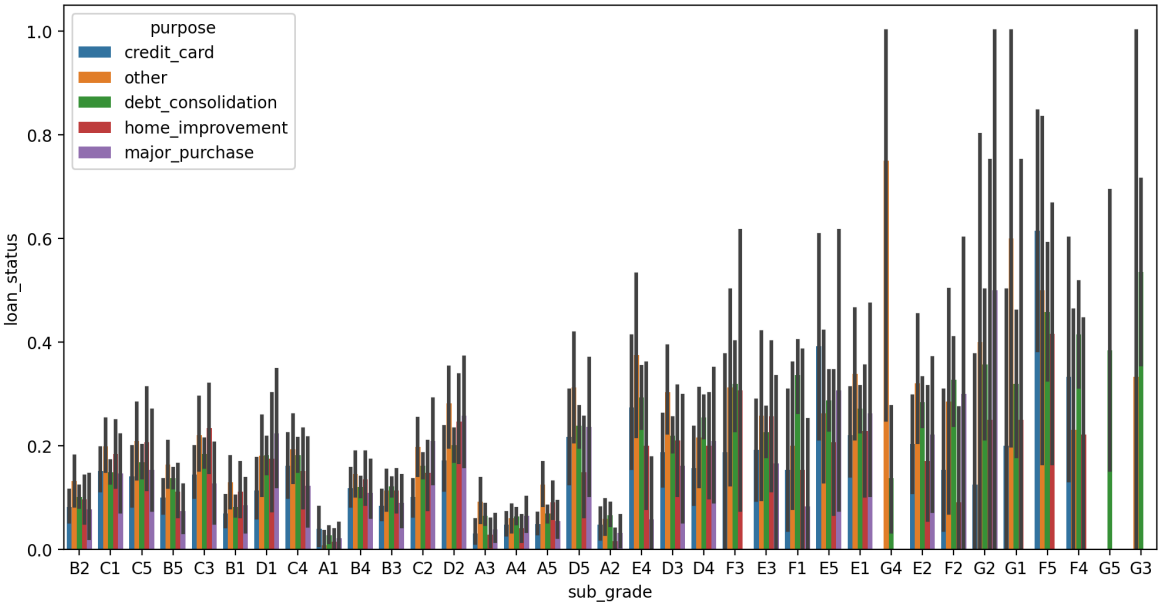
In [101…
```
# now we compare the default rates across the loan purpose and grade wise

plt.figure(figsize=[10, 6])
sns.barplot(x='grade', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



Even among Grade G borrowers, defaults were observed across other purposes.

In [102…

```
# now we compare the default rates across the loan purpose and sub_grade wise

plt.figure(figsize=[12, 6])
sns.barplot(x='sub_grade', y="loan_status", hue='purpose', data=new_data)
plt.show()
```

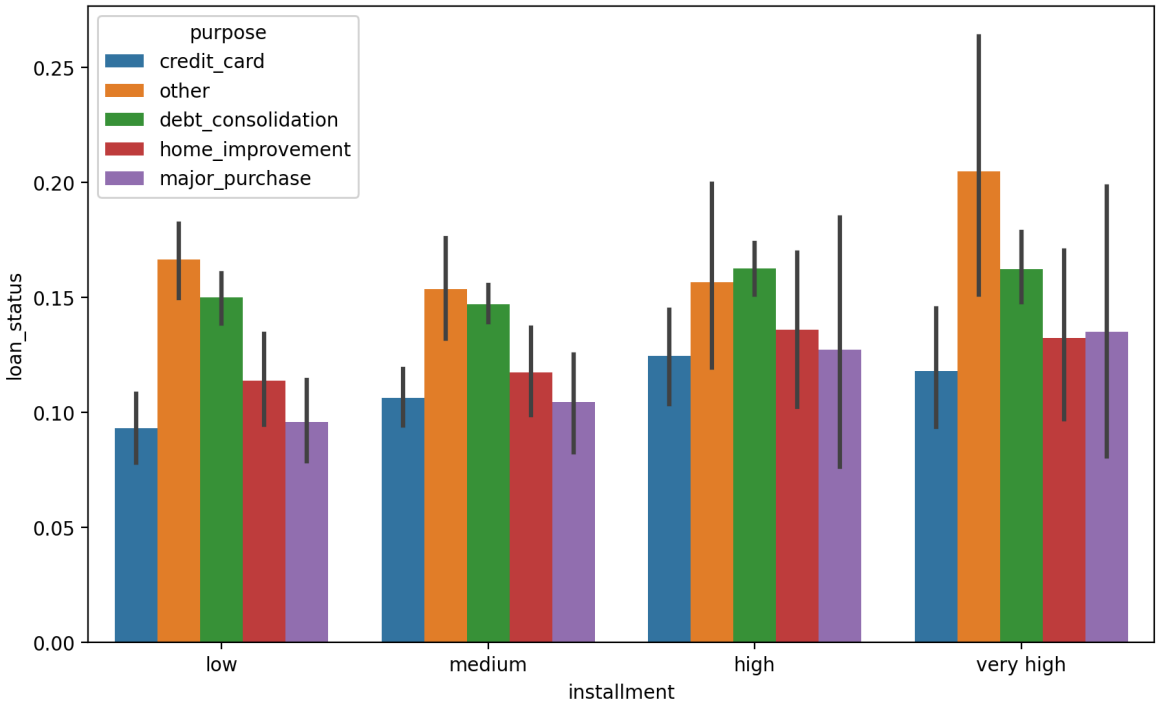


Sub-grades G1 to G4 had a noticeable number of charge-offs, regardless of loan purpose.

In [103…

```
# now we compare the default rates across the loan purpose and installment wise

plt.figure(figsize=[10, 6])
```

```
sns.barplot(x='installment', y="loan_status", hue='purpose', data=new_data)
plt.show()
```



In most of the applicants get Defaults were very high installment ranges when the purpose was other & debt consolidation.

"A good way to quantify th effect of a categorical variable on default rate is to see 'how much does the default rate vary across the categories".

Let's see an example using annual_inc as the categorical variable.

In [104…

```
# let's check the default rates across the annual income

new_data.groupby("annual_inc").loan_status.mean().sort_values(ascending=False) *
```

Out[104…

```
annual_inc
low          16.572789
medium       13.145286
very high    10.848287
high         10.264901
Name: loan_status, dtype: float64
```

Now we analyse the defaulters flow means charged off customers flow by

using filters

Defaulters Means Charge Off Customers/Applicants Data Analysis

In [105…
```
# Filter only defaulters (Charged Off customers)

df_defaulters = new_data[new_data['loan_status'] == 1]
```

In [106…
```
# print the top 5 defaulters

df_defaulters.head()
```

Out[106…

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate
9	1071570	1306721	medium	medium	medium	60 months	medium
12	1064687	1298717	medium	medium	medium	36 months	medium
14	1069057	1303503	medium	medium	medium	36 months	medium
21	1039153	1269083	high	high	high	36 months	medium
24	1069559	1304634	medium	medium	medium	36 months	medium

In [107…
```
# print the last 5 defaulters

df_defaulters.tail()
```

Out[107…

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rat
39638	127830	127824	medium	medium	low	36 months	mediu
39666	119043	119040	high	high	low	36 months	mediu
39667	118823	118026	low	low	low	36 months	mediu
39668	118533	117783	low	low	low	36 months	lo
39688	111227	111223	high	high	low	36 months	mediu

In [108…
```
# Count of defaulters
df_defaulters['loan_status'].value_counts()
```

Out[108…
```
loan_status
1    4511
Name: count, dtype: int64
```

In [109…
```
# Summary statistics
df_defaulters.describe()
```

Out[109…

	id	member_id	issue_d	loan_status	pub_rec_bankruptcie
count	4.511000e+03	4.511000e+03	4511	4511.0	4511.00000
mean	7.044737e+05	8.763418e+05	2010-11-30 09:03:37.867435264	1.0	0.06761
min	6.141900e+04	1.112230e+05	2007-08-01 00:00:00	1.0	0.00000
25%	5.358390e+05	6.922480e+05	2010-06-01 00:00:00	1.0	0.00000
50%	7.068310e+05	8.990130e+05	2011-03-01 00:00:00	1.0	0.00000
75%	8.628910e+05	1.075540e+06	2011-09-01 00:00:00	1.0	0.00000
max	1.071570e+06	1.306721e+06	2011-12-01 00:00:00	1.0	2.00000
std	2.159281e+05	2.705394e+05	NaN	0.0	0.25286

In [110…
```
# Check data types
df_defaulters.dtypes
```

Out[110…
```
id                             int64
member_id                      int64
loan_amnt                     object
funded_amnt                   object
funded_amnt_inv               object
term                          object
int_rate                      object
installment                   object
grade                         object
sub_grade                     object
emp_length                    object
home_ownership                object
annual_inc                    object
verification_status           object
issue_d                 datetime64[ns]
loan_status                    int64
purpose                       object
dti                           object
pub_rec_bankruptcies         float64
month                          int64
year                           int64
dtype: object
```

In [111…
```
# Check missing values
missing_values = df_defaulters.isnull().sum()
```

```
missing_values
```

Out[111…

```
id                      0
member_id               0
loan_amnt               0
funded_amnt             0
funded_amnt_inv         0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
dti                     0
pub_rec_bankruptcies    0
month                   0
year                    0
dtype: int64
```

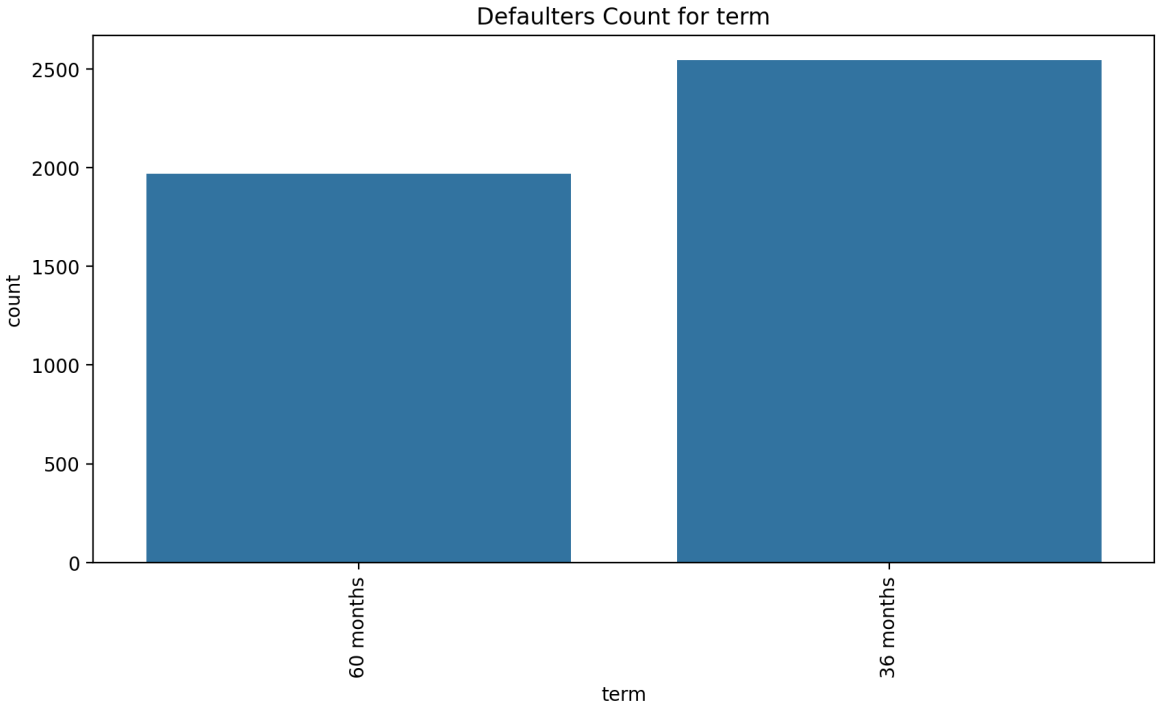
Uni-Variate Analysis

In [112…

```
def plot_defaulters(feature):
    plt.figure(figsize=(10, 5))
    sns.countplot(x=feature, data=df_defaulters)
    plt.xticks(rotation=90)
    plt.title(f"Defaulters Count for {feature}")
    plt.show()
```

In [113…

```
# term wise loan status for defaulters
plot_defaulters("term")
```

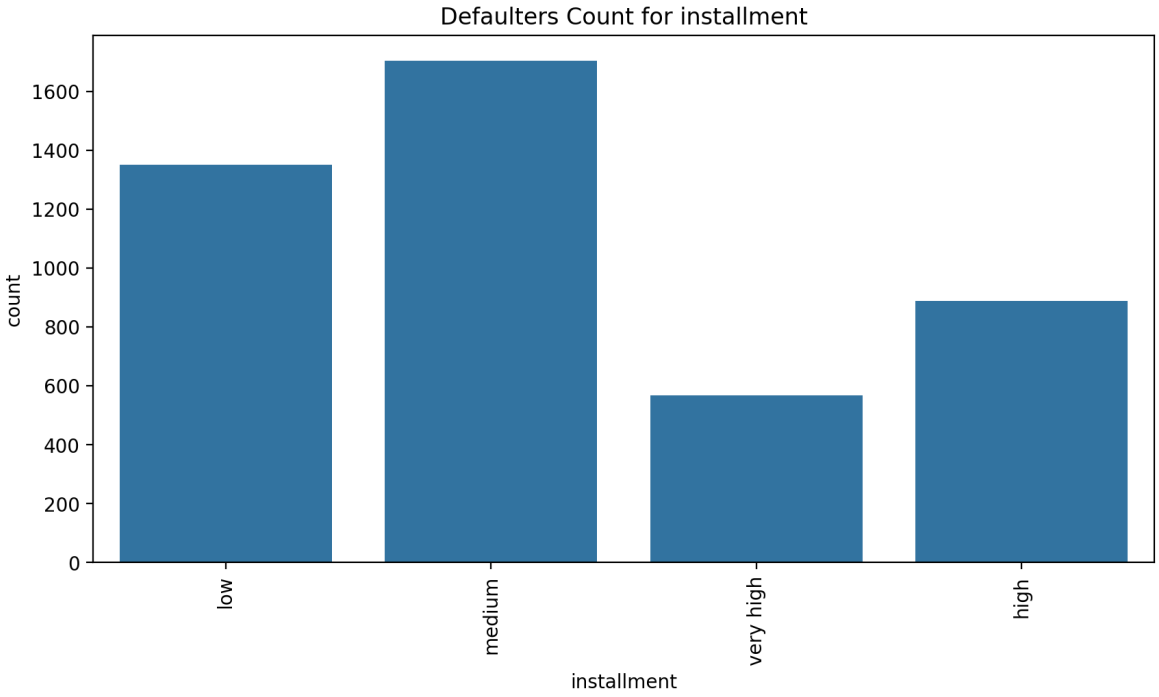


Most of the defaulters are fail to pay their loan on or before time & interesting thing is they have 36 months terms.

In [114…
```
df_defaulters.columns
```

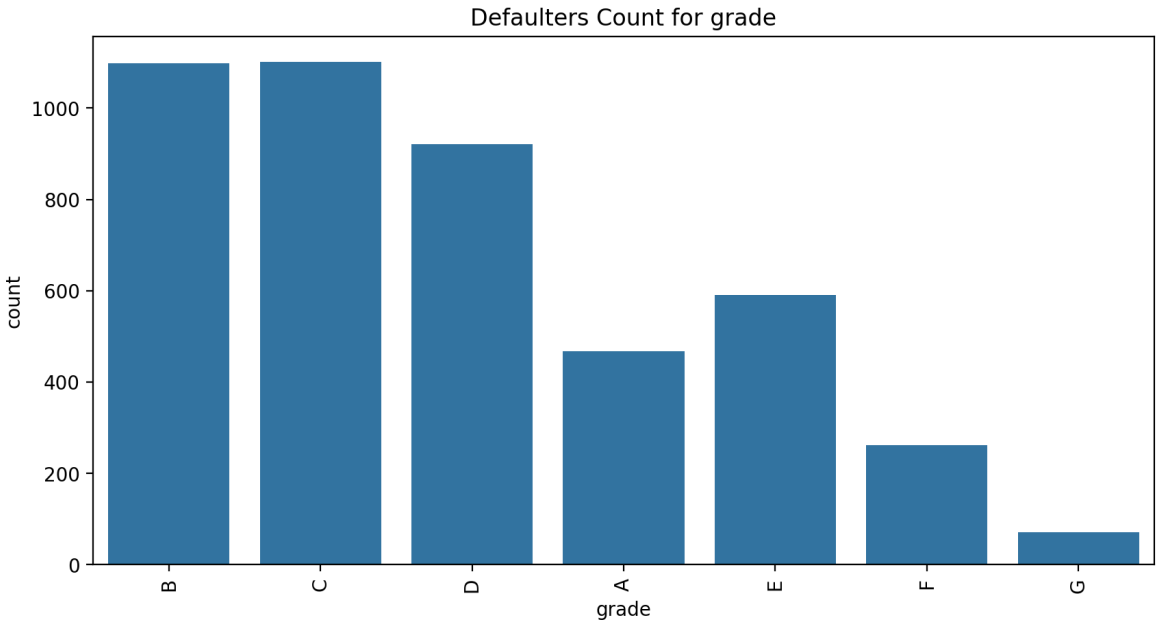
Out[114…
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'pub_rec_bankruptcies', 'month',
       'year'],
      dtype='object')
```

In [115…
```
# installment wise default rate
plot_defaulters("installment")
```



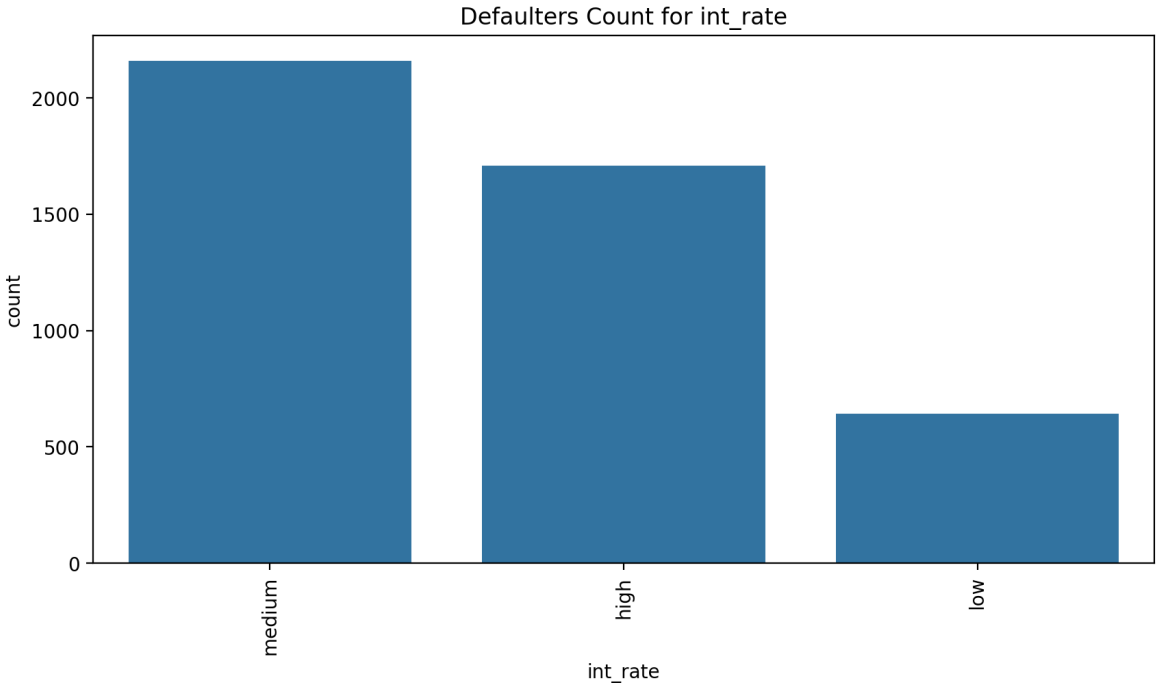
Most of the defaulters have medium installments & interesting thing is low installments is high to get defaulters.

In [116…
```
# grade wise default rate
plot_defaulters("grade")
```



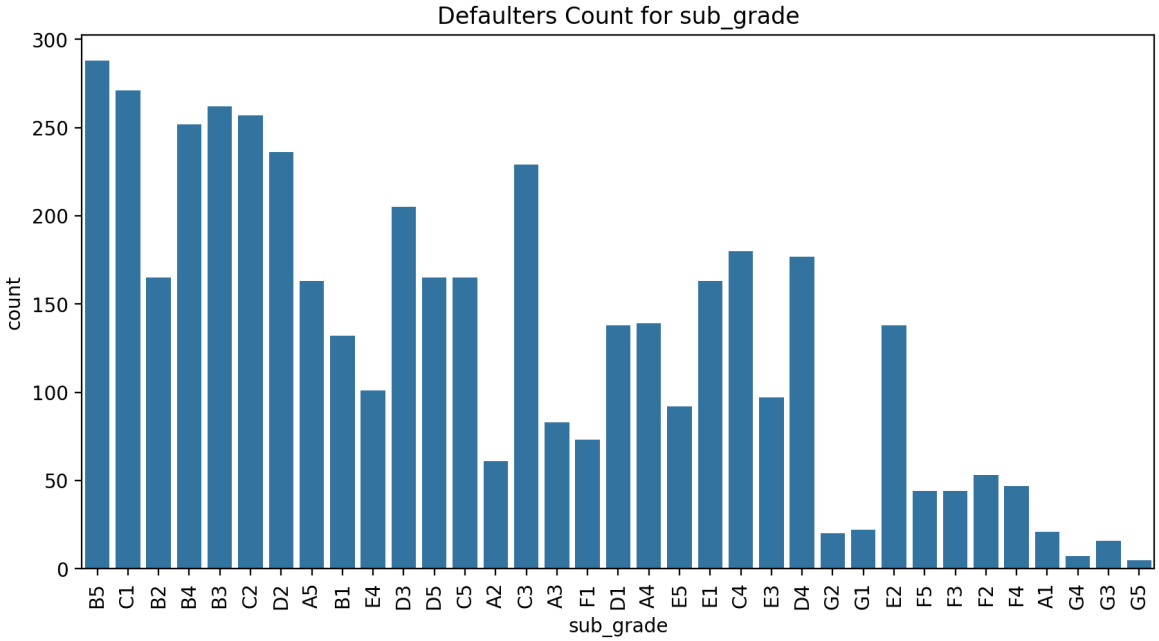
In above chart as you can see the defaulters Grade B,C,D grade.

In [117…
```
# int_rate wise default rate
plot_defaulters("int_rate")
```



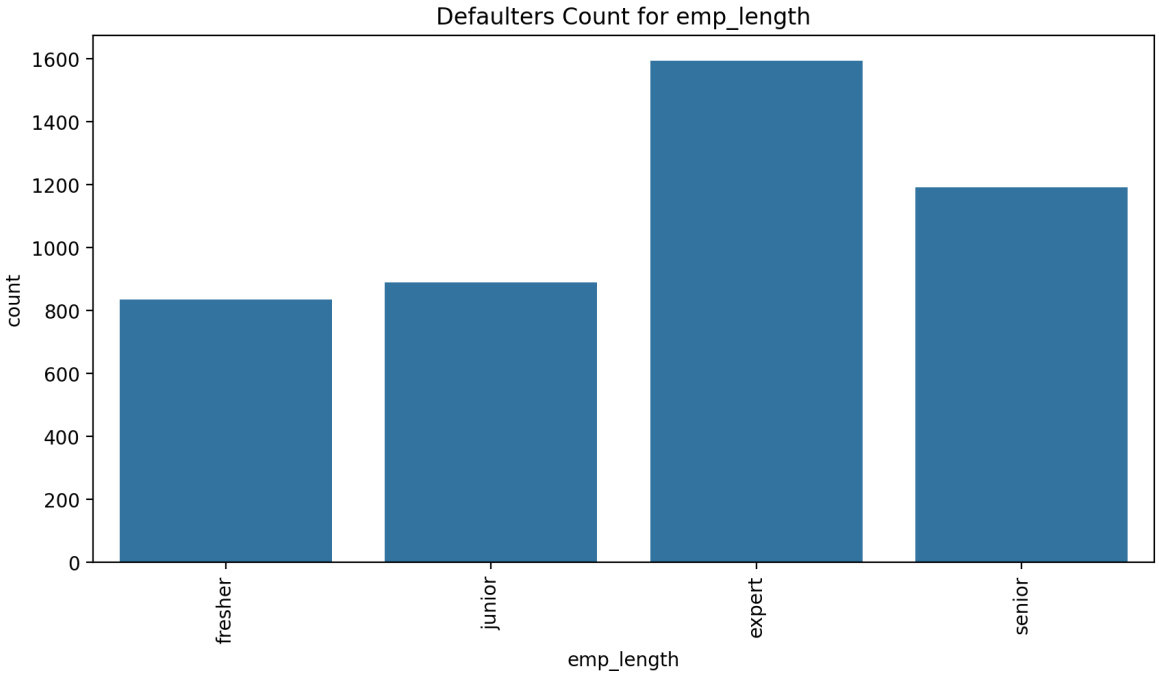
In the above chart you can see the medium category interest get the highest defaulters.

In [118…
```
# sub_grade wise default rate
plot_defaulters("sub_grade")
```



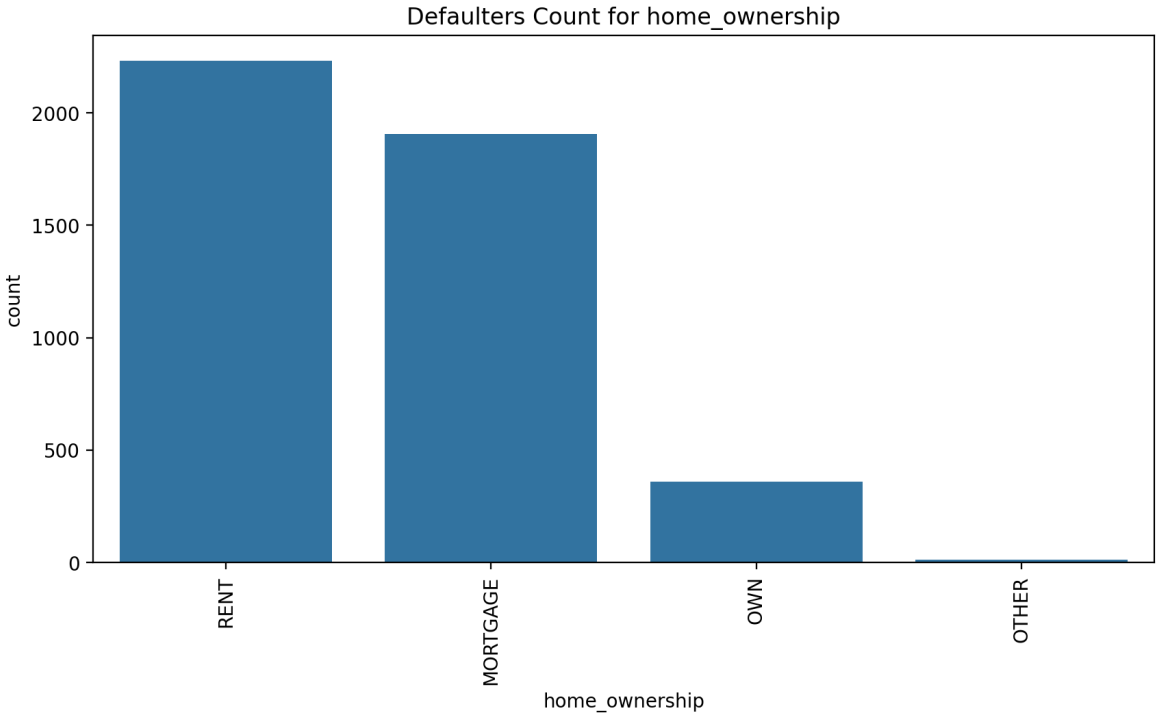
In the above chart you can see the B5 & C1 sub grade category interest get the highest defaulters.

In [119…
```
# emp_length wise default rate
plot_defaulters("emp_length")
```



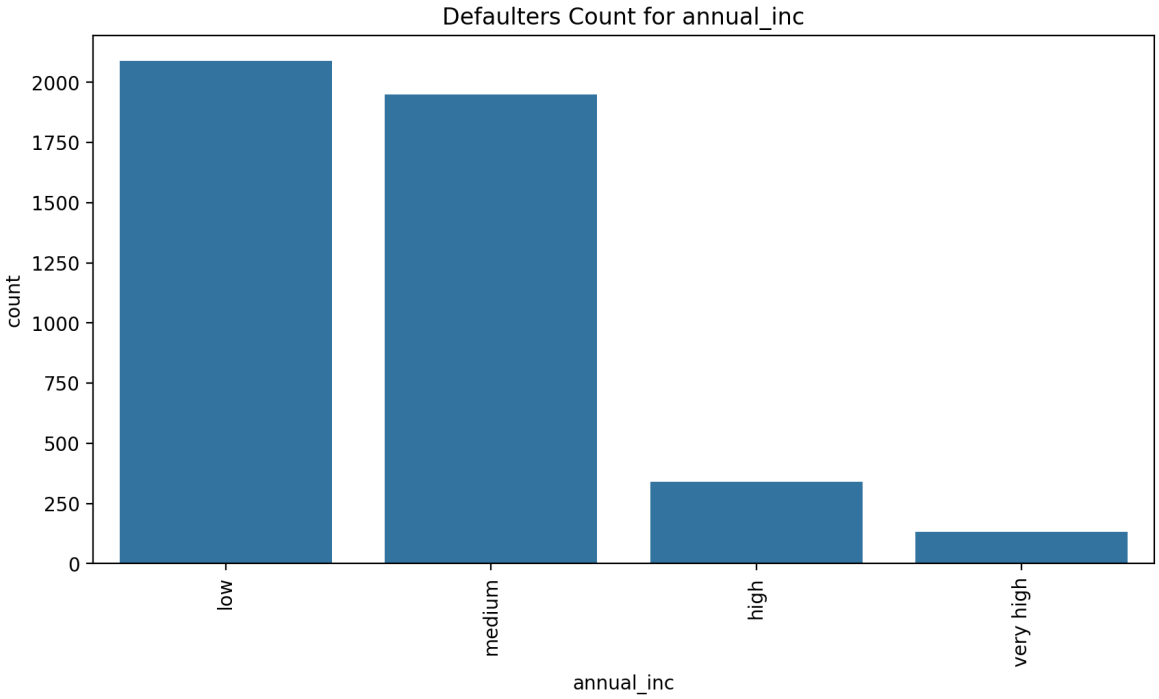
According to employee length in experts are high defaulters.

In [120…
```
# home_ownership wise default rate
plot_defaulters("home_ownership")
```



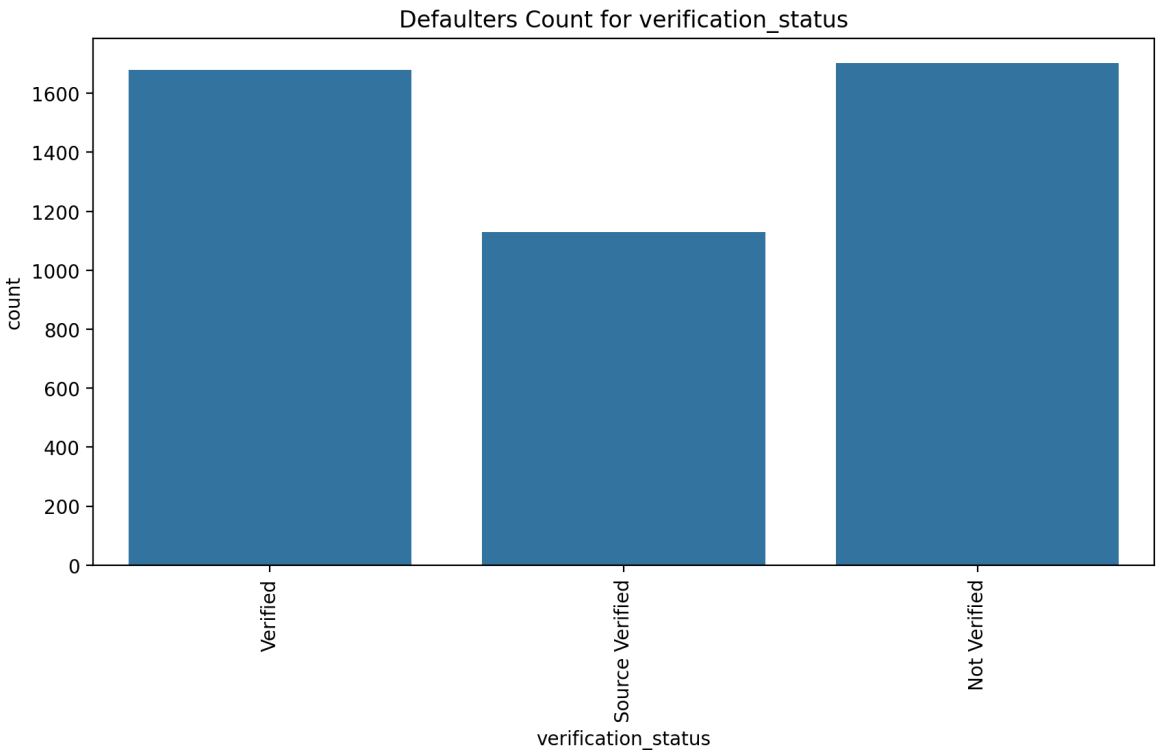
In the above chart you can see the RENT category home ownership get the highest defaulters.

In [121…
```
# annual_inc wise default rate
plot_defaulters("annual_inc")
```



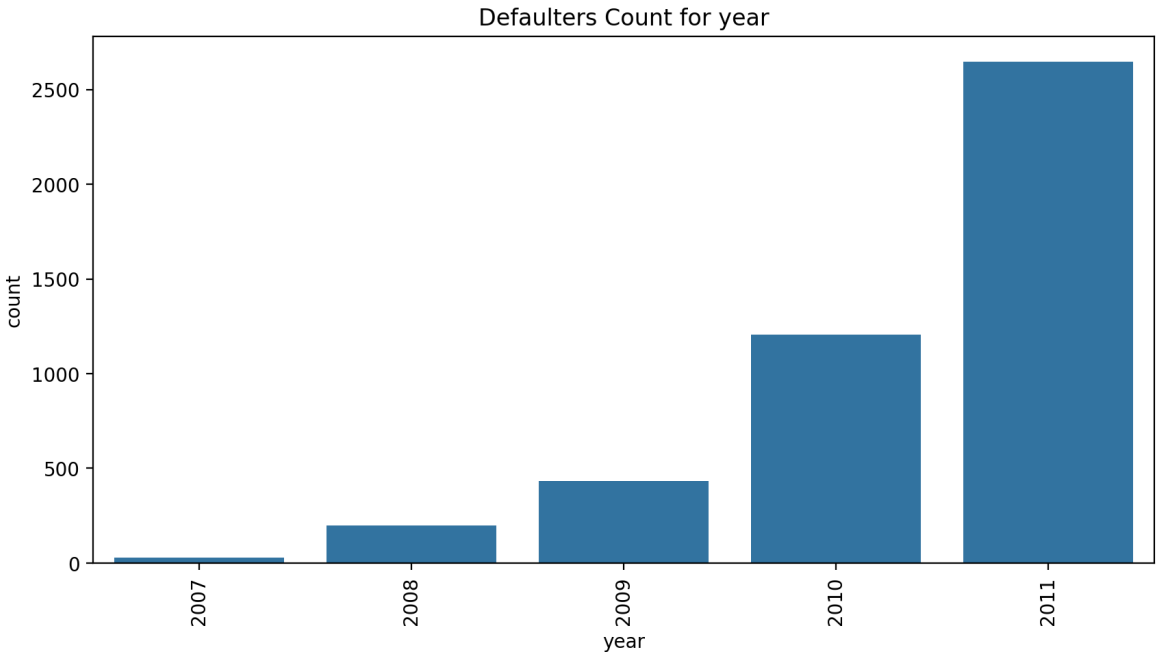
In the above chart you can see the low and medium category annual income get the highest defaulters.

In [122…
```
# verification_status wise default rate
plot_defaulters("verification_status")
```



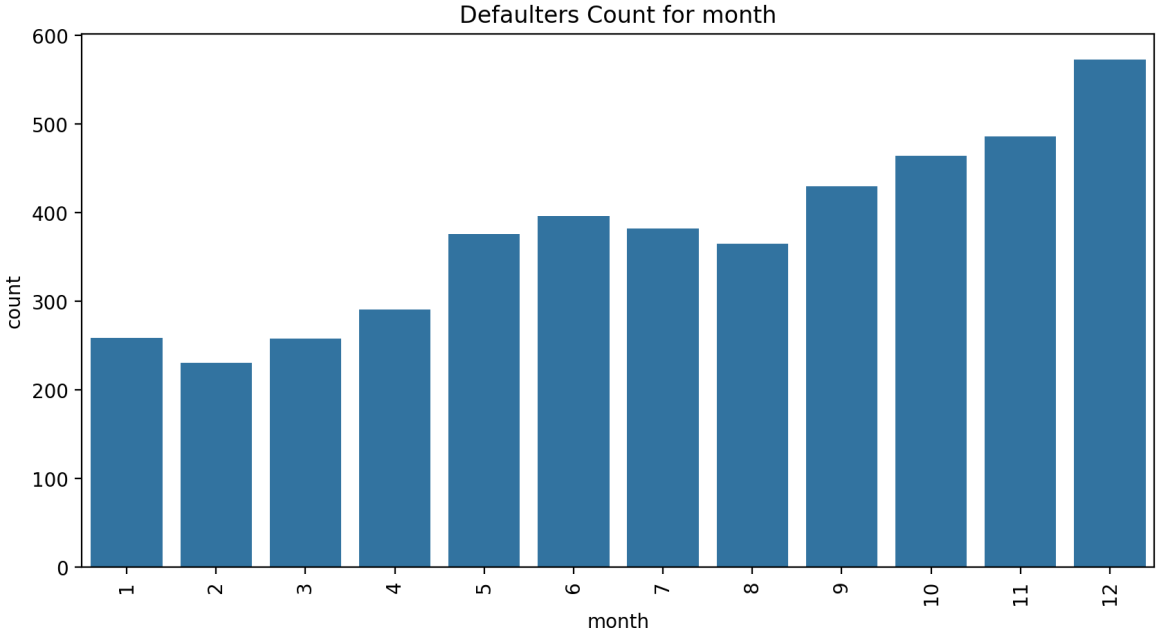
In the above chart you can see the verified verification status & not verified verification statu category get the highest defaulters.

In [123…
```
# year wise default rate
plot_defaulters("year")
```



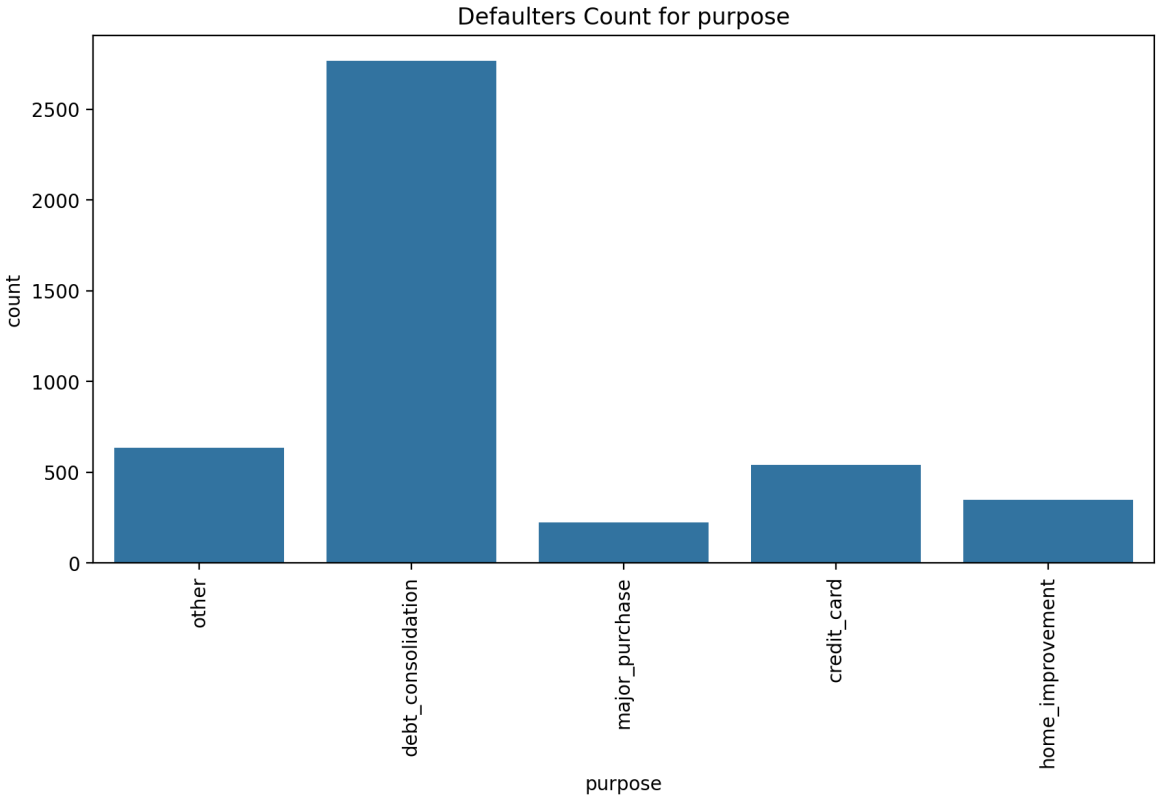
In the above chart you can see the year 2011 get the highest defaulters.

In [124…
```
# month wise default rate
plot_defaulters("month")
```



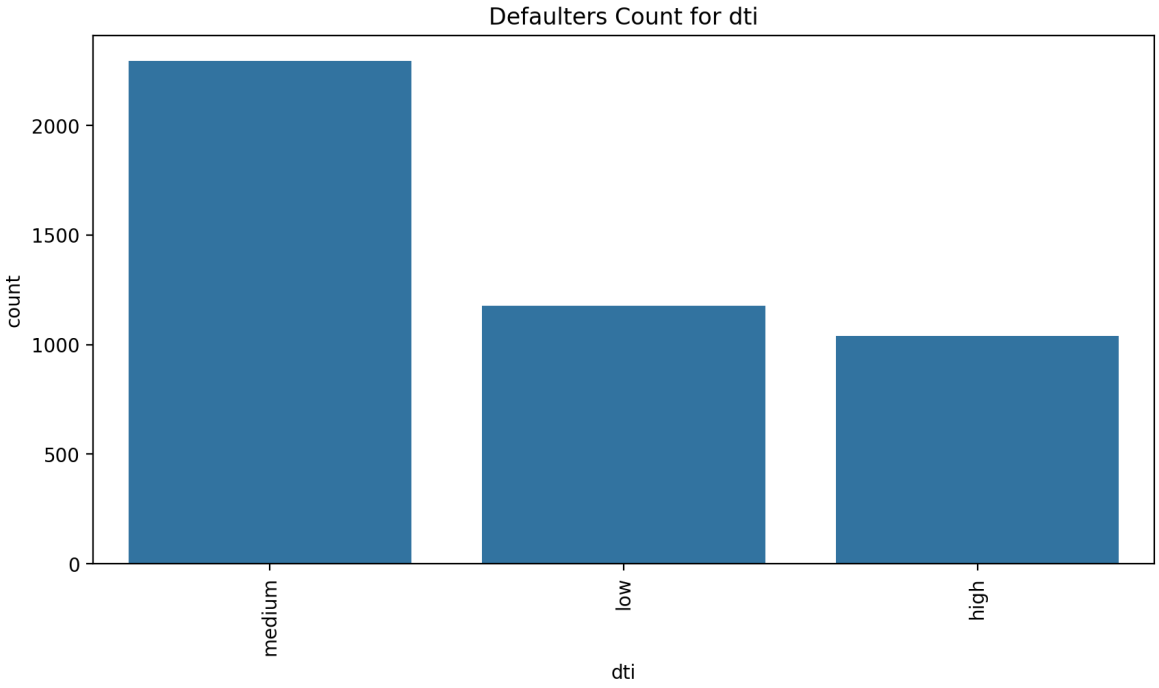
In the above chart you can see the December month get the highest defaulters.

In [125…
```
# purpose wise default rate
plot_defaulters("purpose")
```



In the above chart you can see the debt consolidation loan purpose get the highest defaulters.

In [126…
```
# dti wise default rate
plot_defaulters("dti")
```

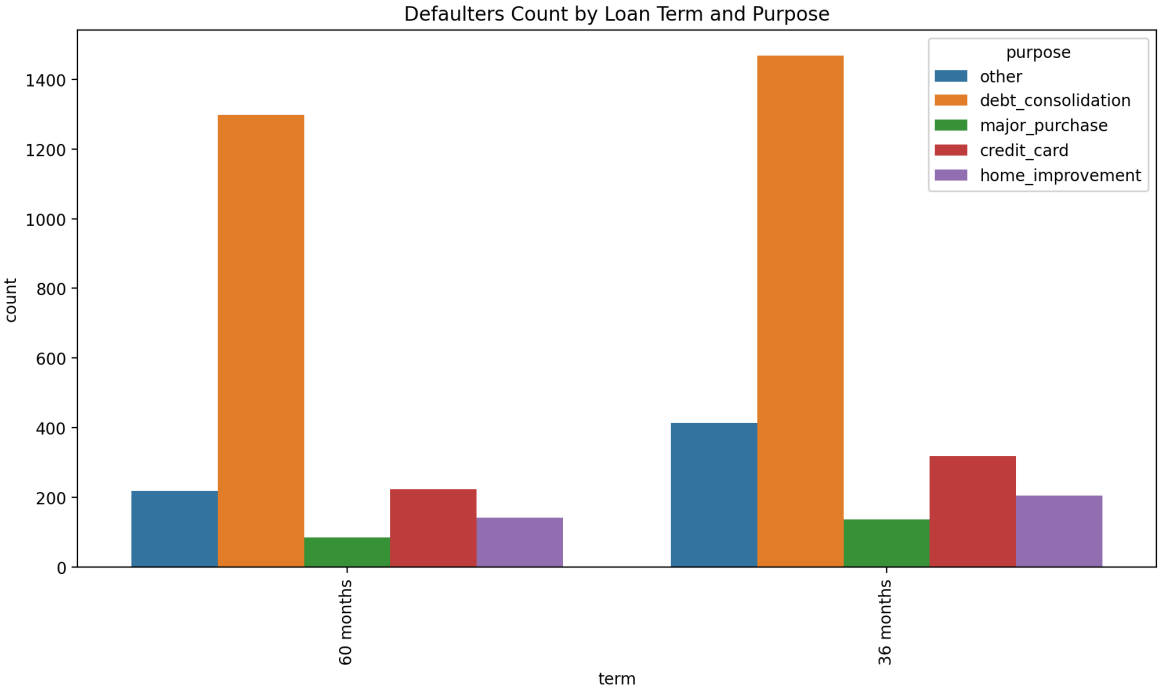


In the above chart you can see the medium dti get the highest defaulters.

Bi-Variate Analysis

In [127…

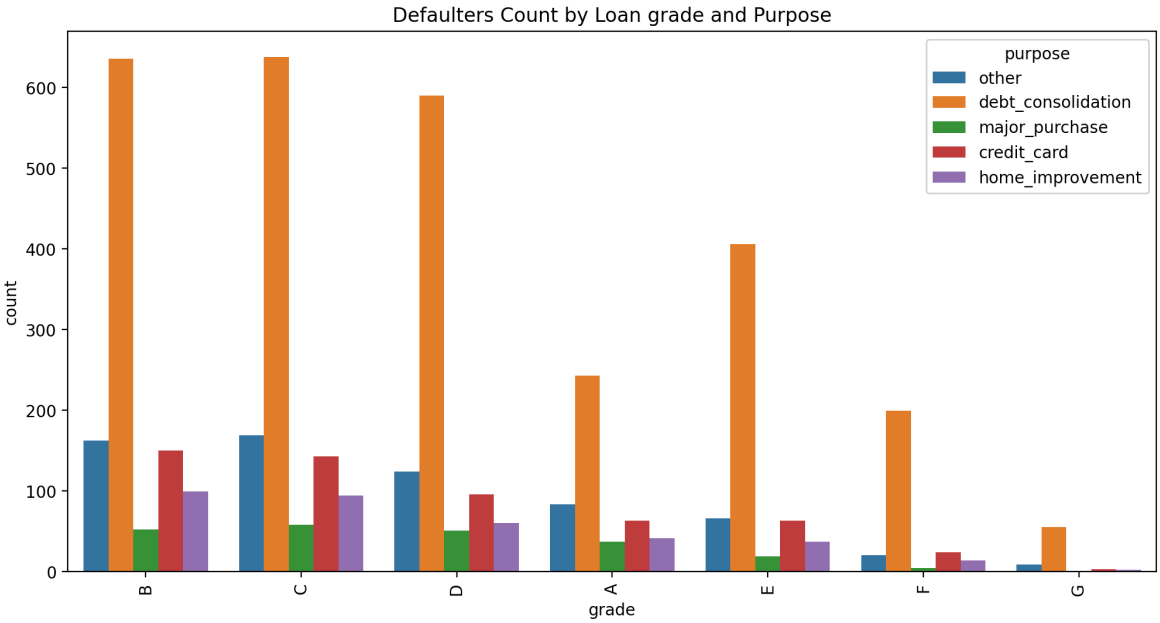
```
# term vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='term', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by Loan Term and Purpose")
plt.show()
```



In the above chart you can see the 36 Months term category get the highest defaulters for the debt consolidation purpose.

In [128…

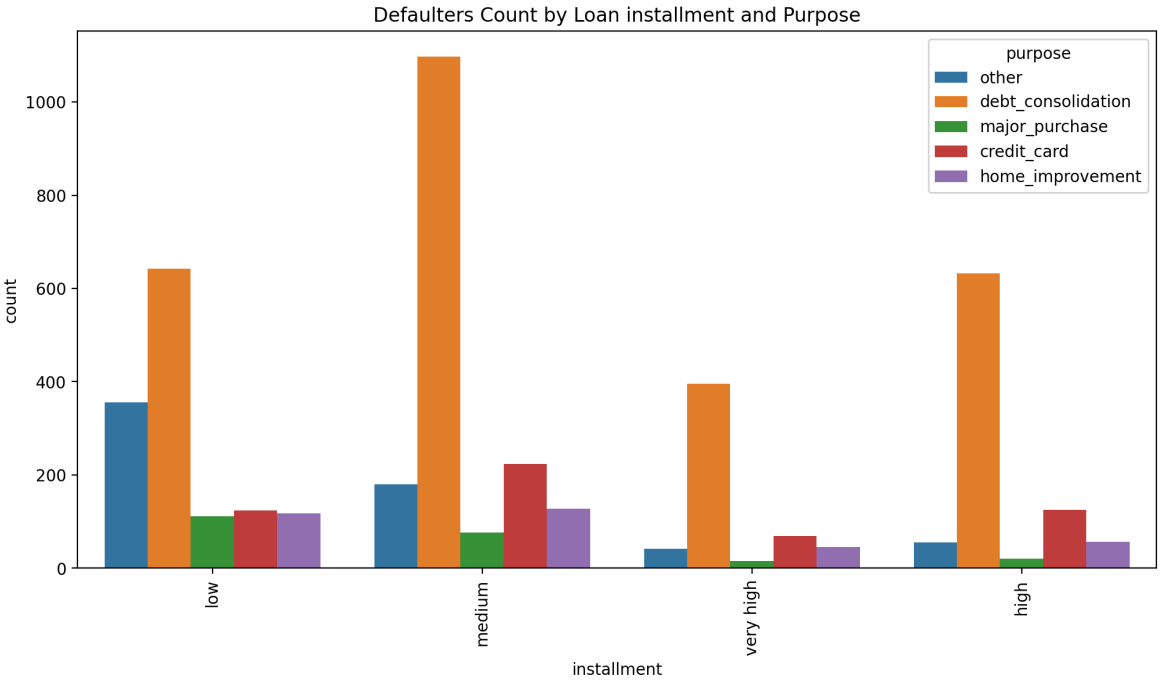
```
# grade vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='grade', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by Loan grade and Purpose")
plt.show()
```



In the above chart you can see the B,C & D grade category applicants get the highest defaulters for the reason of debt consolidation.

In [129…

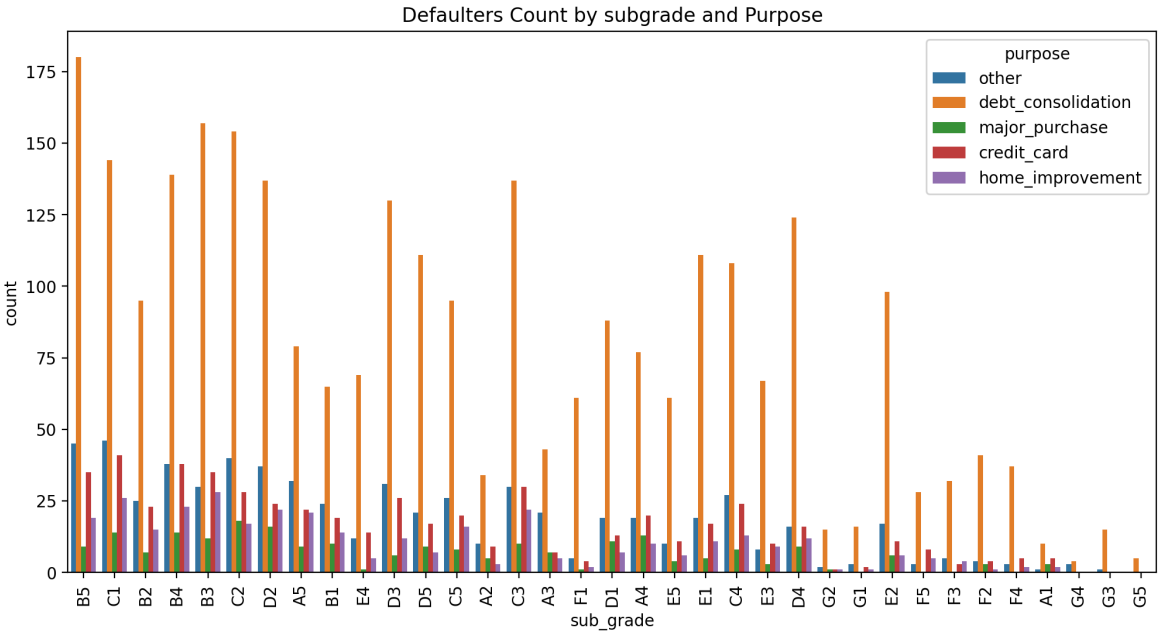
```
# installment vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='installment', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by Loan installment and Purpose")
plt.show()
```



In the above chart you can see the medium category applicants get the highest defaulters for the reason of debt consolidation.

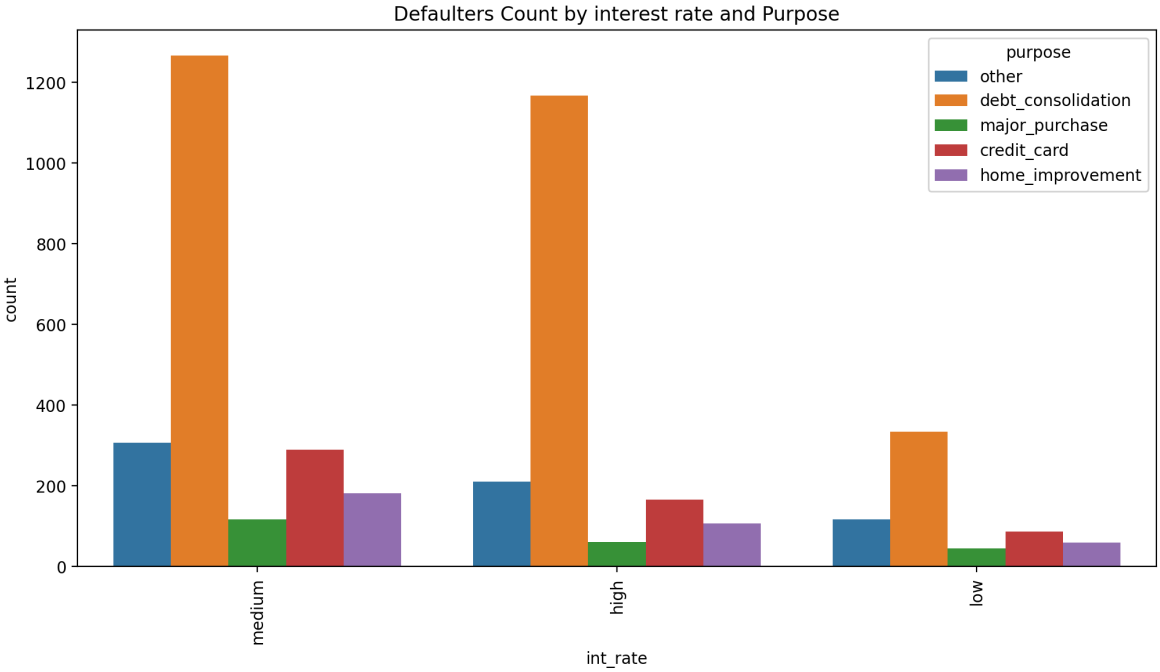
In [130…

```
# sub-grade vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='sub_grade', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by subgrade and Purpose")
plt.show()
```



In the above chart you can see the B5 & B3 sub grade category interest get the highest defaulters for debt consolidation purpose.

In [131…
```
# interes rate vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='int_rate', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by interest rate and Purpose")
plt.show()
```

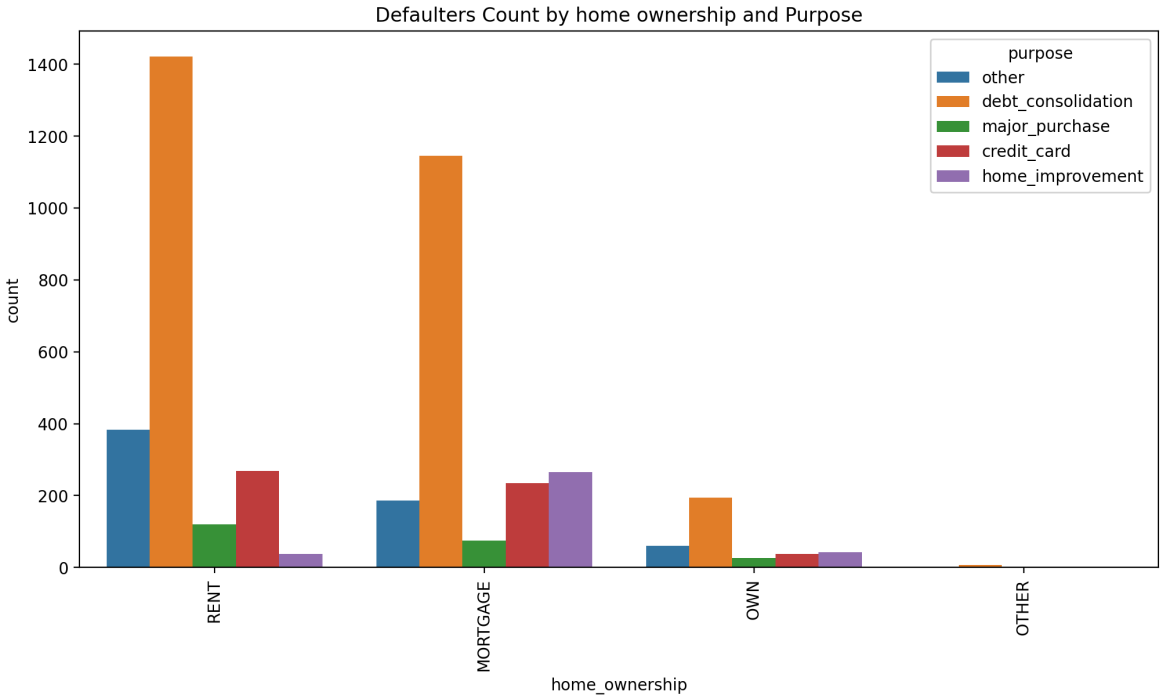


Most of hte defaulters have medium & high interest rates and take the loan for debt consolidation.

In [132…
```
df_defaulters.columns
```

Out[132…
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'dti', 'pub_rec_bankruptcies', 'month',
       'year'],
      dtype='object')
```

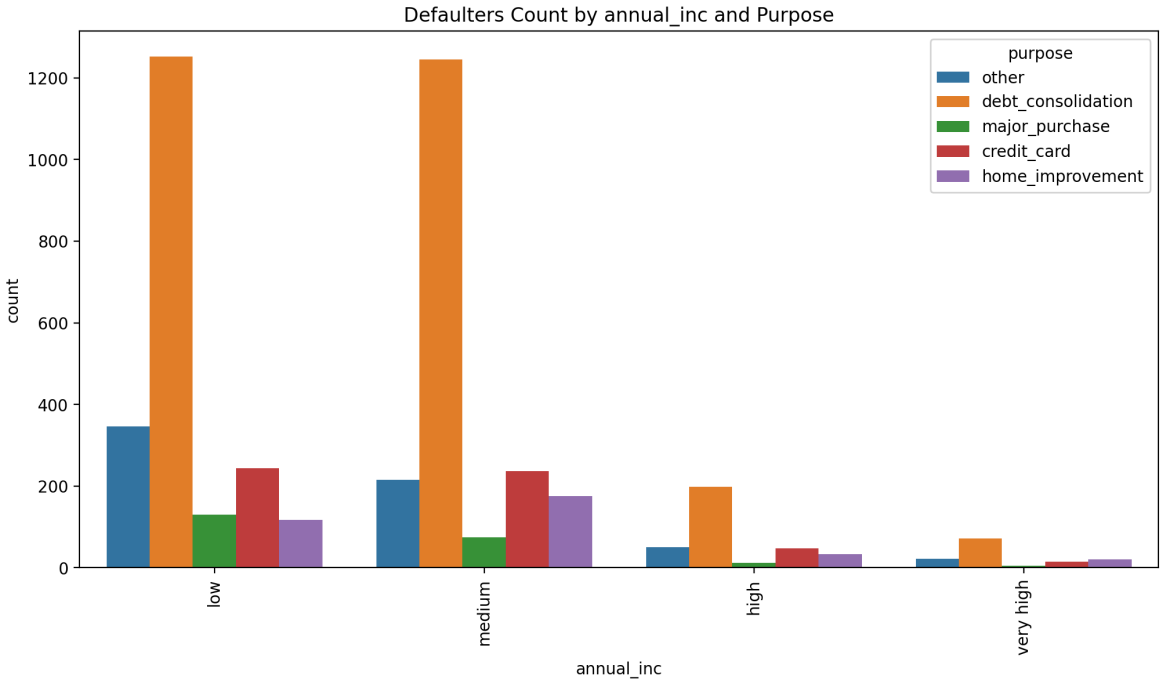
In [133…
```
# home ownership vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='home_ownership', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by home ownership and Purpose")
plt.show()
```



Most of the defaulters are take the loan for the debt consolidation and thier home ownership is rent & mortgage.

In [134…

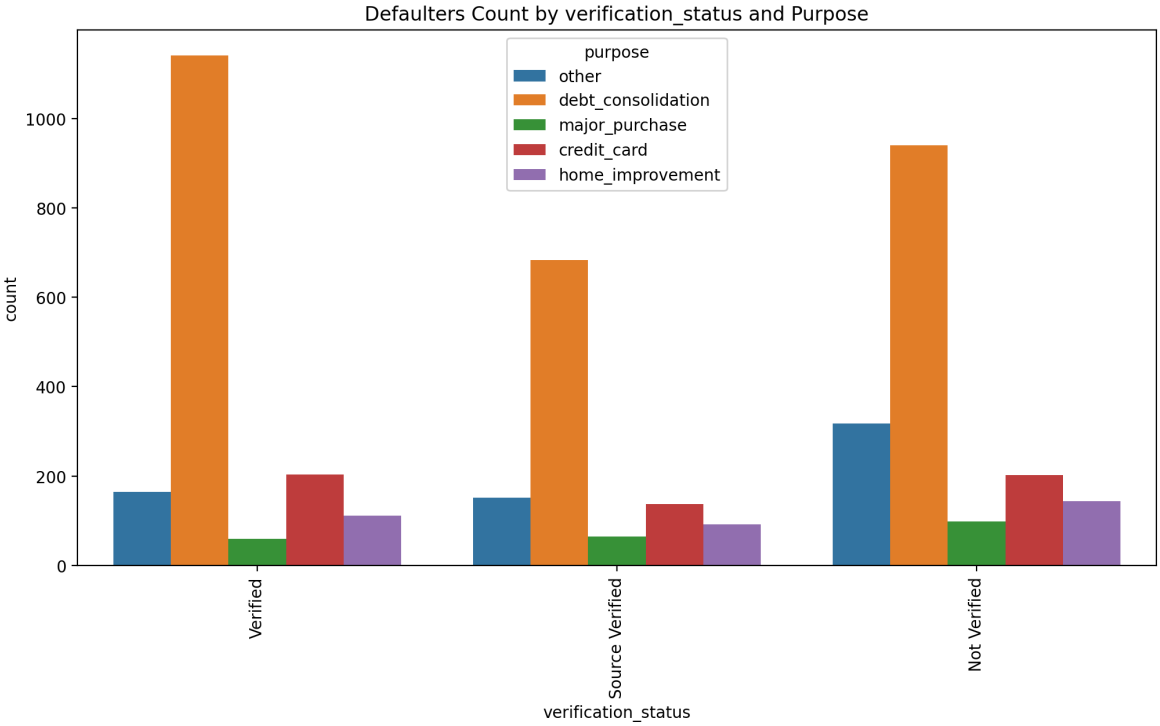
```
# annual_inc vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='annual_inc', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by annual_inc and Purpose")
plt.show()
```



Most of the defautlers have annual income is low and medium category for debt consolidation

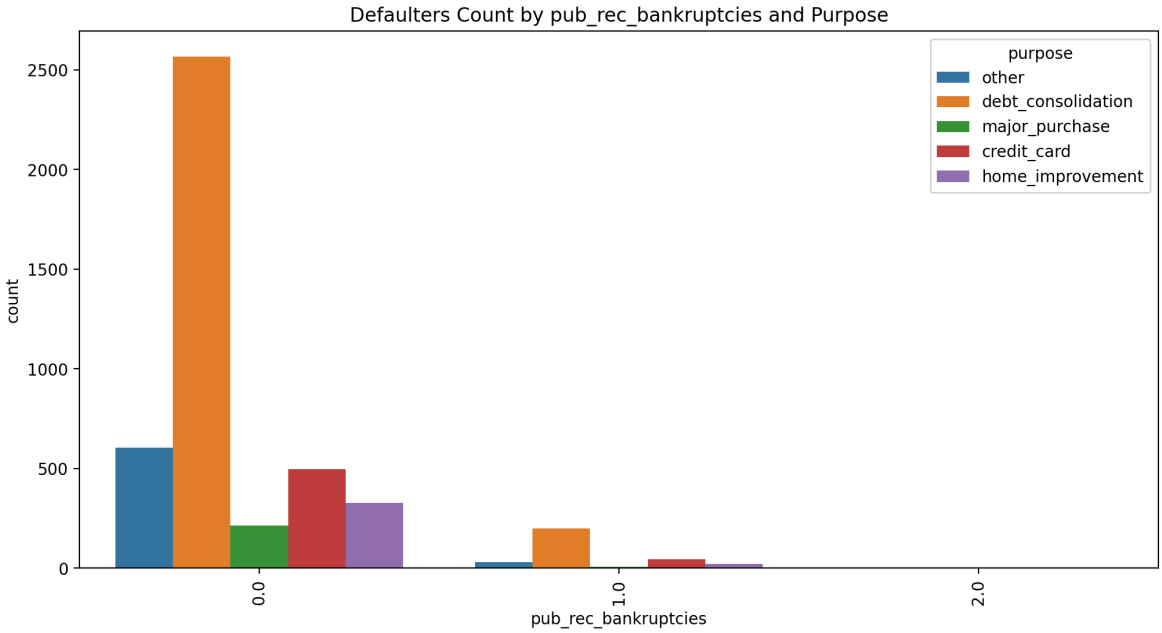
purposes.

In [135…
```
# verification_status vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='verification_status', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by verification_status and Purpose")
plt.show()
```



Most of the defaulters verification status is verified and not verified for debt consolidation purposes.

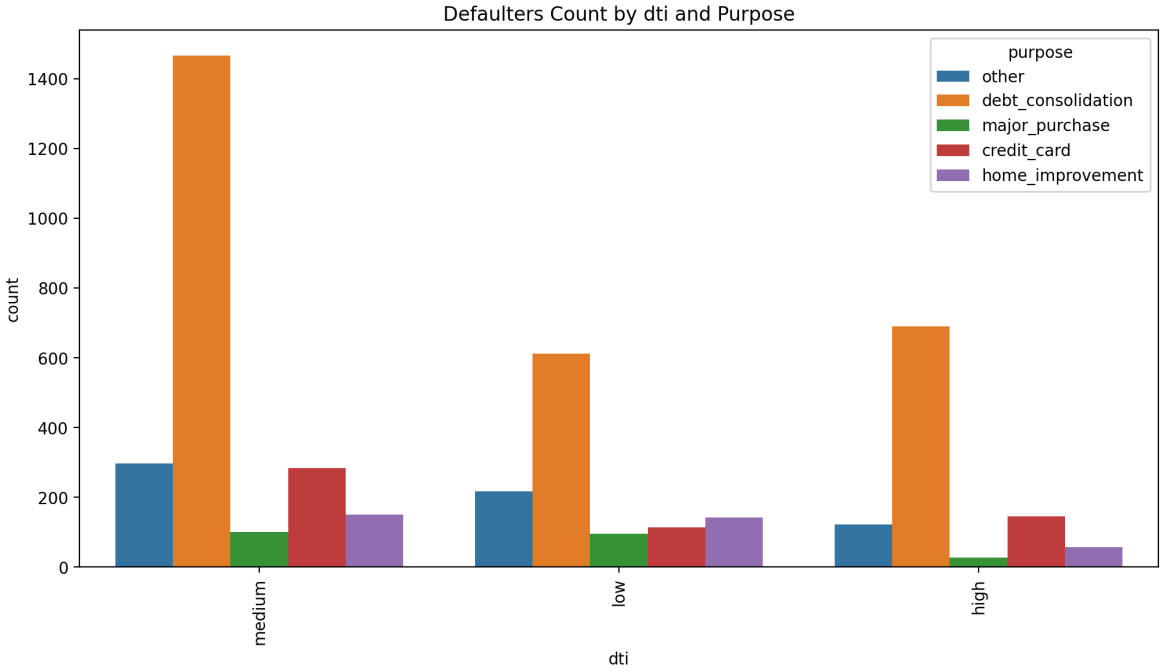
In [136…
```
# pub_rec_bankruptcies vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='pub_rec_bankruptcies', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by pub_rec_bankruptcies and Purpose")
plt.show()
```



In the above chart you can see the 0 public record bankruptcies get the highest defaulters for debt consolidation purposes.

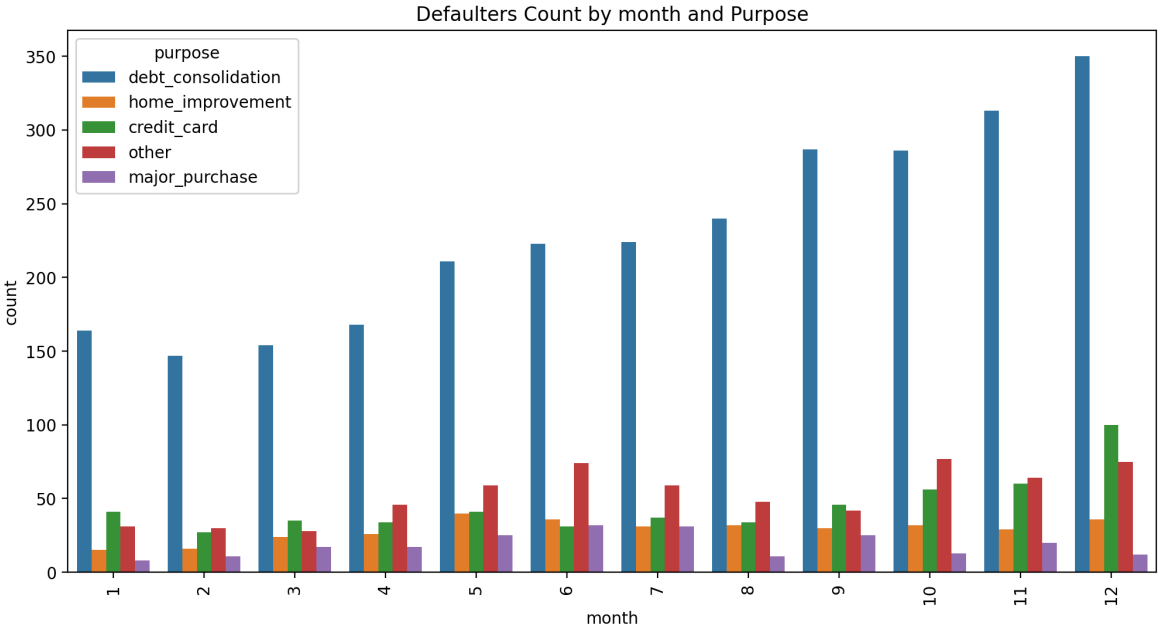
In [137…

```
# dti vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='dti', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by dti and Purpose")
plt.show()
```



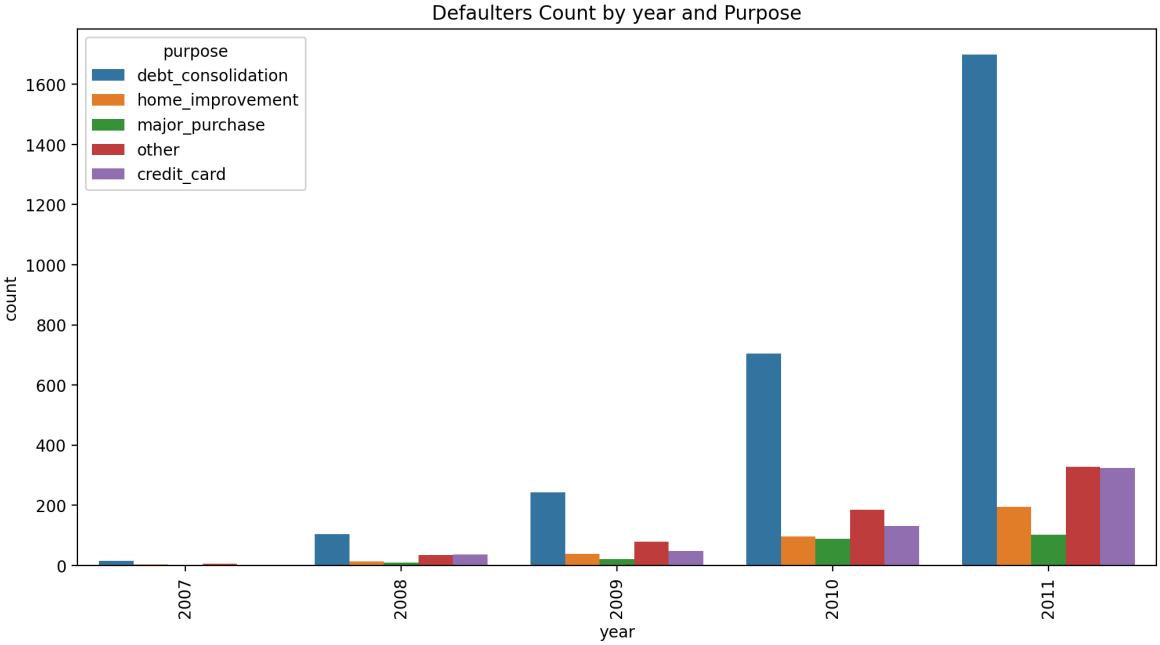
Most of the defaulters have low and medium dti purpose.

In [138…
```
# month vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='month', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by month and Purpose")
plt.show()
```



most of the defaulters are take the loan in the month of 9,10,11,12 for the purpose of debt consolidation.

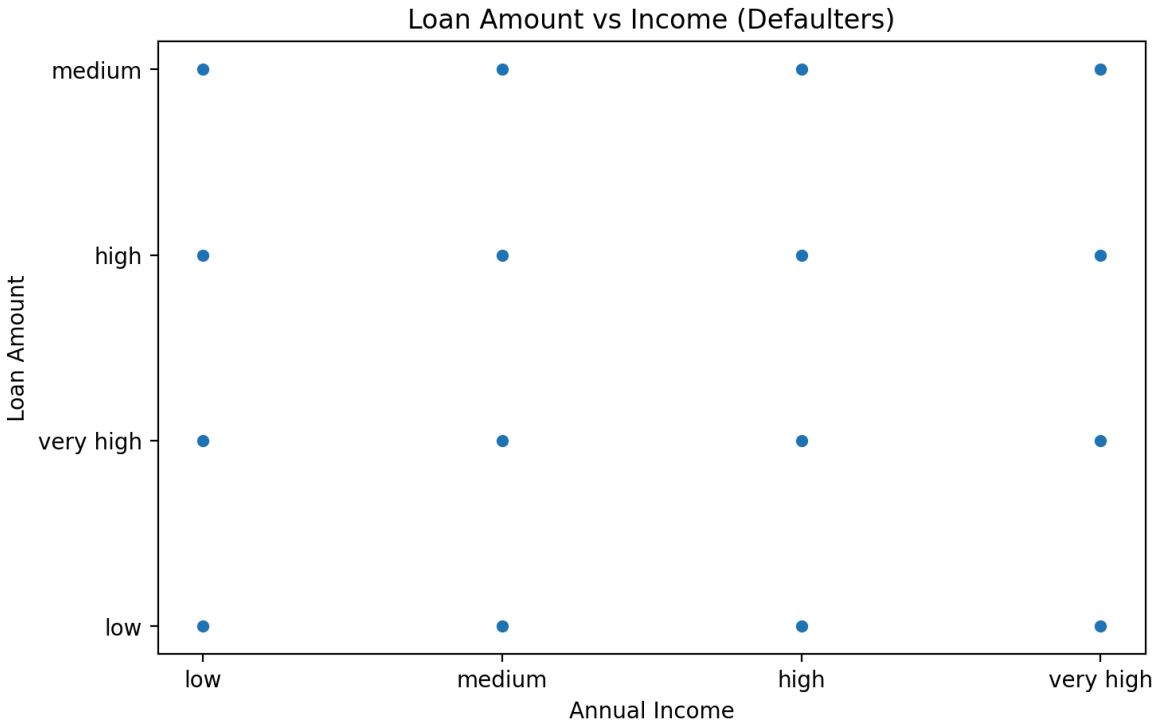
In [139…
```
# year vs purpose default rate
plt.figure(figsize=(12, 6))
sns.countplot(x='year', hue='purpose', data=df_defaulters)
plt.xticks(rotation=90)
plt.title("Defaulters Count by year and Purpose")
plt.show()
```



most of the defaulters take the loan from the year of 2011 for the debt consolidation purpose.

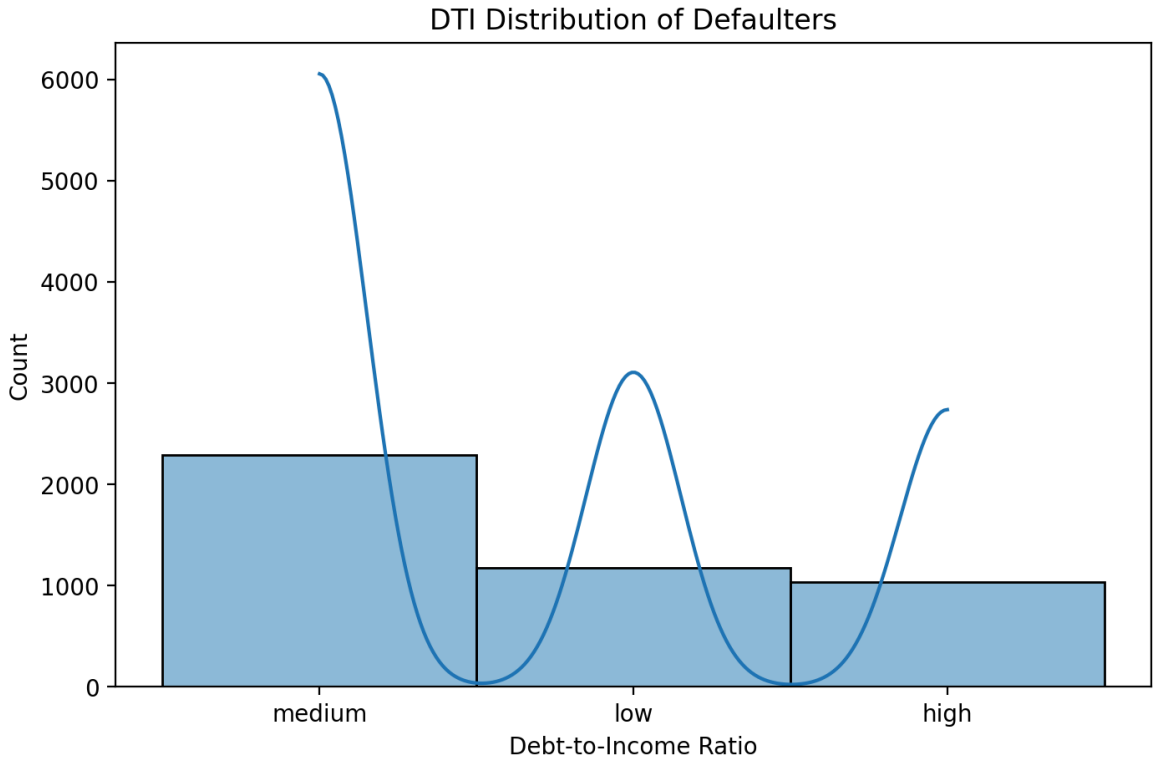
In [140…

```
# Loan Amount vs Income (Defaulters)
plt.figure(figsize=(8,5))
sns.scatterplot(x=df_defaulters['annual_inc'], y=df_defaulters['loan_amnt'])
plt.title('Loan Amount vs Income (Defaulters)')
plt.xlabel('Annual Income')
plt.ylabel('Loan Amount')
plt.show()
```



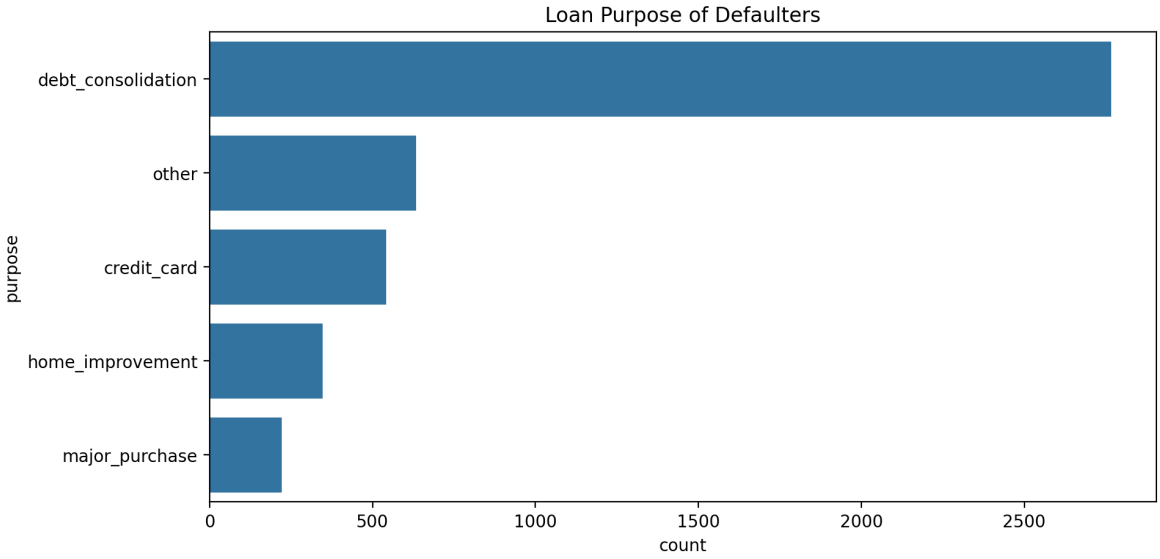
In [141…

```
# DTI (Debt-to-Income) Ratio of Defaulters

plt.figure(figsize=(8,5))
sns.histplot(df_defaulters['dti'], bins=20, kde=True)
plt.title('DTI Distribution of Defaulters')
plt.xlabel('Debt-to-Income Ratio')
plt.show()
```



In [142…

```
# Loan Purpose & Default Rate

plt.figure(figsize=(10,5))
sns.countplot(y=df_defaulters['purpose'], order=df_defaulters['purpose'].value_c
plt.title('Loan Purpose of Defaulters')
plt.show()
```

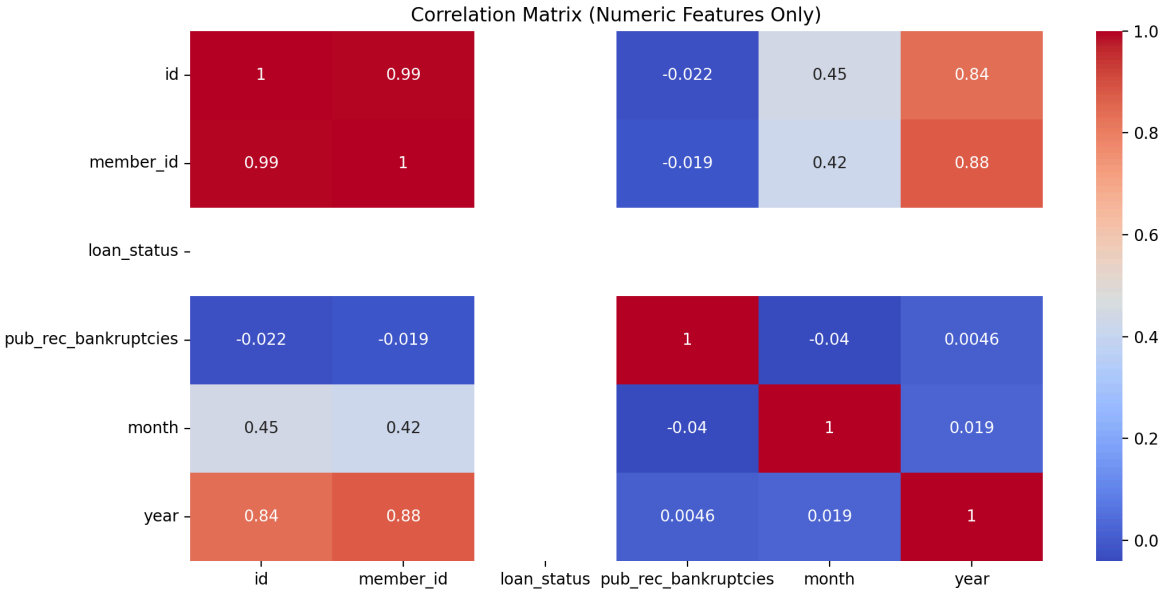


Most of the defaulters take the loan for debt consolidation purpose.

Multi-Variate Analysis (More than Two Variables)

In [143…

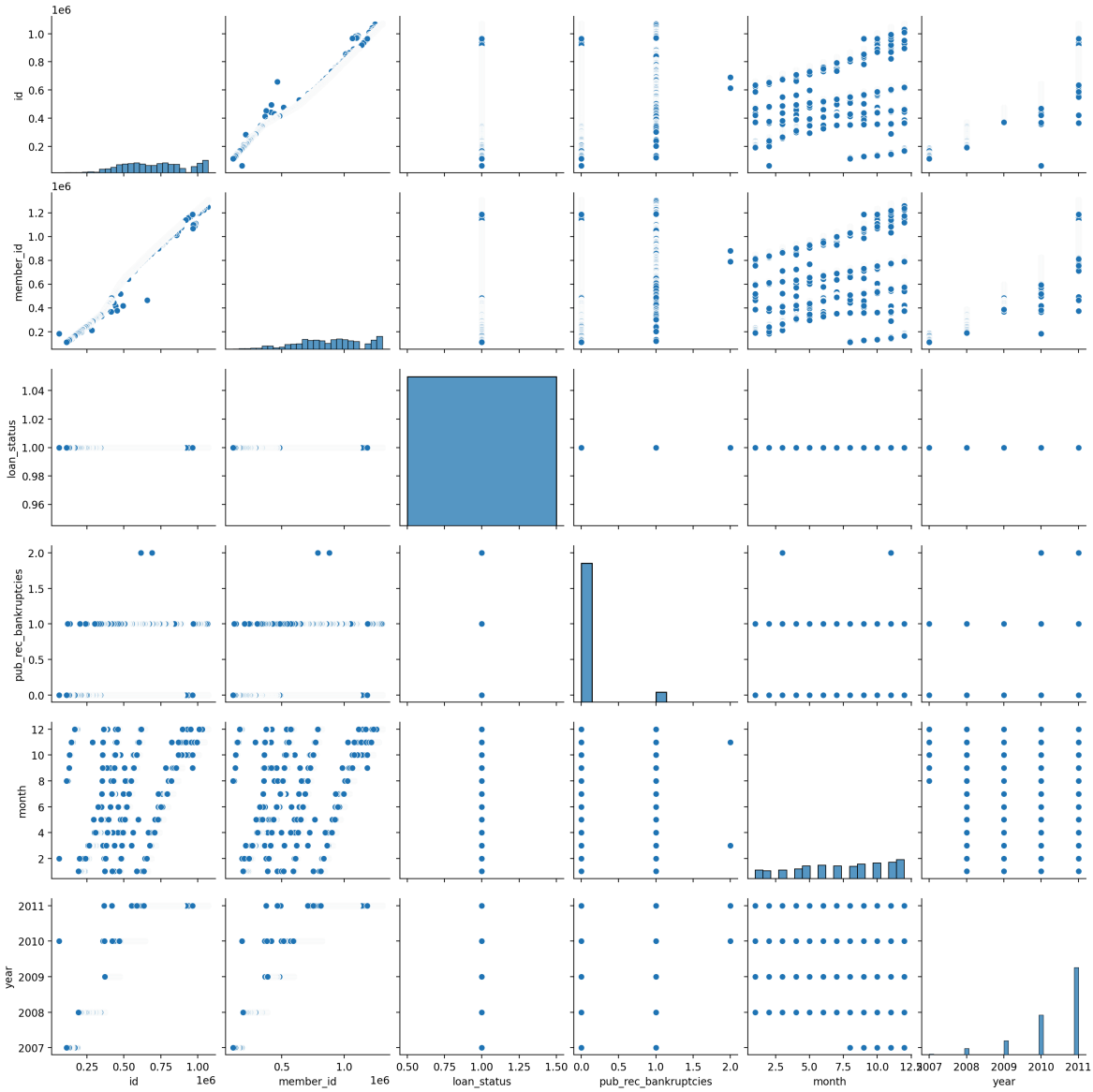
```
# Selecting only numeric columns
numeric_df = df_defaulters.select_dtypes(include=['number'])

# Plotting the heatmap
plt.figure(figsize=(12,6))
sns.heatmap(numeric_df.corr(), annot=True,cmap='coolwarm')
plt.title('Correlation Matrix (Numeric Features Only)')
plt.show()
```



As you can see in the above heatmap no strong relationship between the two another variables.

In [144…

```
# Pairplot for Important Features

# Selecting only numeric columns
numeric_df = df_defaulters.select_dtypes(include=['number'])

# Checking if there are numeric columns
if not numeric_df.empty:
    sns.pairplot(numeric_df)
else:
    print("No numeric columns found in the dataset.")
```



📌 Recommendations Based on Analysis

1. **Risk-Based Interest Adjustment**
   Loans issued to customers with Grade B,C & D (especially sub-grades like B3, B5, C1, etc.) should be charged higher & medium interest rates or subjected to stricter terms, as these categories show a significantly higher default rate.

2. **Stricter Loan Approval for Employment History**
   Customers with employment length of 1, 7, or 10 years means experts show a higher likelihood of default. Employment verification and job stability analysis should be prioritized during approval.

3. **Defaulters Due for Specific Purposes**
   Applicants seeking loans for "debt consolidation" purposes should be evaluated more carefully, as these purposes are common among defaulters.

4. **Home Ownership Analysis**
   Borrowers with "Rent", "Mortgage",home ownership types are more likely to default. "Mortgage" holders appear more reliable and can be treated as lower risk.

5. **Verification Status Checks**
   Even borrowers with "Verified" or "Not Verified" status have high default rates. This indicates that verification alone is not sufficient, and other risk indicators should be considered.

6. **Term-wise Risk Adjustment**
   Loans with a 36-month term show a higher default rate. Consider applying more strickly credit checks or insurance for these term loans.

7. **Time-Based Lending Strategy for Defaulters**
   Most defaults occurred in 2011, particularly in the months of October,November and December.

8. **Bankruptcy Is Not the Only Indicator**
   Most defaulters have **0 bankruptcies**, indicating that relying solely on bankruptcy history to judge credit risk is insufficient.

9. **Debt to Income Ratio**
   Most defaulters have **Medium debt ratio**, indicating that Normal dti ratio get higher defaulters.

10. **Most Common Purpose That Find In Defaulters**
    Most defaulters Purpose is **Debt-Consolidation**, indicating that Be Carefull Before Apply a loan for this Purpose.

In [ ]: