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

In [2]:
```
#import warnings
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
loan = pd.read_csv("EDA/loan.csv")
loan.head()
```

Out[3]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	insta
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 rows × 111 columns

Data Understanding

In [4]:
```
loan.head()
```

Out[4]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	insta
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 rows × 111 columns

In [5]:
```
loan.columns
```

Out[5]:
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       ...
       'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'pct_tl_nvr_dlq',
       'percent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens',
       'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit'],
      dtype='object', length=111)
```

Some of the important columns in the dataset are loan_amount, term, interest rate, grade, sub grade, annual income, purpose of the loan etc. The target variable, which we want to compare across the independent variables, is loan status. The strategy is to figure out compare the average default rates across various independent variables and identify the ones that affect default rate the most.

Data Cleaning

Some columns have a large number of missing values, let's first fix the missing values and then check for other types of data quality problems.

In [6]:
```
loan.isnull().sum()
```

Out[6]:
```
id                                0
member_id                         0
loan_amnt                         0
funded_amnt                       0
funded_amnt_inv                   0
                              ...
tax_liens                        39
tot_hi_cred_lim               39717
total_bal_ex_mort             39717
total_bc_limit                39717
total_il_high_credit_limit    39717
Length: 111, dtype: int64
```

In [7]:
```
round(loan.isnull().sum()/len(loan.index), 2)*100
```

Out[7]:
```
id                              0.0
member_id                       0.0
loan_amnt                       0.0
funded_amnt                     0.0
funded_amnt_inv                 0.0
                              ...
tax_liens                       0.0
tot_hi_cred_lim               100.0
total_bal_ex_mort             100.0
total_bc_limit                100.0
total_il_high_credit_limit    100.0
Length: 111, dtype: float64
```

You can see that many columns have 100% missing values, some have 65%, 33% etc. First, let's get rid of the columns having 100% missing values.

In [8]:
```
missing_columns = loan.columns[100*(loan.isnull().sum()/len(loan.index)) > 9
print(missing_columns)
```

```
Index(['mths_since_last_record', 'next_pymnt_d', 'mths_since_last_major_dero
g',
       'annual_inc_joint', 'dti_joint', 'verification_status_joint',
       'tot_coll_amt', 'tot_cur_bal', 'open_acc_6m', 'open_il_6m',
       'open_il_12m', 'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il',
       'il_util', 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util',
       'total_rev_hi_lim', 'inq_fi', 'total_cu_tl', 'inq_last_12m',
       'acc_open_past_24mths', 'avg_cur_bal', 'bc_open_to_buy', 'bc_util',
       'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_o
p',
       'mo_sin_rcnt_tl', 'mort_acc', 'mths_since_recent_bc',
       'mths_since_recent_bc_dlq', 'mths_since_recent_inq',
       'mths_since_recent_revol_delinq', 'num_accts_ever_120_pd',
       'num_actv_bc_tl', 'num_actv_rev_tl', 'num_bc_sats', 'num_bc_tl',
       'num_il_tl', 'num_op_rev_tl', 'num_rev_accts', 'num_rev_tl_bal_gt_0',
       'num_sats', 'num_tl_120dpd_2m', 'num_tl_30dpd', 'num_tl_90g_dpd_24m',
       'num_tl_op_past_12m', 'pct_tl_nvr_dlq', 'percent_bc_gt_75',
       'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit'],
      dtype='object')
```

In [9]:
```
loan = loan.drop(missing_columns, axis=1)
print(loan.shape)
```

```
(39717, 55)
```

In [10]:
```
100*(loan.isnull().sum()/len(loan.index))
```

Out[10]:

```
id                             0.000000
member_id                      0.000000
loan_amnt                      0.000000
funded_amnt                    0.000000
funded_amnt_inv                0.000000
term                           0.000000
int_rate                       0.000000
installment                    0.000000
grade                          0.000000
sub_grade                      0.000000
emp_title                      6.191303
emp_length                     2.706650
home_ownership                 0.000000
annual_inc                     0.000000
verification_status            0.000000
issue_d                        0.000000
loan_status                    0.000000
pymnt_plan                     0.000000
url                            0.000000
desc                          32.580507
purpose                        0.000000
title                          0.027696
zip_code                       0.000000
addr_state                     0.000000
dti                            0.000000
delinq_2yrs                    0.000000
earliest_cr_line               0.000000
inq_last_6mths                 0.000000
mths_since_last_delinq        64.662487
open_acc                       0.000000
pub_rec                        0.000000
revol_bal                      0.000000
revol_util                     0.125891
total_acc                      0.000000
initial_list_status            0.000000
out_prncp                      0.000000
out_prncp_inv                  0.000000
total_pymnt                    0.000000
total_pymnt_inv                0.000000
total_rec_prncp                0.000000
total_rec_int                  0.000000
total_rec_late_fee             0.000000
recoveries                     0.000000
collection_recovery_fee        0.000000
last_pymnt_d                   0.178765
last_pymnt_amnt                0.000000
last_credit_pull_d             0.005036
collections_12_mths_ex_med     0.140998
policy_code                    0.000000
application_type               0.000000
acc_now_delinq                 0.000000
chargeoff_within_12_mths       0.140998
delinq_amnt                    0.000000
pub_rec_bankruptcies           1.754916
tax_liens                      0.098195
dtype: float64
```

In [11]:

```
loan.loc[:, ['desc', 'mths_since_last_delinq']].head()
```

Out[11]:

	desc	mths_since_last_delinq
0	Borrower added on 12/22/11 > I need to upgra...	NaN
1	Borrower added on 12/22/11 > I plan to use t...	NaN
2	NaN	NaN
3	Borrower added on 12/21/11 > to pay for prop...	35.0
4	Borrower added on 12/21/11 > I plan on combi...	38.0

The column description contains the comments the applicant had written while applying for the loan. Although one can use some text analysis techniques to derive new features from this column (such as sentiment, number of positive/negative words etc.), we will not use this column in this analysis.

Secondly, months since last delinquent represents the number months passed since the person last fell into the 90 DPD group. There is an important reason we shouldn't use this column in analysis - since at the time of loan application, we will not have this data (it gets generated months after the loan has been approved), it cannot be used as a predictor of default at the time of loan approval.

Thus let's drop the two columns.

In [12]:
```
loan = loan.drop(['desc', 'mths_since_last_delinq'], axis=1)
```

In [13]:
```
100*(loan.isnull().sum()/len(loan.index))
```

Out[13]:

```
id                            0.000000
member_id                     0.000000
loan_amnt                     0.000000
funded_amnt                   0.000000
funded_amnt_inv               0.000000
term                          0.000000
int_rate                      0.000000
installment                   0.000000
grade                         0.000000
sub_grade                     0.000000
emp_title                     6.191303
emp_length                    2.706650
home_ownership                0.000000
annual_inc                    0.000000
verification_status           0.000000
issue_d                       0.000000
loan_status                   0.000000
pymnt_plan                    0.000000
url                           0.000000
purpose                       0.000000
title                         0.027696
zip_code                      0.000000
addr_state                    0.000000
dti                           0.000000
delinq_2yrs                   0.000000
earliest_cr_line              0.000000
inq_last_6mths                0.000000
open_acc                      0.000000
pub_rec                       0.000000
revol_bal                     0.000000
revol_util                    0.125891
total_acc                     0.000000
initial_list_status           0.000000
out_prncp                     0.000000
out_prncp_inv                 0.000000
total_pymnt                   0.000000
total_pymnt_inv               0.000000
total_rec_prncp               0.000000
total_rec_int                 0.000000
total_rec_late_fee            0.000000
recoveries                    0.000000
collection_recovery_fee       0.000000
last_pymnt_d                  0.178765
last_pymnt_amnt               0.000000
last_credit_pull_d            0.005036
collections_12_mths_ex_med    0.140998
policy_code                   0.000000
application_type              0.000000
acc_now_delinq                0.000000
chargeoff_within_12_mths      0.140998
delinq_amnt                   0.000000
pub_rec_bankruptcies          1.754916
tax_liens                     0.098195
dtype: float64
```

There are some more columns with missing values, but let's ignore them for now (since we are not doing any modeling, we don't need to impute all missing values anyway).

But let's check whether some rows have a large number of missing values.

In [14]:

```
loan.isnull().sum(axis=1)
```

Out[14]:
```
0        1
1        0
2        1
3        0
4        0
        ..
39712    4
39713    4
39714    5
39715    5
39716    4
Length: 39717, dtype: int64
```

In [15]:
```
len(loan[loan.isnull().sum(axis=1) > 5].index)
```

Out[15]:
```
0
```

The data looks clean by and large. Let's also check whether all columns are in the correct format.

In [16]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 53 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  purpose                     39717 non-null  object 
 20  title                       39706 non-null  object 
 21  zip_code                    39717 non-null  object 
 22  addr_state                  39717 non-null  object 
 23  dti                         39717 non-null  float64
 24  delinq_2yrs                 39717 non-null  int64  
 25  earliest_cr_line            39717 non-null  object 
 26  inq_last_6mths              39717 non-null  int64  
 27  open_acc                    39717 non-null  int64  
 28  pub_rec                     39717 non-null  int64  
 29  revol_bal                   39717 non-null  int64  
 30  revol_util                  39667 non-null  object 
 31  total_acc                   39717 non-null  int64  
 32  initial_list_status         39717 non-null  object 
 33  out_prncp                   39717 non-null  float64
 34  out_prncp_inv               39717 non-null  float64
 35  total_pymnt                 39717 non-null  float64
 36  total_pymnt_inv             39717 non-null  float64
 37  total_rec_prncp             39717 non-null  float64
 38  total_rec_int               39717 non-null  float64
 39  total_rec_late_fee          39717 non-null  float64
 40  recoveries                  39717 non-null  float64
 41  collection_recovery_fee     39717 non-null  float64
 42  last_pymnt_d                39646 non-null  object 
 43  last_pymnt_amnt             39717 non-null  float64
 44  last_credit_pull_d          39715 non-null  object 
 45  collections_12_mths_ex_med  39661 non-null  float64
 46  policy_code                 39717 non-null  int64  
 47  application_type            39717 non-null  object 
 48  acc_now_delinq              39717 non-null  int64  
 49  chargeoff_within_12_mths    39661 non-null  float64
 50  delinq_amnt                 39717 non-null  int64  
 51  pub_rec_bankruptcies        39020 non-null  float64
 52  tax_liens                   39678 non-null  float64
dtypes: float64(18), int64(13), object(22)
memory usage: 16.1+ MB
```

In [17]:
```
loan['int_rate'] = loan['int_rate'].apply(lambda x: pd.to_numeric(x.split("%
```

In [18]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 53 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_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  purpose                     39717 non-null  object 
 20  title                       39706 non-null  object 
 21  zip_code                    39717 non-null  object 
 22  addr_state                  39717 non-null  object 
 23  dti                         39717 non-null  float64
 24  delinq_2yrs                 39717 non-null  int64  
 25  earliest_cr_line            39717 non-null  object 
 26  inq_last_6mths              39717 non-null  int64  
 27  open_acc                    39717 non-null  int64  
 28  pub_rec                     39717 non-null  int64  
 29  revol_bal                   39717 non-null  int64  
 30  revol_util                  39667 non-null  object 
 31  total_acc                   39717 non-null  int64  
 32  initial_list_status         39717 non-null  object 
 33  out_prncp                   39717 non-null  float64
 34  out_prncp_inv               39717 non-null  float64
 35  total_pymnt                 39717 non-null  float64
 36  total_pymnt_inv             39717 non-null  float64
 37  total_rec_prncp             39717 non-null  float64
 38  total_rec_int               39717 non-null  float64
 39  total_rec_late_fee          39717 non-null  float64
 40  recoveries                  39717 non-null  float64
 41  collection_recovery_fee     39717 non-null  float64
 42  last_pymnt_d                39646 non-null  object 
 43  last_pymnt_amnt             39717 non-null  float64
 44  last_credit_pull_d          39715 non-null  object 
 45  collections_12_mths_ex_med  39661 non-null  float64
 46  policy_code                 39717 non-null  int64  
 47  application_type            39717 non-null  object 
 48  acc_now_delinq              39717 non-null  int64  
 49  chargeoff_within_12_mths    39661 non-null  float64
 50  delinq_amnt                 39717 non-null  int64  
 51  pub_rec_bankruptcies        39020 non-null  float64
 52  tax_liens                   39678 non-null  float64
dtypes: float64(19), int64(13), object(21)
memory usage: 16.1+ MB
```

In [19]:
```
loan = loan[~loan['emp_length'].isnull()]
import re
loan['emp_length'] = loan['emp_length'].apply(lambda x: re.findall('\d+', st
loan['emp_length'] = loan['emp_length'].apply(lambda x: pd.to_numeric(x))
```

In [20]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38642 entries, 0 to 39716
Data columns (total 53 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   id                          38642 non-null  int64  
 1   member_id                   38642 non-null  int64  
 2   loan_amnt                   38642 non-null  int64  
 3   funded_amnt                 38642 non-null  int64  
 4   funded_amnt_inv             38642 non-null  float64
 5   term                        38642 non-null  object 
 6   int_rate                    38642 non-null  float64
 7   installment                 38642 non-null  float64
 8   grade                       38642 non-null  object 
 9   sub_grade                   38642 non-null  object 
 10  emp_title                   37202 non-null  object 
 11  emp_length                  38642 non-null  int64  
 12  home_ownership              38642 non-null  object 
 13  annual_inc                  38642 non-null  float64
 14  verification_status         38642 non-null  object 
 15  issue_d                     38642 non-null  object 
 16  loan_status                 38642 non-null  object 
 17  pymnt_plan                  38642 non-null  object 
 18  url                         38642 non-null  object 
 19  purpose                     38642 non-null  object 
 20  title                       38632 non-null  object 
 21  zip_code                    38642 non-null  object 
 22  addr_state                  38642 non-null  object 
 23  dti                         38642 non-null  float64
 24  delinq_2yrs                 38642 non-null  int64  
 25  earliest_cr_line            38642 non-null  object 
 26  inq_last_6mths              38642 non-null  int64  
 27  open_acc                    38642 non-null  int64  
 28  pub_rec                     38642 non-null  int64  
 29  revol_bal                   38642 non-null  int64  
 30  revol_util                  38595 non-null  object 
 31  total_acc                   38642 non-null  int64  
 32  initial_list_status         38642 non-null  object 
 33  out_prncp                   38642 non-null  float64
 34  out_prncp_inv               38642 non-null  float64
 35  total_pymnt                 38642 non-null  float64
 36  total_pymnt_inv             38642 non-null  float64
 37  total_rec_prncp             38642 non-null  float64
 38  total_rec_int               38642 non-null  float64
 39  total_rec_late_fee          38642 non-null  float64
 40  recoveries                  38642 non-null  float64
 41  collection_recovery_fee     38642 non-null  float64
 42  last_pymnt_d                38576 non-null  object 
 43  last_pymnt_amnt             38642 non-null  float64
 44  last_credit_pull_d          38640 non-null  object 
 45  collections_12_mths_ex_med  38586 non-null  float64
 46  policy_code                 38642 non-null  int64  
 47  application_type            38642 non-null  object 
 48  acc_now_delinq              38642 non-null  int64  
 49  chargeoff_within_12_mths    38586 non-null  float64
 50  delinq_amnt                 38642 non-null  int64  
 51  pub_rec_bankruptcies        37945 non-null  float64
 52  tax_liens                   38603 non-null  float64
dtypes: float64(19), int64(14), object(20)
memory usage: 15.9+ MB
```

Data Analysis

Let's now move to data analysis. To start with, let's understand the objective of the analysis clearly and identify the variables that we want to consider for 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. Now, there are broadly three types of variables - 1. those which are related to the applicant (demographic variables such as age, occupation, employment details etc.), 2. loan characteristics (amount of loan, interest rate, purpose of loan etc.) and 3. Customer behaviour variables (those which are generated after the loan is approved such as delinquent 2 years, revolving balance, next payment date etc.).

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

Thus, going forward, we will use only the other two types of variables.

In [21]:
```
behaviour_var =  [
  "delinq_2yrs",
  "earliest_cr_line",
  "inq_last_6mths",
  "open_acc",
  "pub_rec",
  "revol_bal",
  "revol_util",
  "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_d",
  "last_pymnt_amnt",
  "last_credit_pull_d",
  "application_type"]
behaviour_var
```

Out[21]:
```
['delinq_2yrs',
 'earliest_cr_line',
 'inq_last_6mths',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 '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_d',
 'last_pymnt_amnt',
 'last_credit_pull_d',
 'application_type']
```

In [22]:
```
df = loan.drop(behaviour_var, axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38642 entries, 0 to 39716
Data columns (total 32 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   id                          38642 non-null  int64  
 1   member_id                   38642 non-null  int64  
 2   loan_amnt                   38642 non-null  int64  
 3   funded_amnt                 38642 non-null  int64  
 4   funded_amnt_inv             38642 non-null  float64
 5   term                        38642 non-null  object 
 6   int_rate                    38642 non-null  float64
 7   installment                 38642 non-null  float64
 8   grade                       38642 non-null  object 
 9   sub_grade                   38642 non-null  object 
 10  emp_title                   37202 non-null  object 
 11  emp_length                  38642 non-null  int64  
 12  home_ownership              38642 non-null  object 
 13  annual_inc                  38642 non-null  float64
 14  verification_status         38642 non-null  object 
 15  issue_d                     38642 non-null  object 
 16  loan_status                 38642 non-null  object 
 17  pymnt_plan                  38642 non-null  object 
 18  url                         38642 non-null  object 
 19  purpose                     38642 non-null  object 
 20  title                       38632 non-null  object 
 21  zip_code                    38642 non-null  object 
 22  addr_state                  38642 non-null  object 
 23  dti                         38642 non-null  float64
 24  initial_list_status         38642 non-null  object 
 25  collections_12_mths_ex_med  38586 non-null  float64
 26  policy_code                 38642 non-null  int64  
 27  acc_now_delinq              38642 non-null  int64  
 28  chargeoff_within_12_mths    38586 non-null  float64
 29  delinq_amnt                 38642 non-null  int64  
 30  pub_rec_bankruptcies        37945 non-null  float64
 31  tax_liens                   38603 non-null  float64
dtypes: float64(9), int64(8), object(15)
memory usage: 9.7+ MB
```

Typically, variables such as acc_now_delinquent, chargeoff within 12 months etc. (which are related to the applicant's past loans) are available from the credit bureau.

In [23]:
```
df = df.drop(['title', 'url', 'zip_code', 'addr_state'], axis=1)
```

Next, let's have a look at the target variable - loan_status. We need to relabel the values to a binary form - 0 or 1, 1 indicating that the person has defaulted and 0 otherwise.

In [24]:
```
df['loan_status'] = df['loan_status'].astype('category')
df['loan_status'].value_counts()
```

Out[24]:
```
Fully Paid     32145
Charged Off     5399
Current         1098
Name: loan_status, dtype: int64
```

We can see that fully paid comprises most of the loans. The ones marked 'current' are neither fully paid not defaulted, so let's get rid of the current loans. Also, let's tag the other two values as 0 or 1.

In [25]:
```
df = df[df['loan_status'] != 'Current']
df['loan_status'] = df['loan_status'].apply(lambda x: 0 if x=='Fully Paid' e
df['loan_status'] = df['loan_status'].apply(lambda x: pd.to_numeric(x))
df['loan_status'].value_counts()
```

Out[25]:
```
0    32145
1     5399
Name: loan_status, dtype: int64
```

Next, let's start with univariate analysis and then move to bivariate analysis.

Univariate Analysis

First, let's look at the overall default rate.

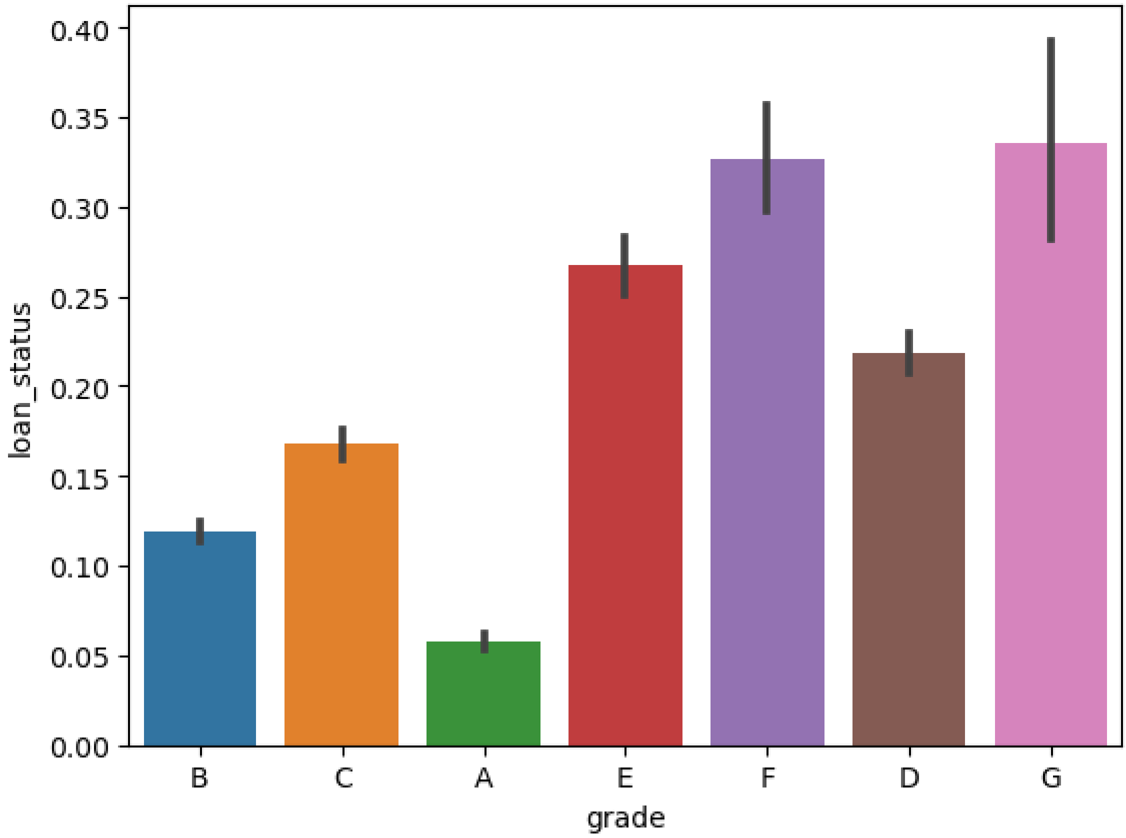
In [26]:
```
round(np.mean(df['loan_status']), 2)
```

Out[26]:
```
0.14
```

The overall default rate is about 14%.

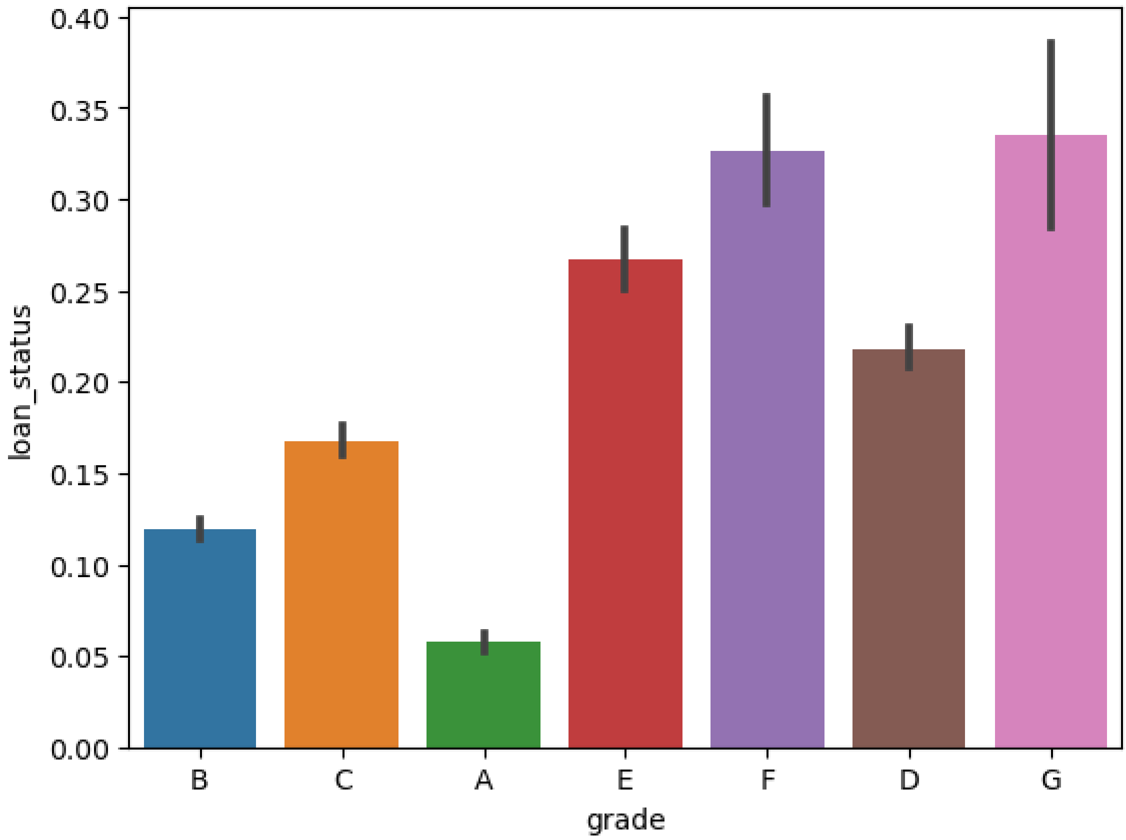
Let's first visualise the average default rates across categorical variables.

In [27]:
```
sns.barplot(x='grade', y='loan_status', data=df)
plt.show()
```



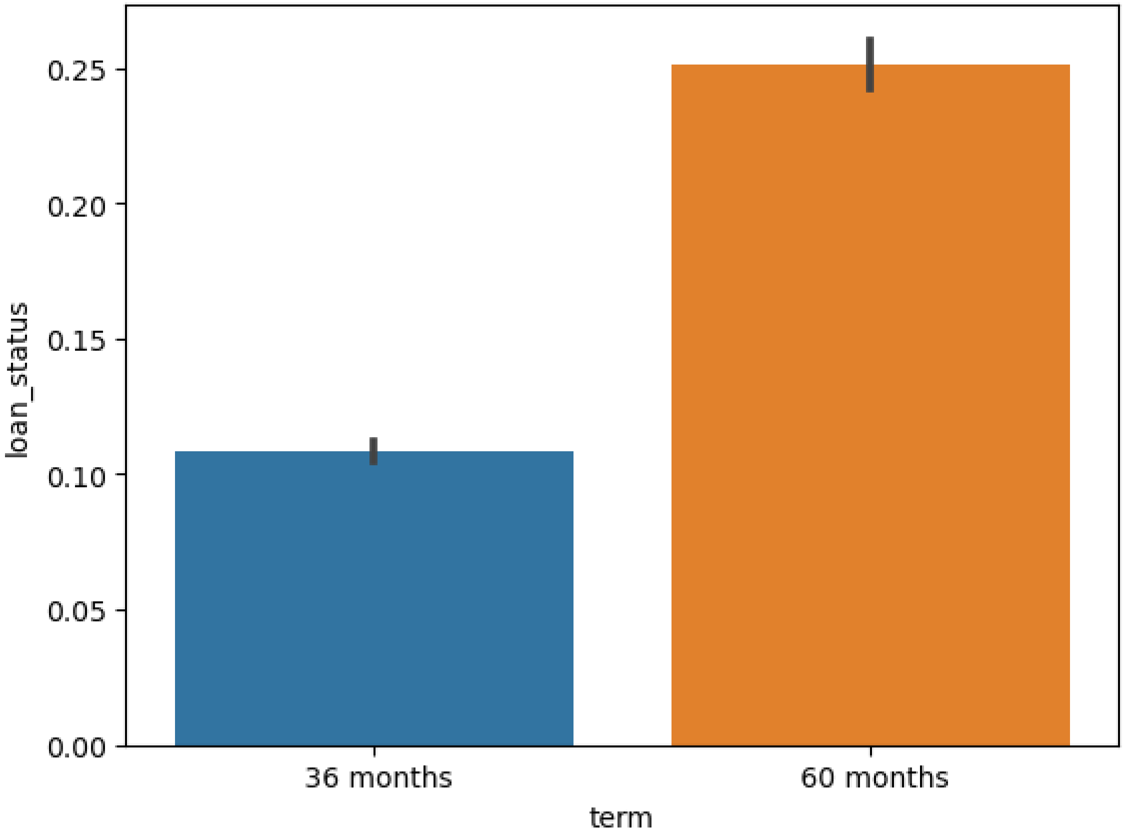
In [28]:
```
def plot_cat(cat_var):
    sns.barplot(x=cat_var, y='loan_status', data=df)
    plt.show()
```

In [29]:
```
plot_cat('grade')
```

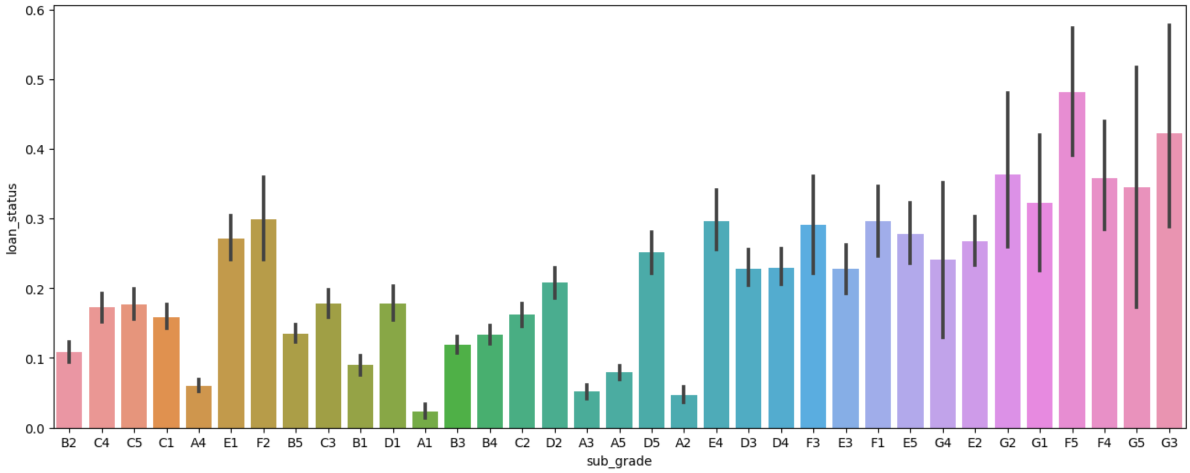


Clearly, as the grade of loan goes from A to G, the default rate increases. This is expected because the grade is decided by Lending Club based on the riskiness of the loan.

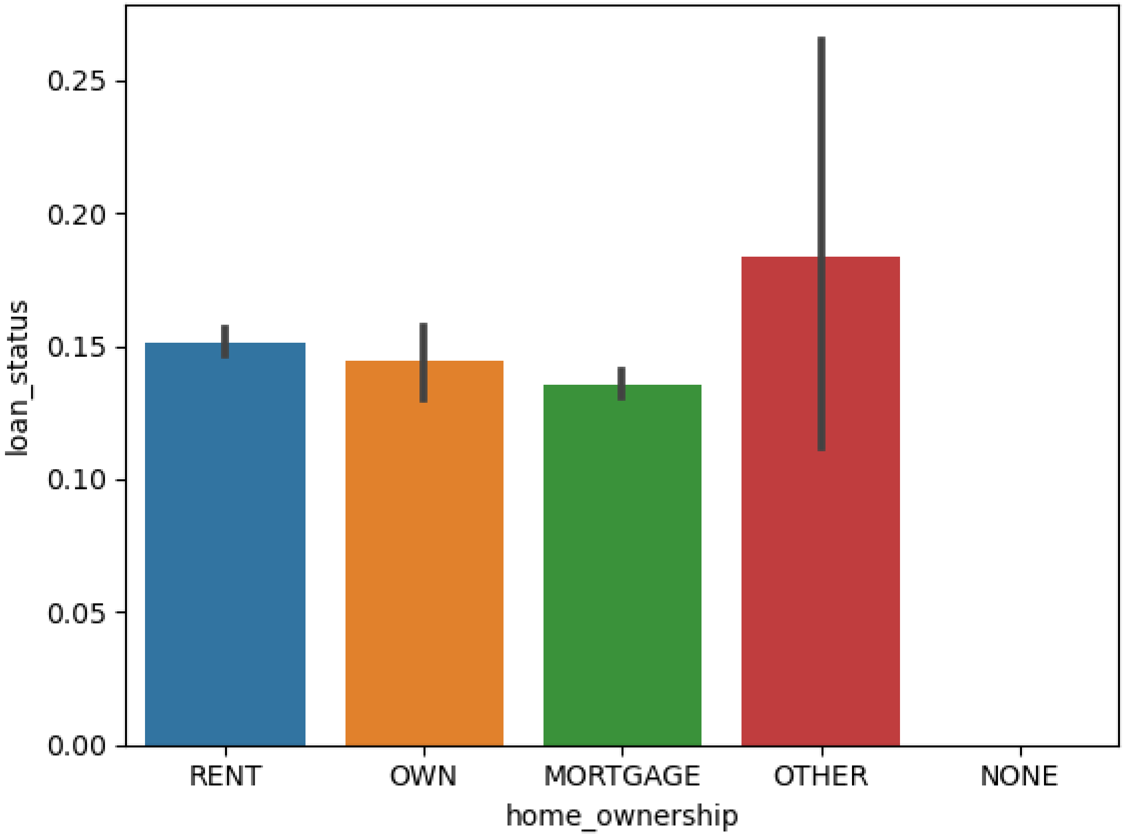
In [30]:
```
plot_cat('term')
```



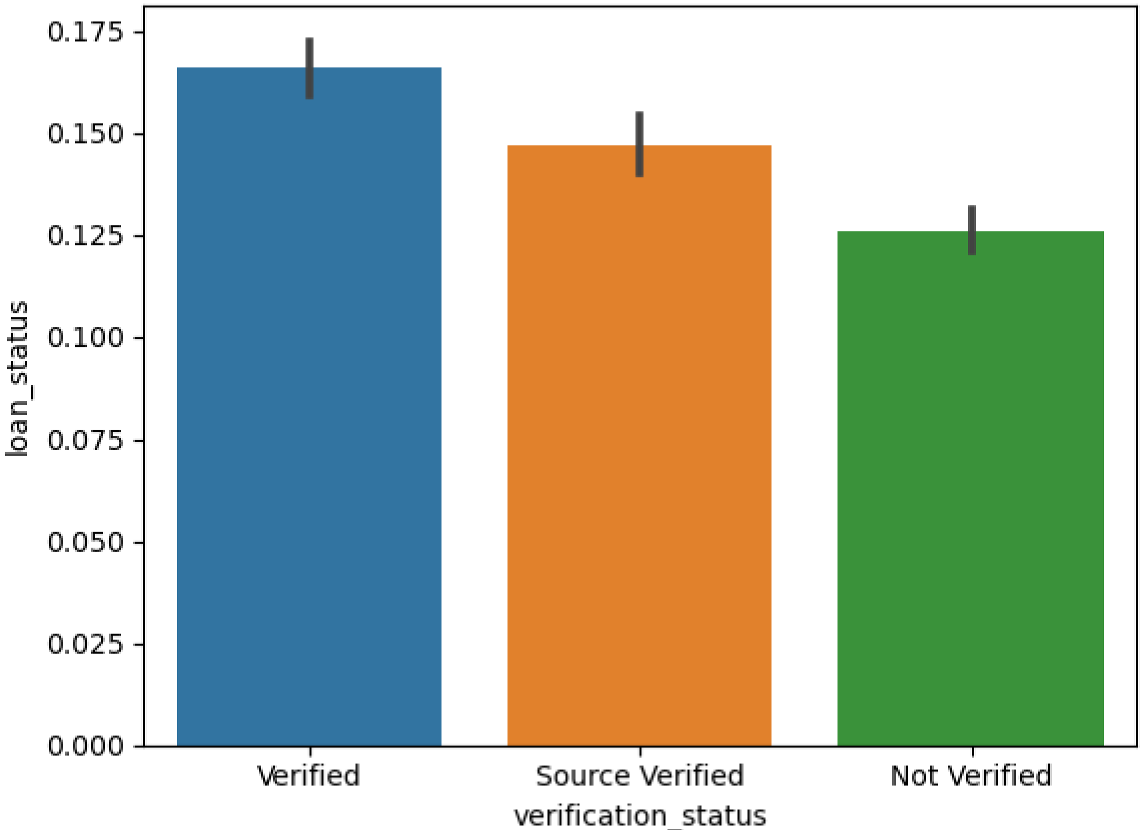
In [31]:
```
plt.figure(figsize=(16, 6))
plot_cat('sub_grade')
```



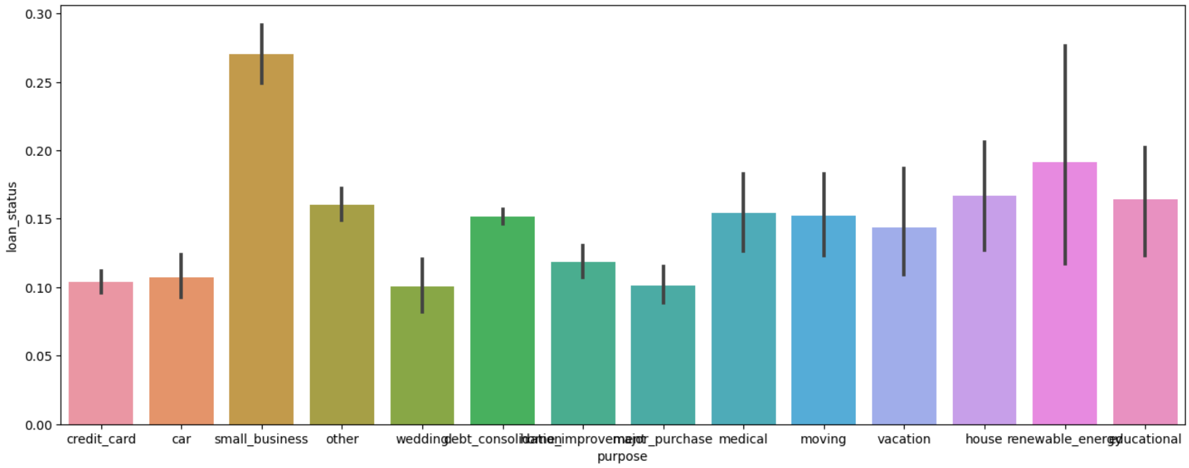
In [32]:
```
plot_cat('home_ownership')
```



In [33]:
```
plot_cat('verification_status')
```



In [34]:
```
plt.figure(figsize=(16, 6))
plot_cat('purpose')
```



In [35]:
```
df['issue_d'].head()
```

Out[35]:
```
0    Dec-11
1    Dec-11
2    Dec-11
3    Dec-11
5    Dec-11
Name: issue_d, dtype: object
```

In [36]:
```
from datetime import datetime
df['issue_d'] = df['issue_d'].apply(lambda x: datetime.strptime(x, '%b-%y'))
```

In [37]:
```
df['month'] = df['issue_d'].apply(lambda x: x.month)
df['year'] = df['issue_d'].apply(lambda x: x.year)
```

In [38]:
```
df.groupby('year').year.count()
```

Out[38]:
```
year
2007      251
2008     1562
2009     4716
2010    11214
2011    19801
Name: year, dtype: int64
```

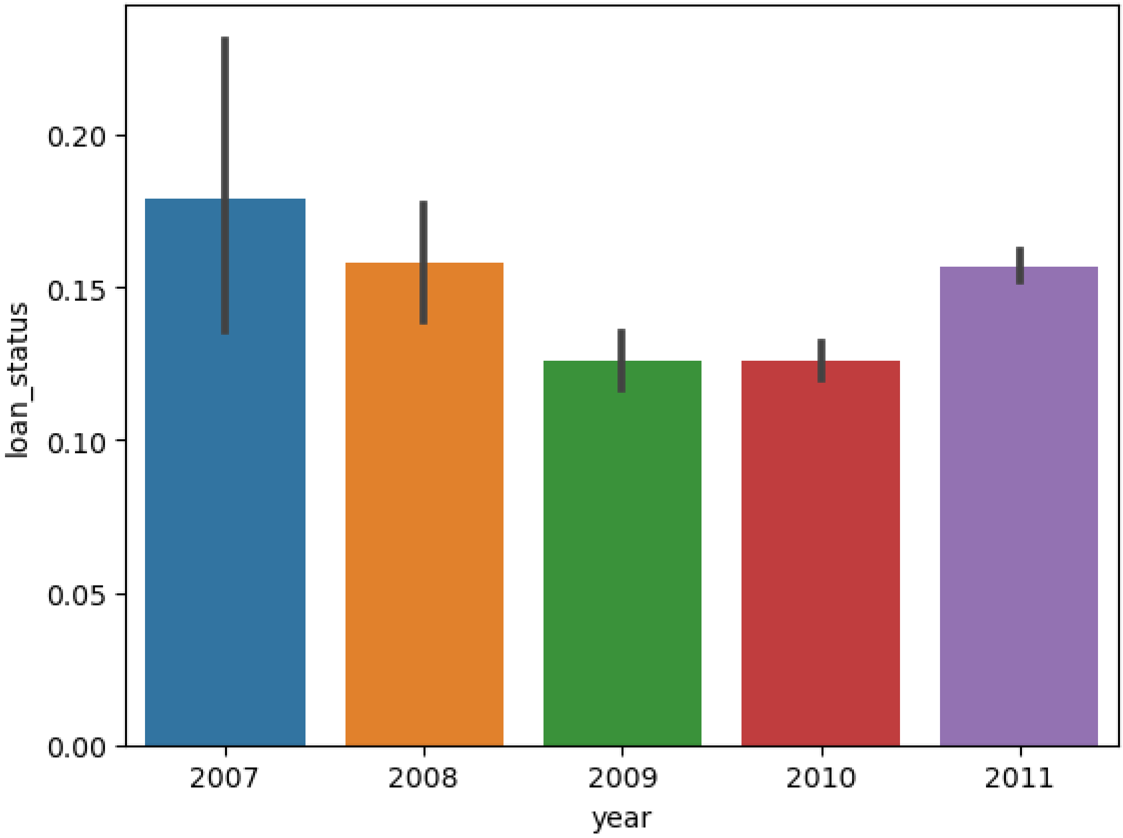
You can see that the number of loans has increased steadily across years.

In [39]:
```
df.groupby('month').month.count()
```

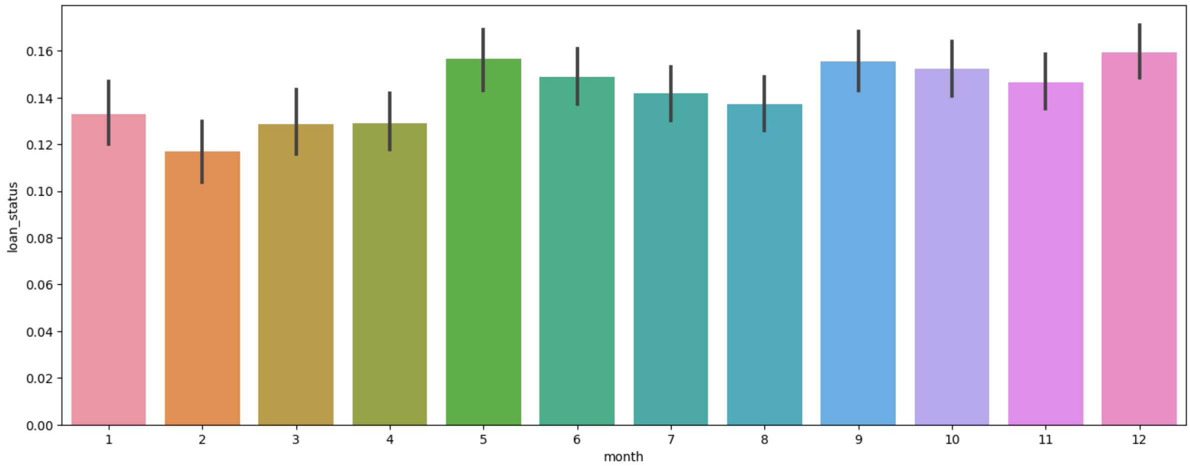
Out[39]:
```
month
1     2331
2     2278
3     2632
4     2756
5     2838
6     3094
7     3253
8     3321
9     3394
10    3637
11    3890
12    4120
Name: month, dtype: int64
```

Most loans are granted in December, and in general in the latter half of the year.

In [40]:
```
plot_cat('year')
```

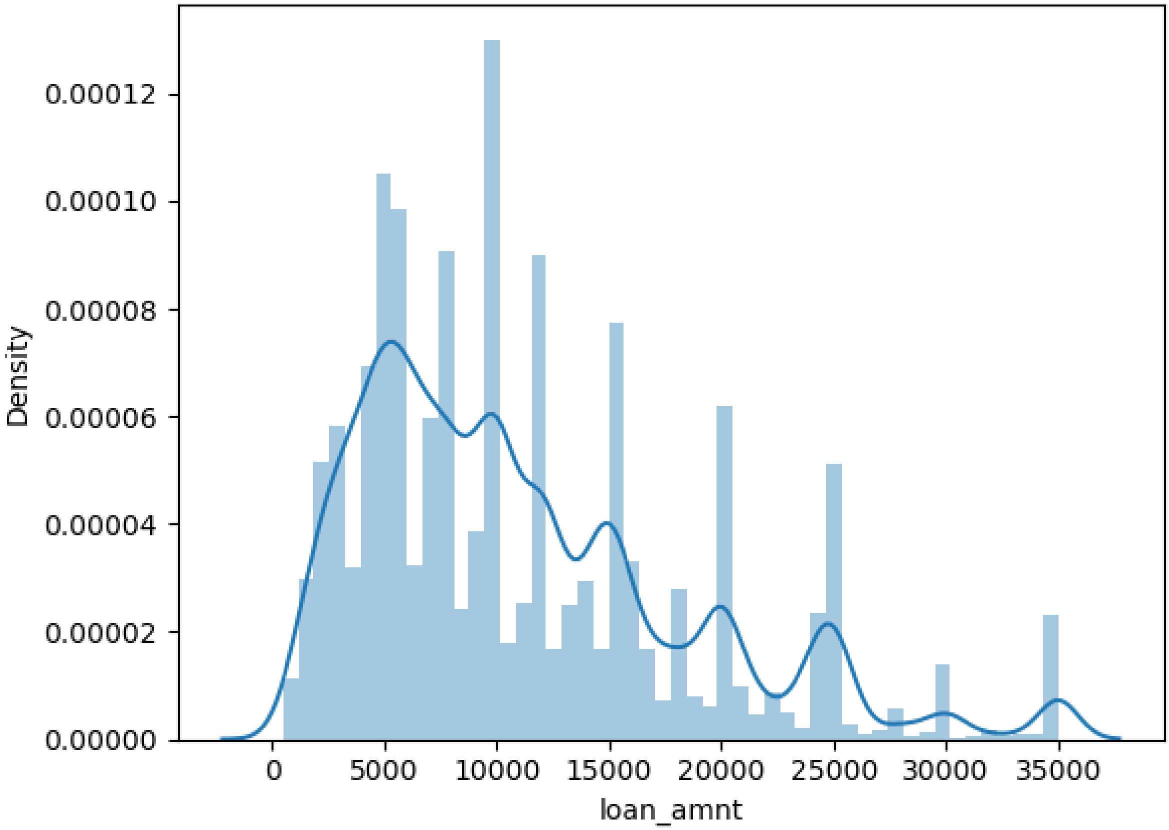


In [41]:
```
plt.figure(figsize=(16, 6))
plot_cat('month')
```



Let's now analyse how the default rate varies across continuous variables.

In [42]:
```
sns.distplot(df['loan_amnt'])
plt.show()
```



The easiest way to analyse how default rates vary across continous variables is to bin the variables into discrete categories.

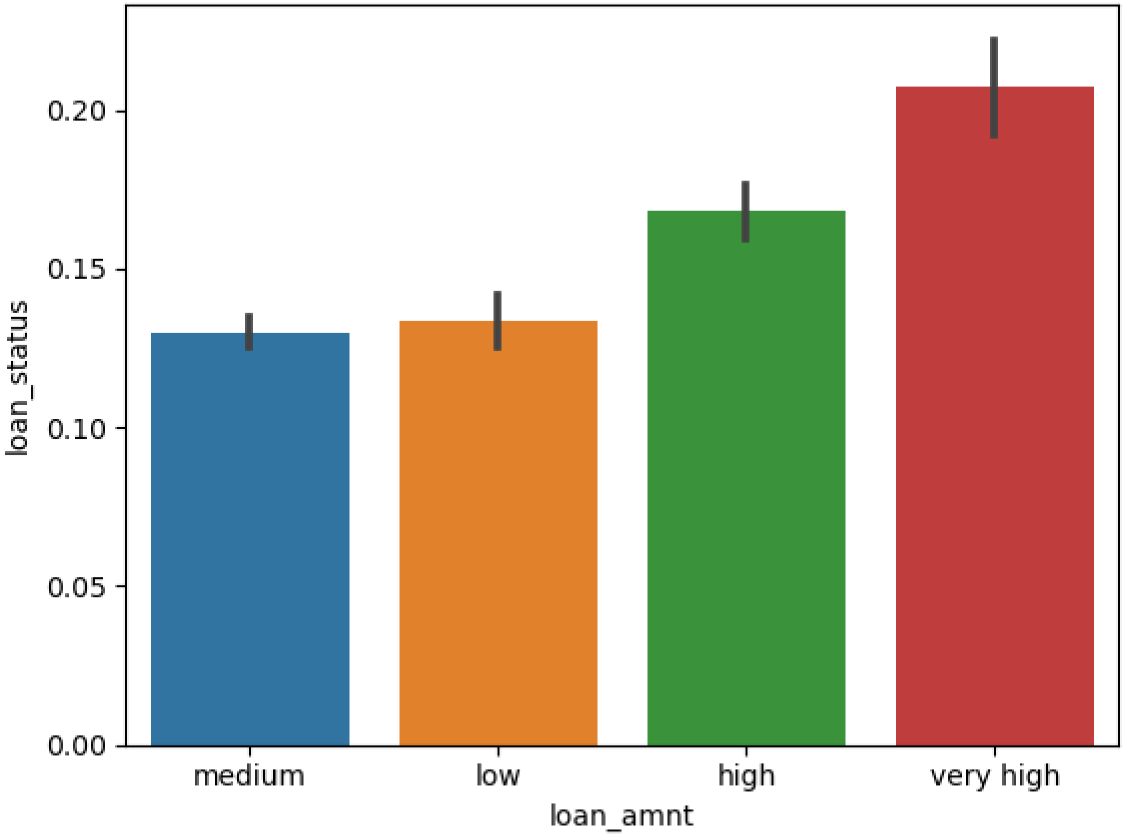
Let's bin the loan amount variable into small, medium, high, very high.

In [43]:
```
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'

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

In [44]:
```
df['loan_amnt'].value_counts()
```

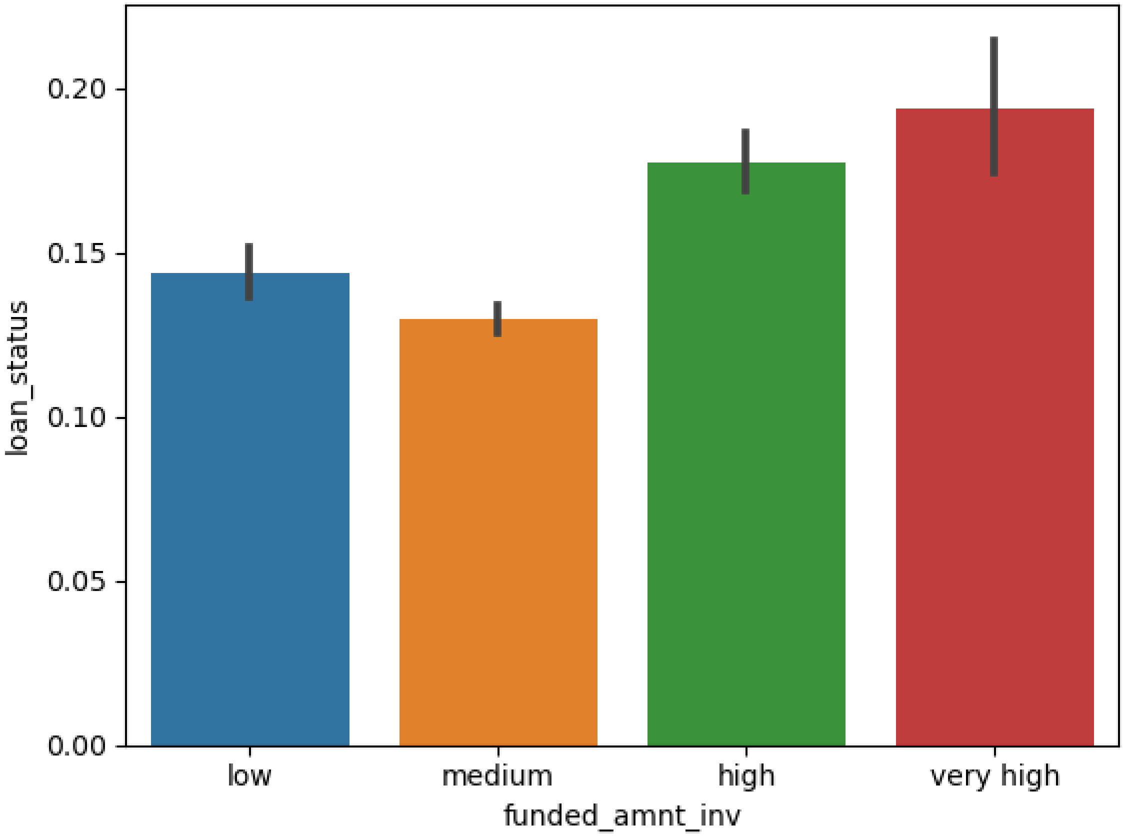
Out[44]:
```
medium       20157
high          7572
low           7095
very high     2720
Name: loan_amnt, dtype: int64
```

In [45]:
```
plot_cat('loan_amnt')
```



In [46]:
```
df['funded_amnt_inv'] = df['funded_amnt_inv'].apply(lambda x: loan_amount(x)
```

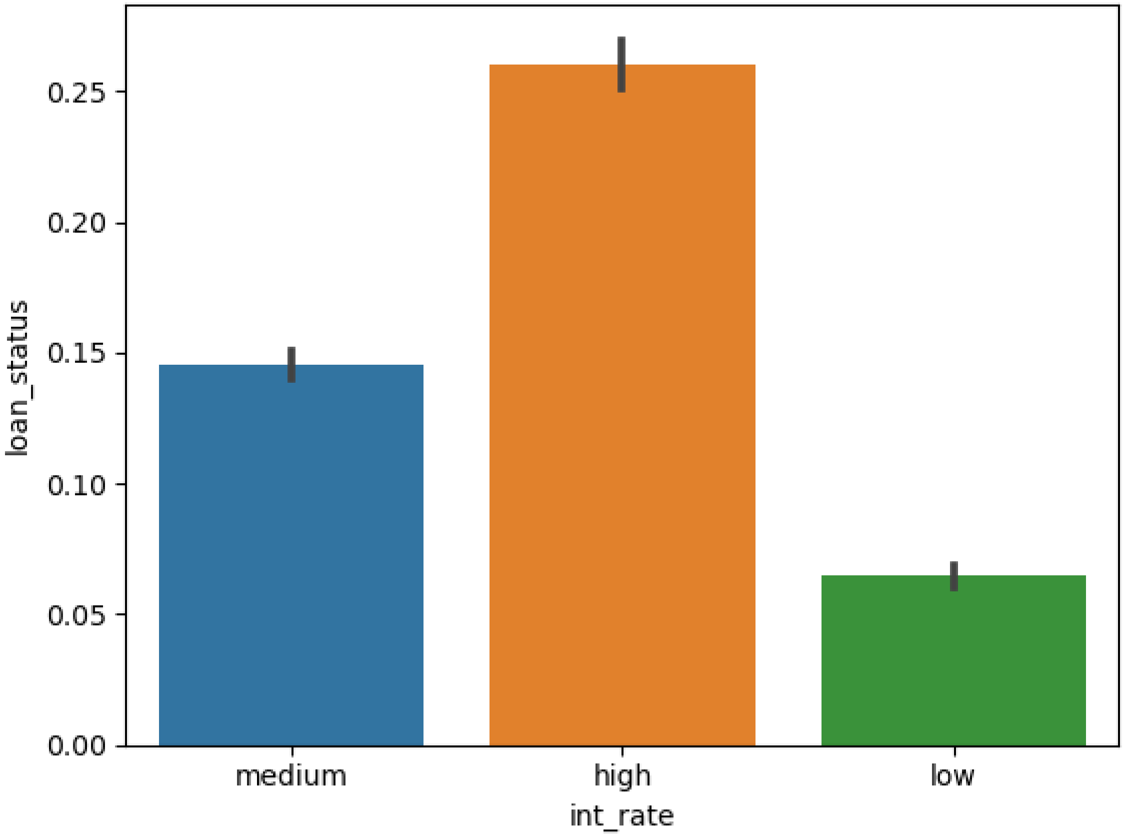
In [47]:
```
plot_cat('funded_amnt_inv')
```



In [48]:
```
def int_rate(n):
    if n <= 10:
        return 'low'
    elif n > 10 and n <=15:
```

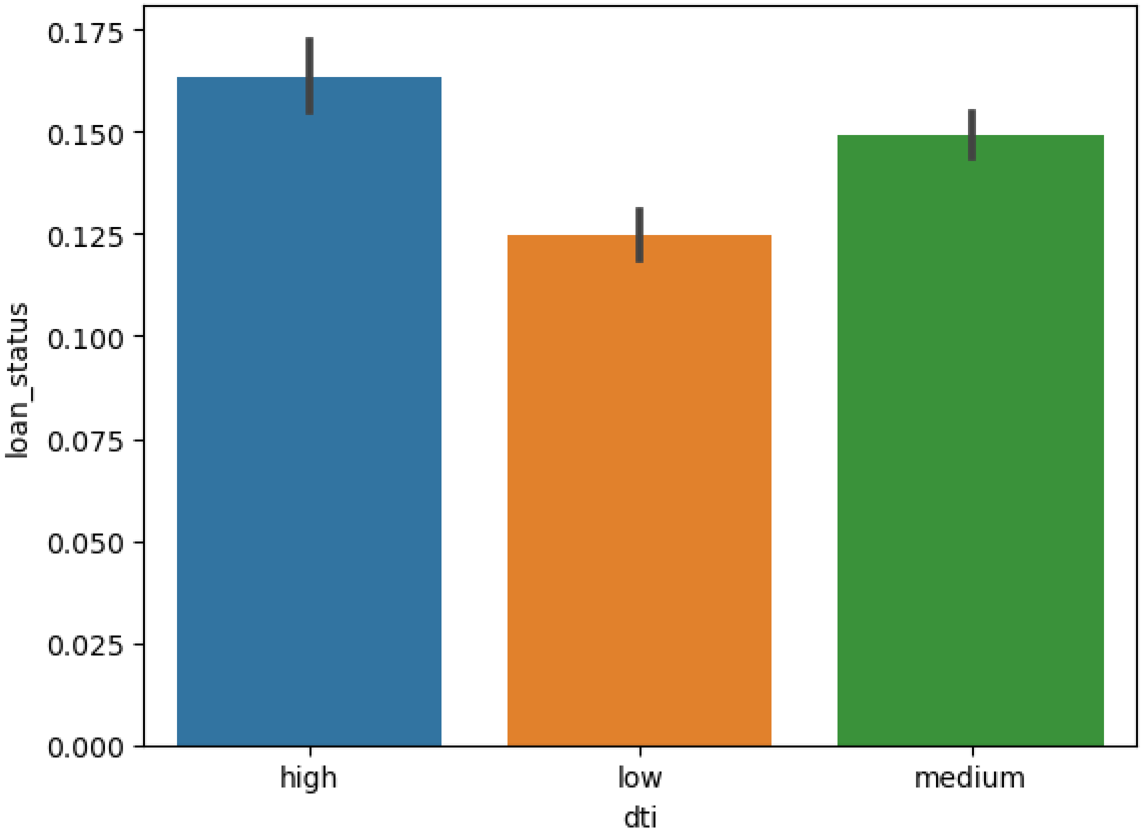
```
        return 'medium'
    else:
        return 'high'


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

In [49]:
```
plot_cat('int_rate')
```



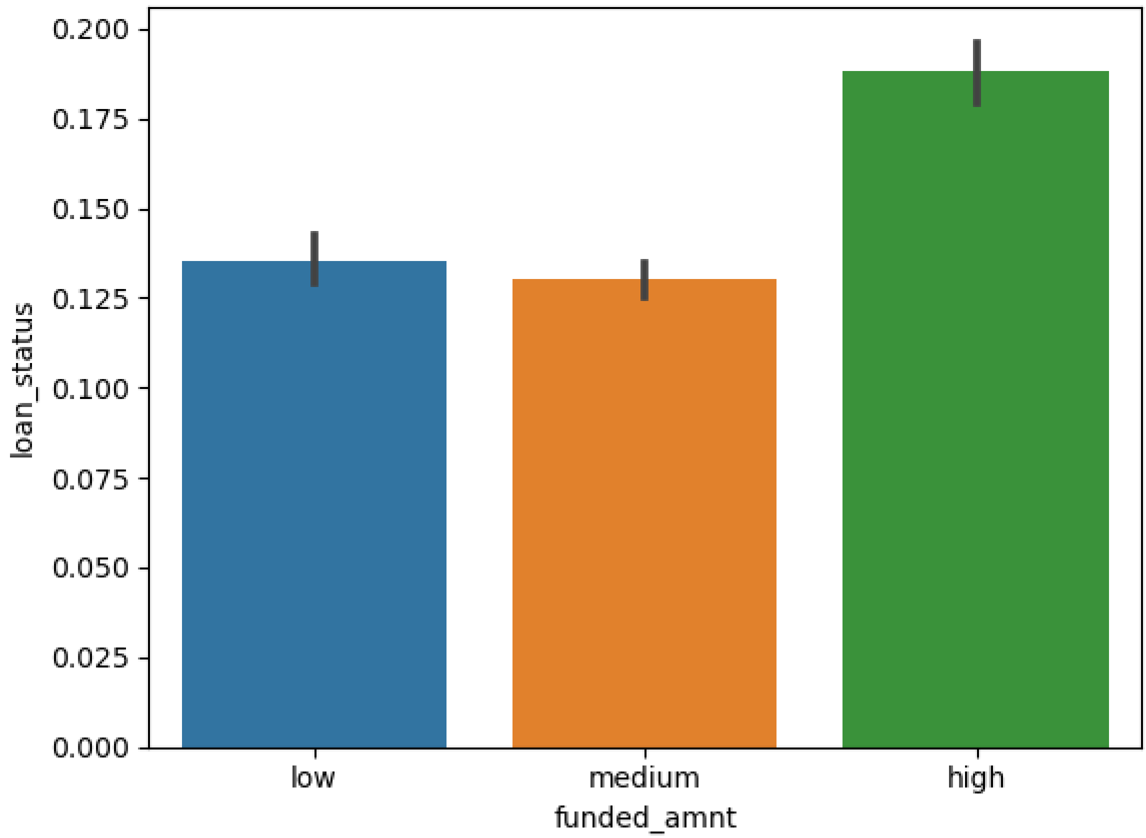
In [50]:
```
def dti(n):
    if n <= 10:
        return 'low'
    elif n > 10 and n <=20:
        return 'medium'
    else:
        return 'high'


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

In [51]:
```
plot_cat('dti')
```



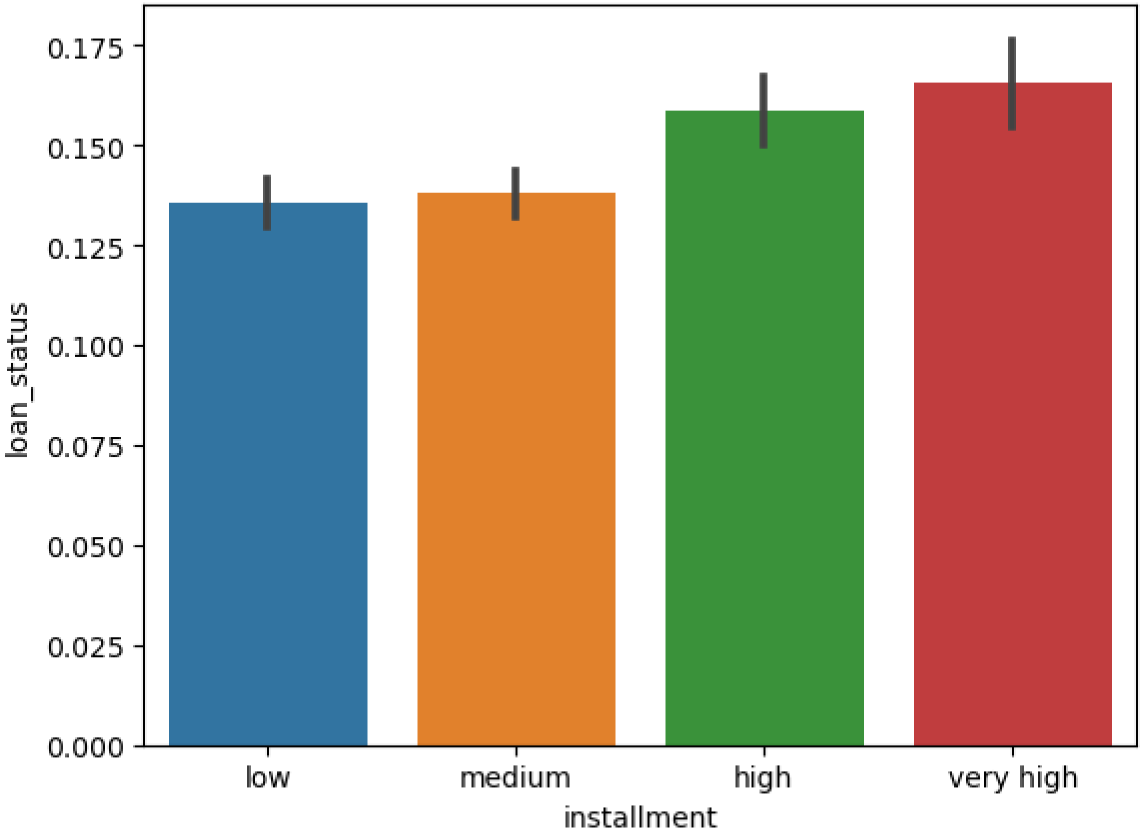
In [52]:
```
def funded_amount(n):
    if n <= 5000:
        return 'low'
    elif n > 5000 and n <=15000:
        return 'medium'
    else:
        return 'high'

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

In [53]:
```
plot_cat('funded_amnt')
```



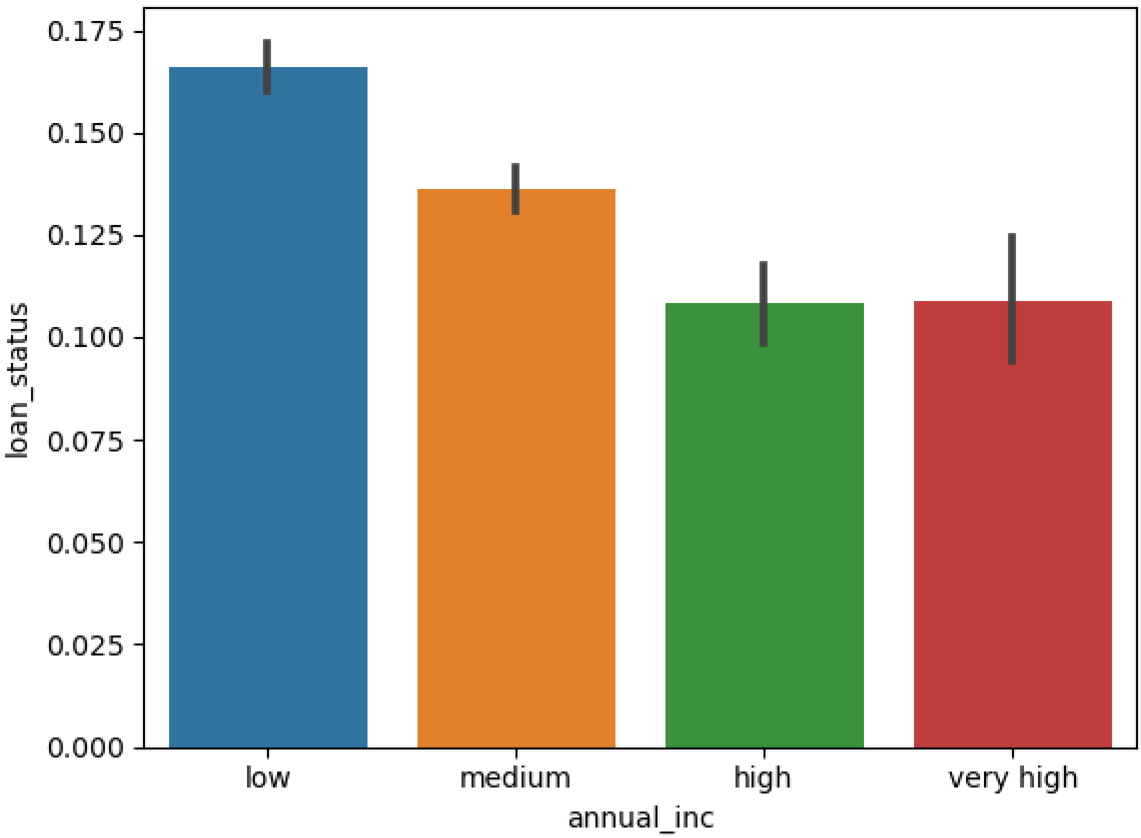
In [54]:
```
def installment(n):
    if n <= 200:
        return 'low'
    elif n > 200 and n <=400:
        return 'medium'
    elif n > 400 and n <=600:
        return 'high'
    else:
        return 'very high'

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

In [55]:
```
plot_cat('installment')
```



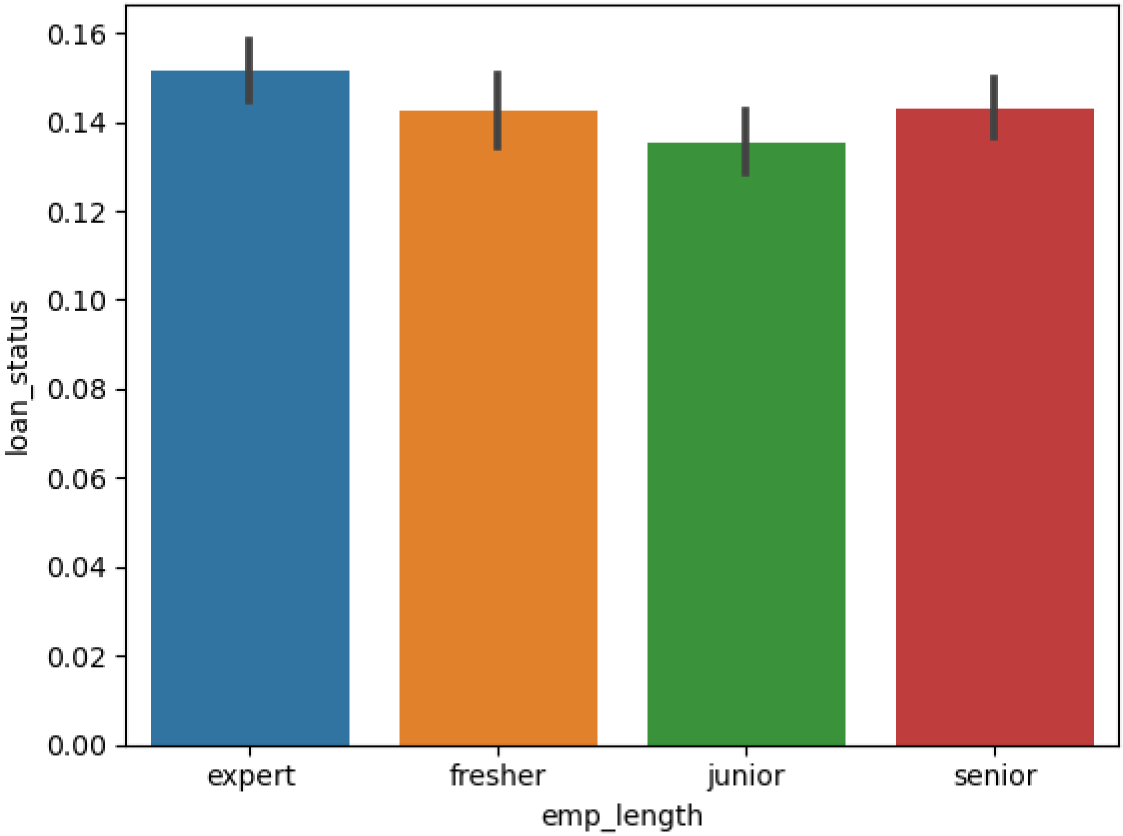
In [56]:
```
def annual_income(n):
    if n <= 50000:
        return 'low'
    elif n > 50000 and n <=100000:
        return 'medium'
    elif n > 100000 and n <=150000:
        return 'high'
    else:
        return 'very high'

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

In [57]:
```
plot_cat('annual_inc')
```



In [58]:
```
df = df[~df['emp_length'].isnull()]
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'

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

In [59]:
```
plot_cat('emp_length')
```

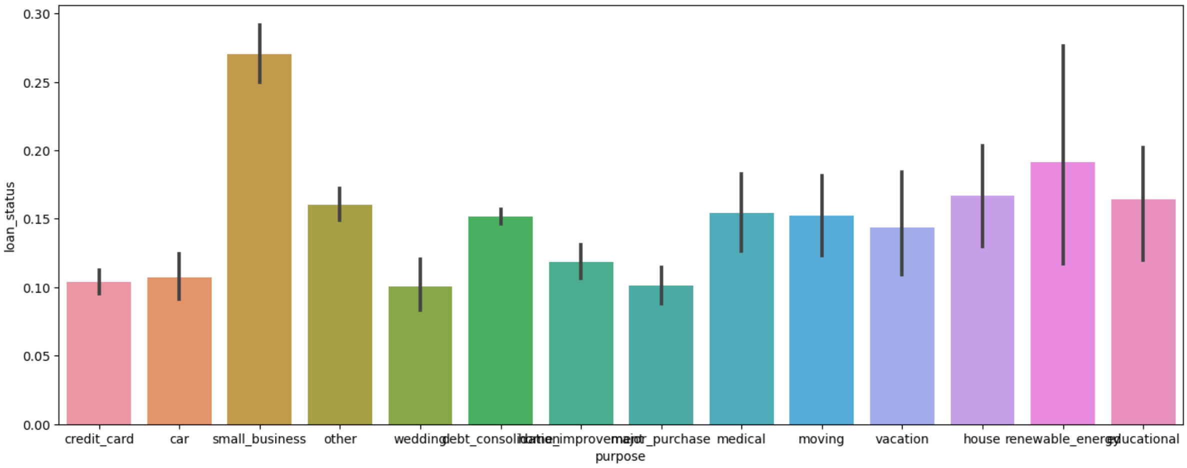


Segmented Univariate Analysis

We have now compared the default rates across various variables, and some of the important predictors are purpose of the loan, interest rate, annual income, grade etc.

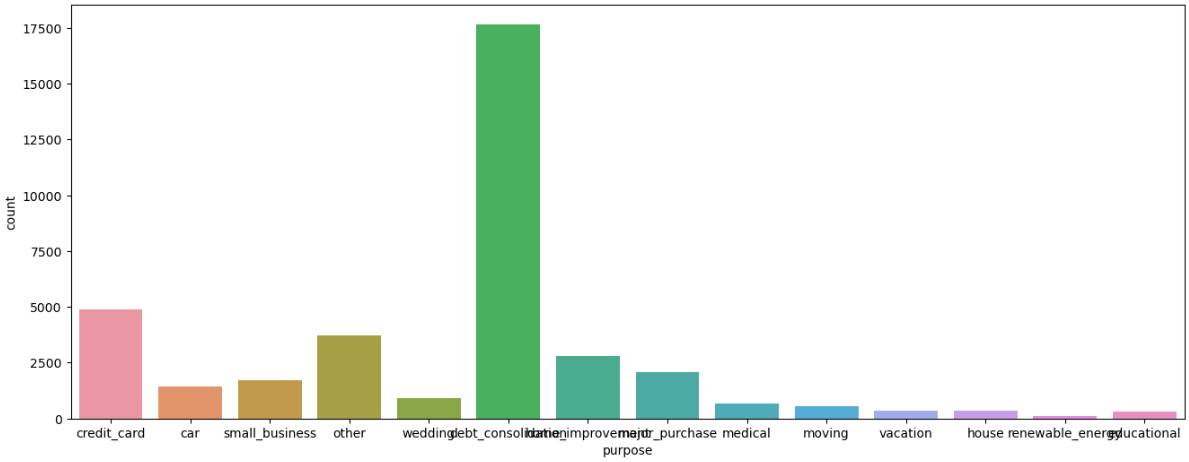
In the credit industry, one of the most important factors affecting default is the purpose of the loan - home loans perform differently than credit cards, credit cards are very different from debt condolidation loans etc. This comes from business understanding, though let's again have a look at the default rates across the purpose of the loan.

In [60]:
```
plt.figure(figsize=(16, 6))
plot_cat('purpose')
```



In the upcoming analyses, we will segment the loan applications across the purpose of the loan, since that is a variable affecting many other variables - the type of applicant, interest rate, income, and finally the default rate.

In [61]:
```
plt.figure(figsize=(16, 6))
sns.countplot(x='purpose', data=df)
plt.show()
```

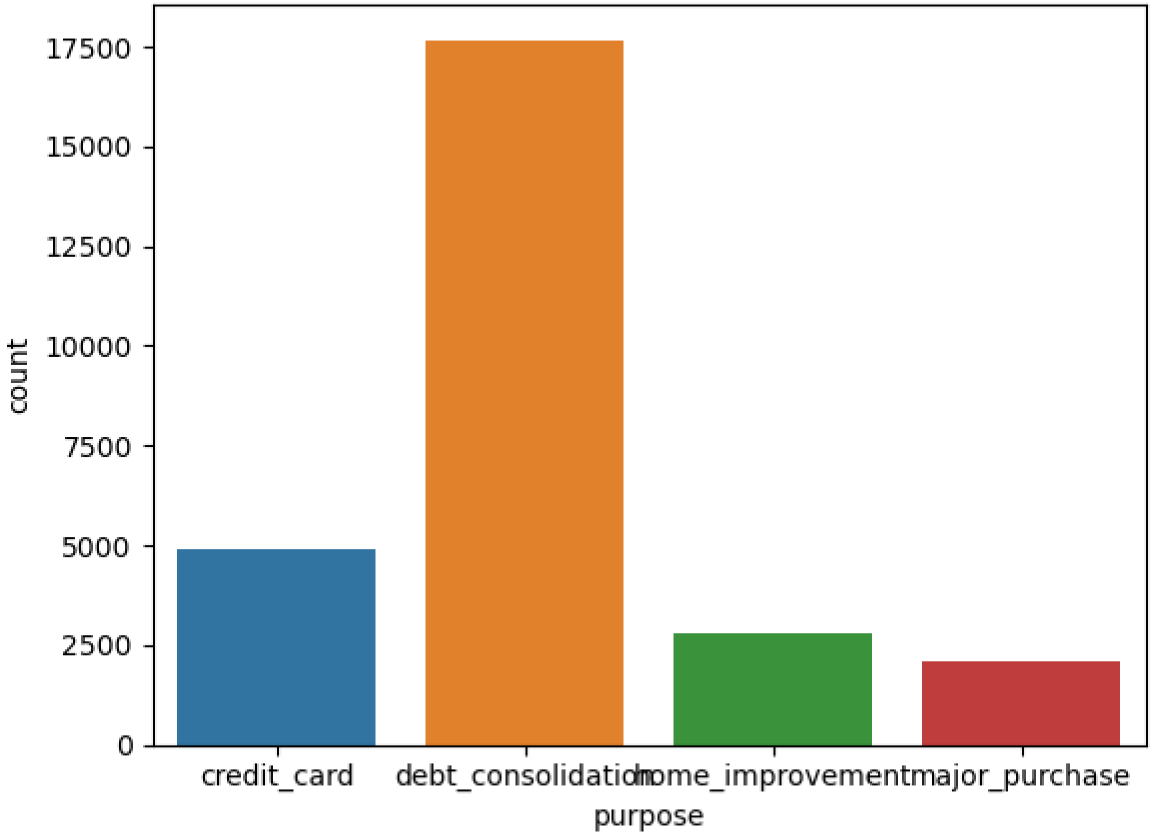


Let's analyse the top 4 types of loans based on purpose: consolidation, credit card, home improvement and major purchase.

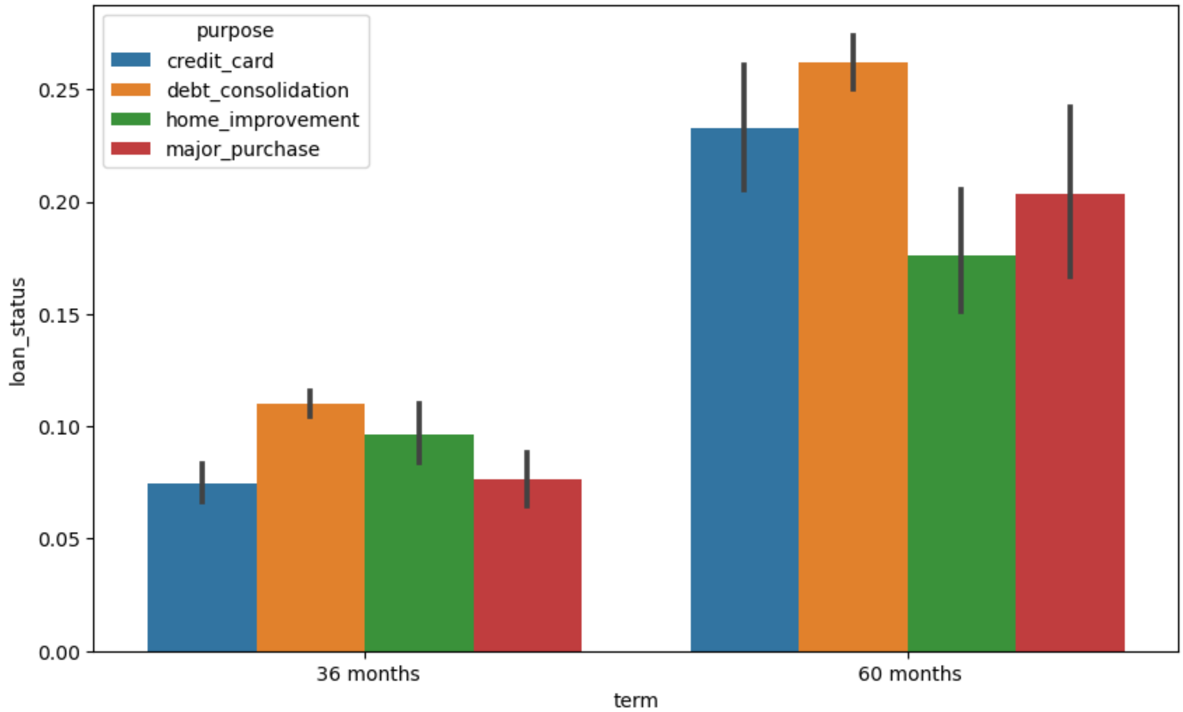
In [62]:
```
main_purposes = ["credit_card","debt_consolidation","home_improvement","majc
df = df[df['purpose'].isin(main_purposes)]
df['purpose'].value_counts()
```

Out[62]:
```
debt_consolidation    17675
credit_card            4899
home_improvement       2785
major_purchase         2080
Name: purpose, dtype: int64
```

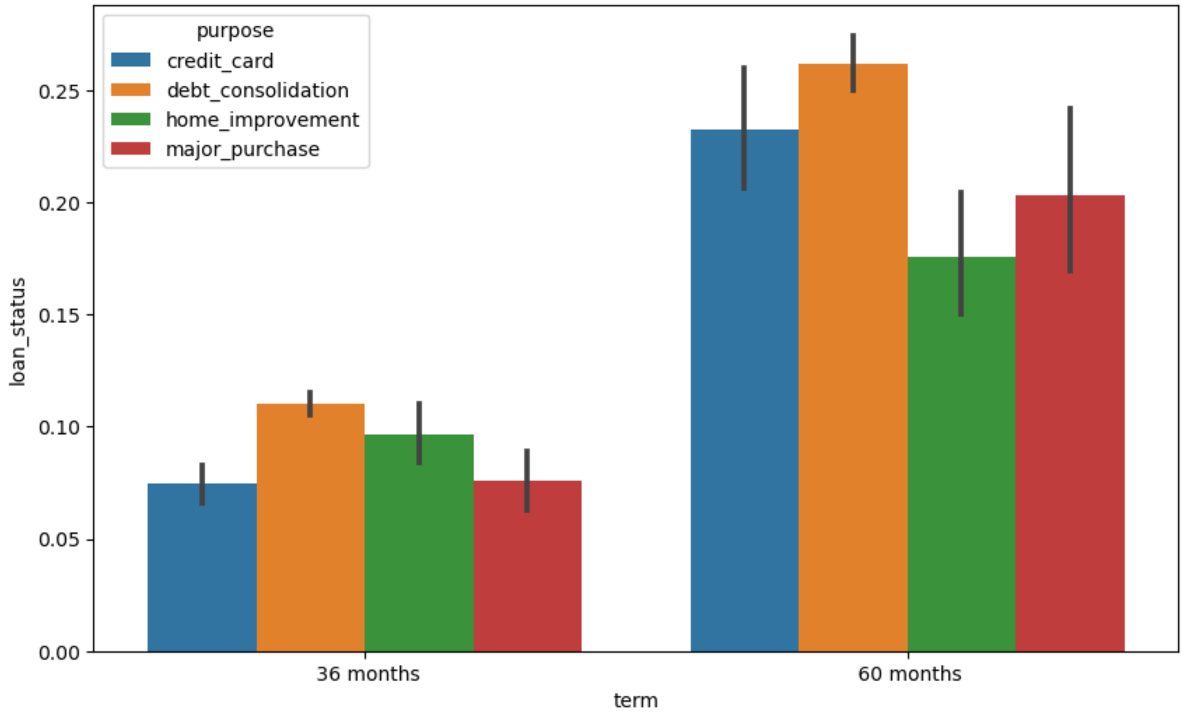
In [63]:
```
sns.countplot(x=df['purpose'])
plt.show()
```



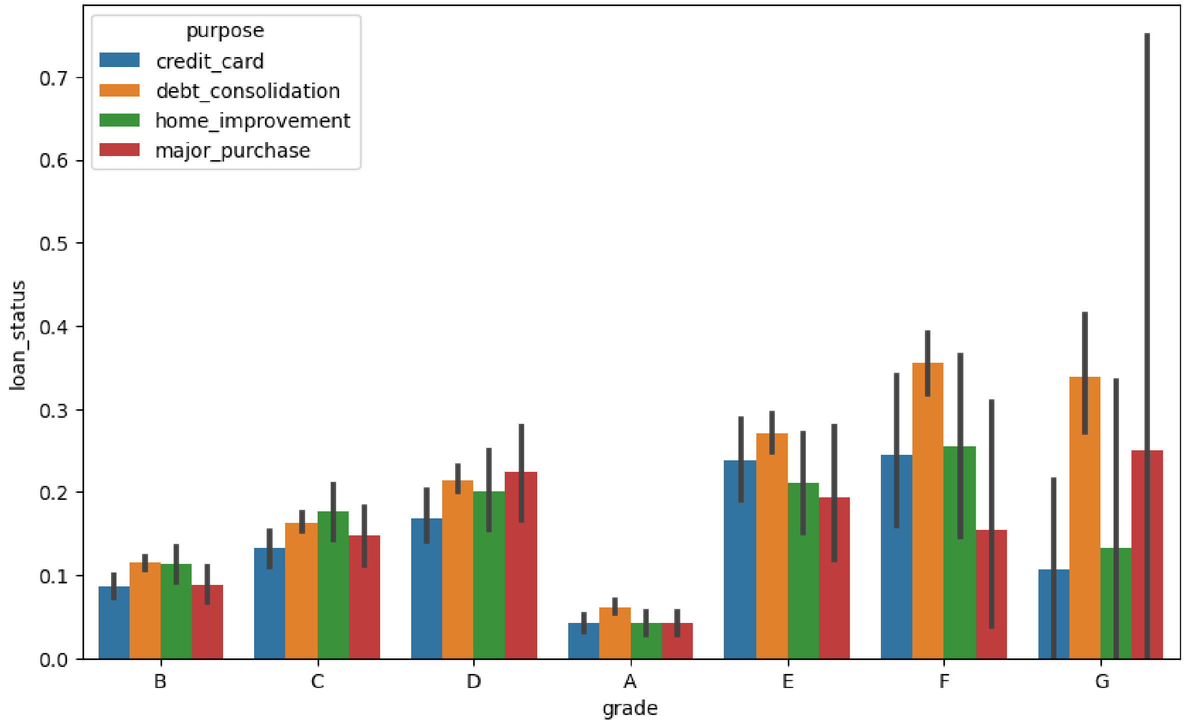
In [64]:
```
plt.figure(figsize=[10, 6])
sns.barplot(x='term', y="loan_status", hue='purpose', data=df)
plt.show()
```



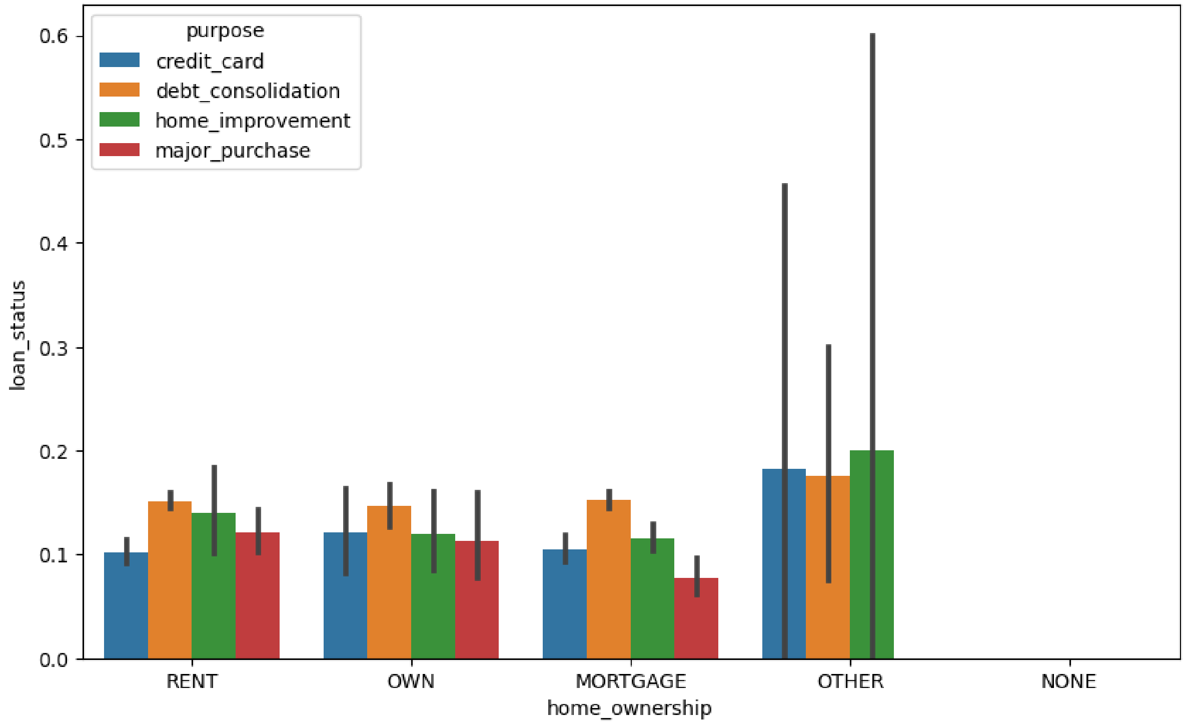
In [65]:
```
def plot_segmented(cat_var):
    plt.figure(figsize=(10, 6))
    sns.barplot(x=cat_var, y='loan_status', hue='purpose', data=df)
    plt.show()


plot_segmented('term')
```



In [66]:
```
plot_segmented('grade')
```

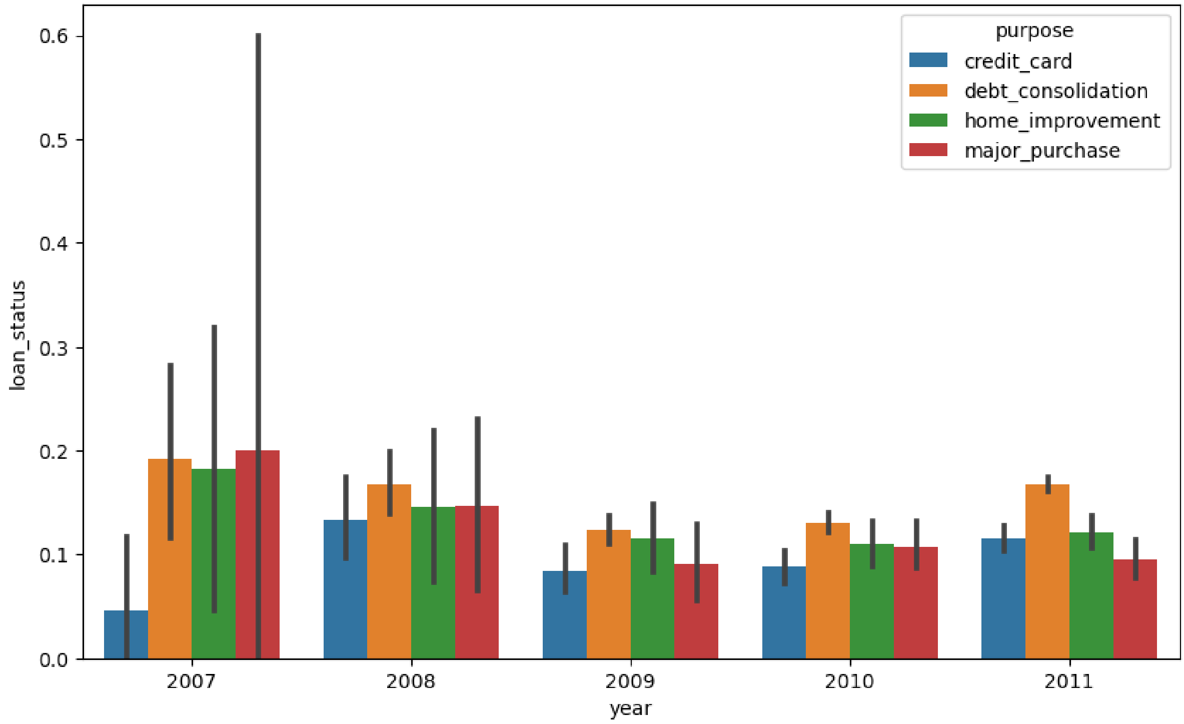


In [67]:
```
plot_segmented('home_ownership')
```

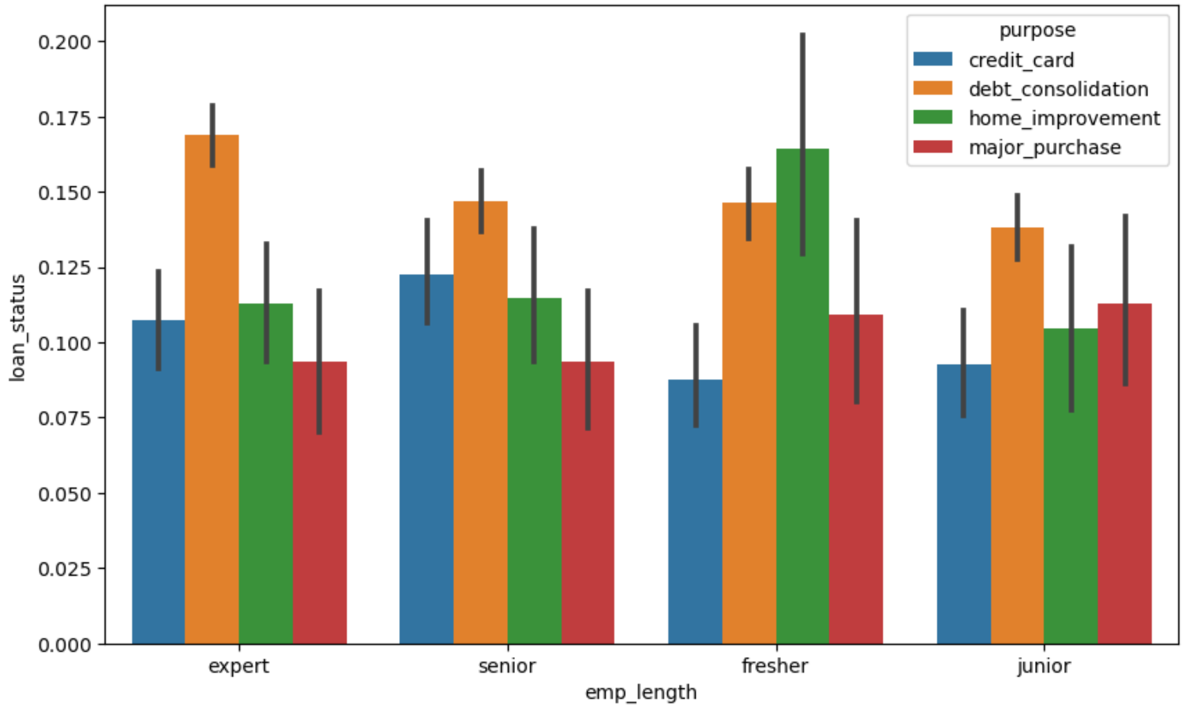


In general, debt consolidation loans have the highest default rates. Lets compare across other categories as well.

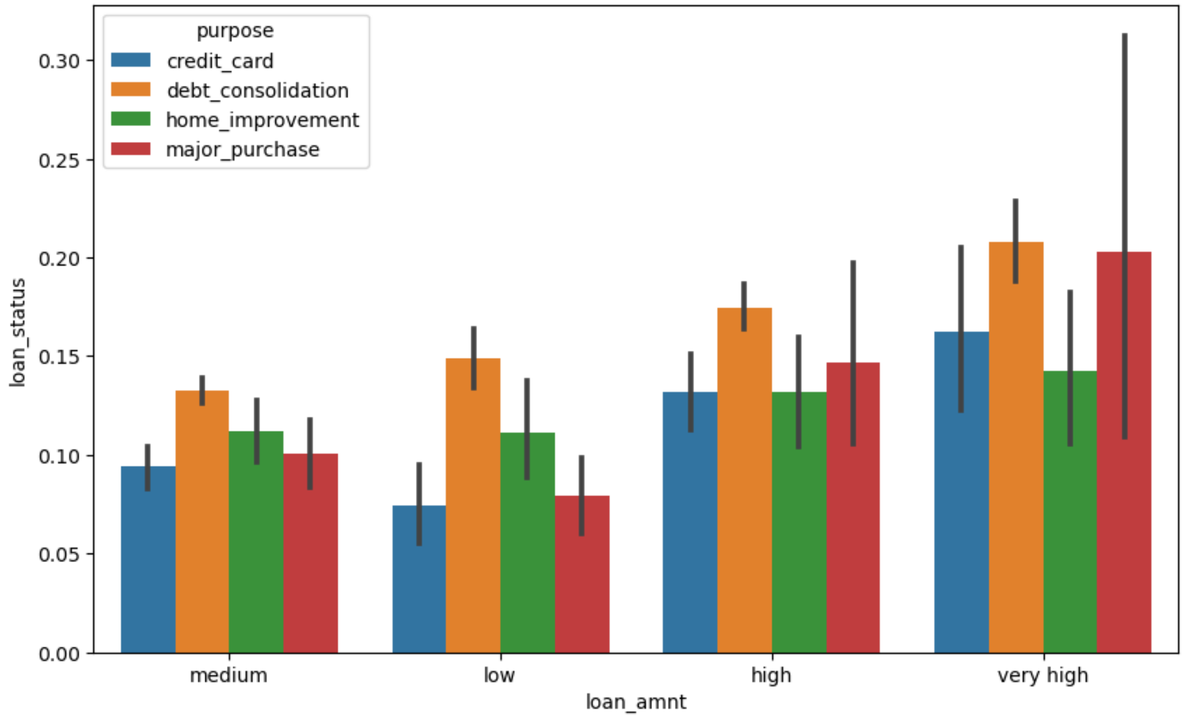
In [68]:
```
plot_segmented('year')
```



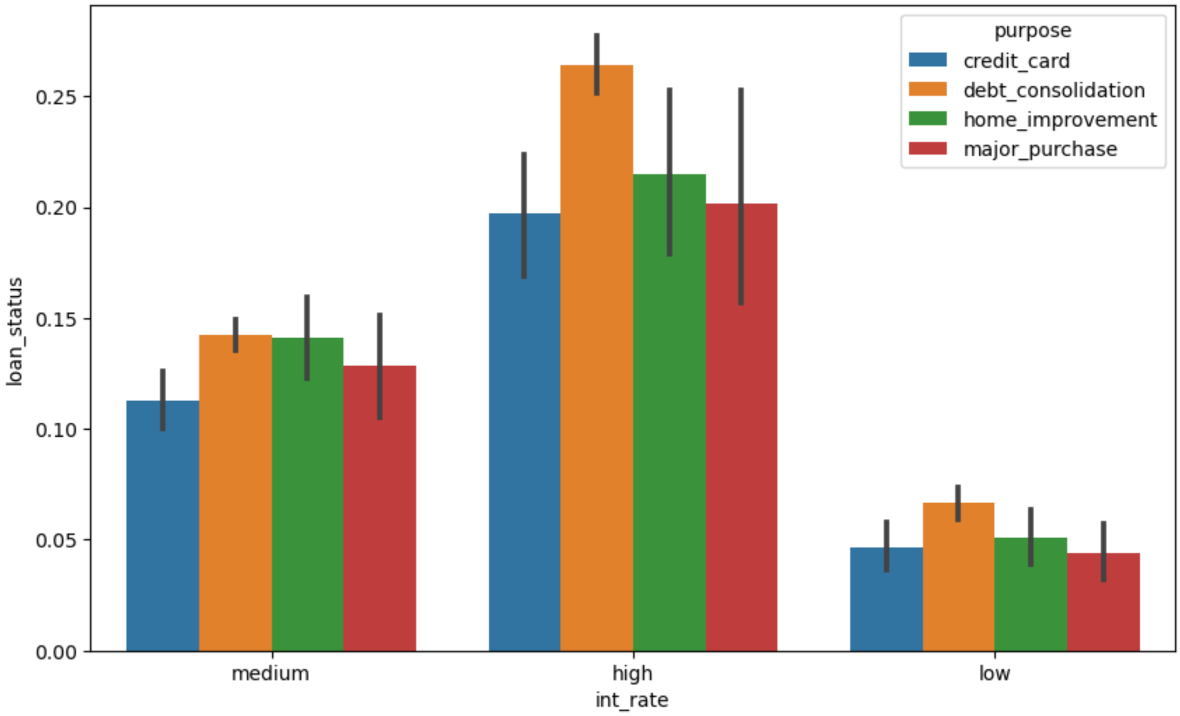
In [69]:
```
plot_segmented('emp_length')
```



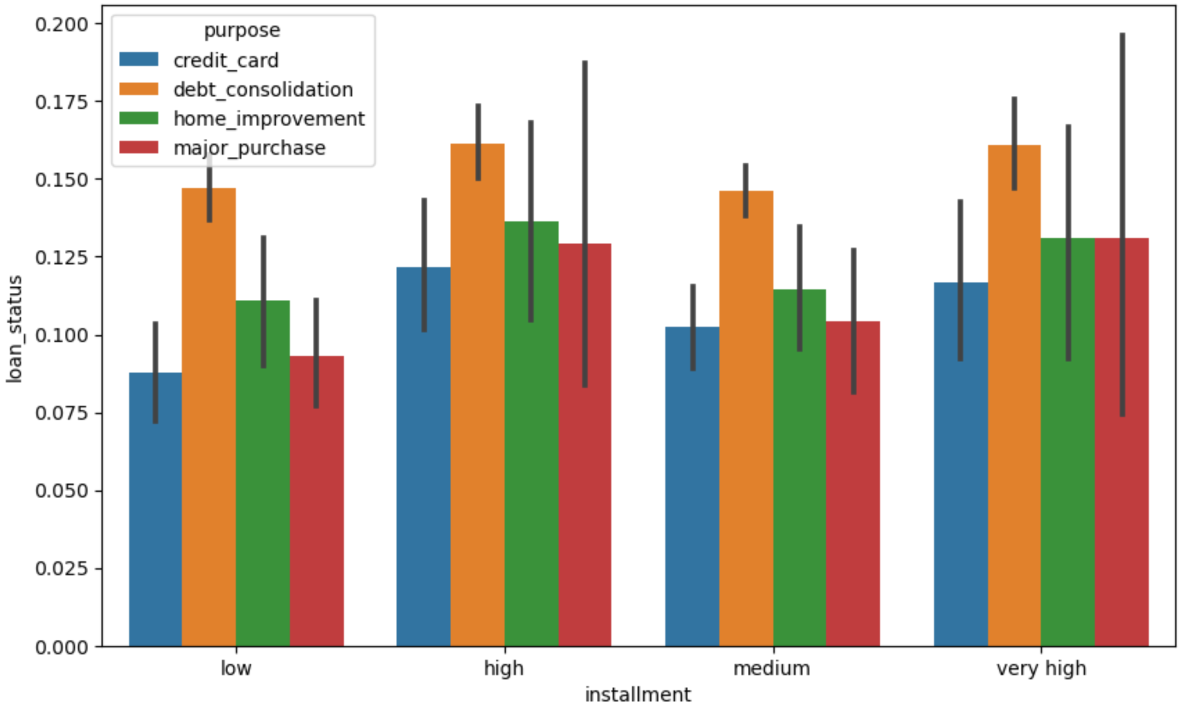
In [70]:
```
plot_segmented('loan_amnt')
```



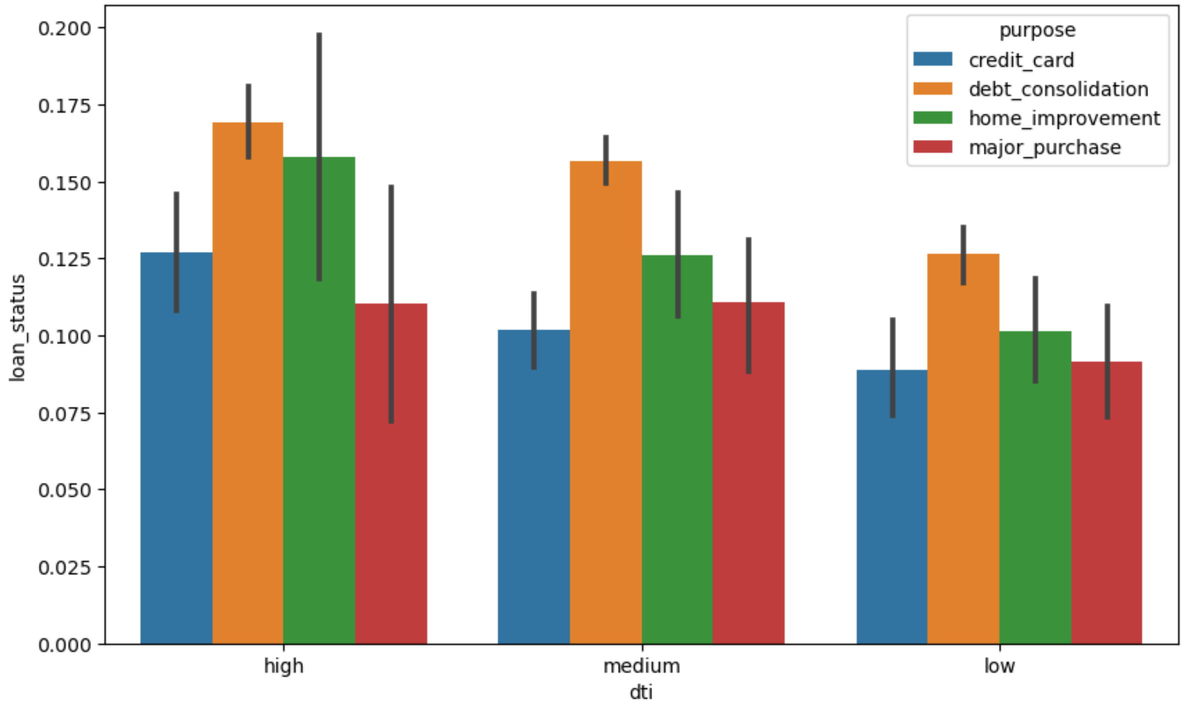
In [71]:
```
plot_segmented('int_rate')
```



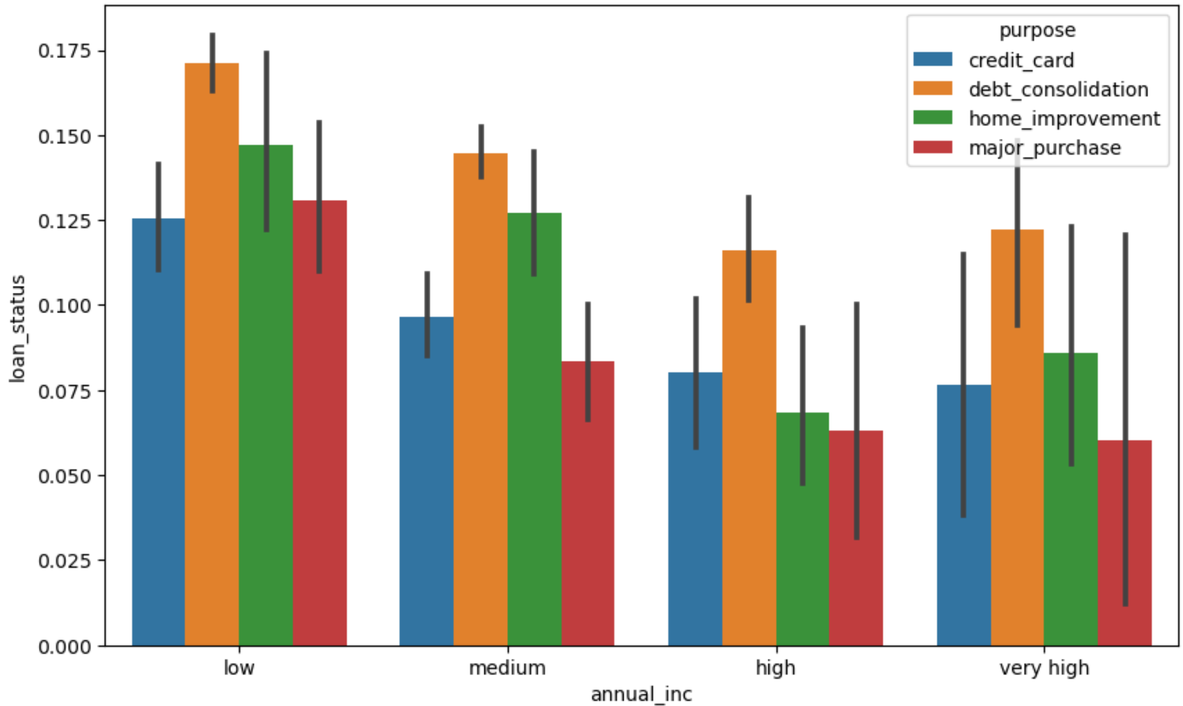
In [72]:
```
plot_segmented('installment')
```



In [73]:
```
plot_segmented('dti')
```



In [74]:
```
plot_segmented('annual_inc')
```



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'.

In [75]:
```
df.groupby('annual_inc').loan_status.mean().sort_values(ascending=False)
```

Out[75]:
```
annual_inc
low          0.157966
medium       0.130075
very high    0.101570
high         0.097749
Name: loan_status, dtype: float64
```

In [76]:
```
def diff_rate(cat_var):
    default_rates = df.groupby(cat_var).loan_status.mean().sort_values(ascen
    return (round(default_rates, 2), round(default_rates[0] - default_rates[
```

```
default_rates, diff = diff_rate('annual_inc')
print(default_rates)
print(diff)
```

```
annual_inc
low          0.16
medium       0.13
very high    0.10
high         0.10
Name: loan_status, dtype: float64
0.06
```

In [77]:
```
df_categorical = df.loc[:, df.dtypes == object]
df_categorical['loan_status'] = df['loan_status']
print([i for i in df.columns])
```

```
['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_stat
us', 'pymnt_plan', 'purpose', 'dti', 'initial_list_status', 'collections_12_
mths_ex_med', 'policy_code', 'acc_now_delinq', 'chargeoff_within_12_mths',
'delinq_amnt', 'pub_rec_bankruptcies', 'tax_liens', 'month', 'year']
```

In [78]:
```
d = {key: diff_rate(key)[1]*100 for key in df_categorical.columns if key !=
print(d)
```

```
{'loan_amnt': 7.000000000000001, 'funded_amnt': 5.0, 'funded_amnt_inv': 6.0,
'term': 15.0, 'int_rate': 19.0, 'installment': 3.0, 'grade': 27.0, 'sub_grad
e': 46.0, 'emp_title': 100.0, 'emp_length': 2.0, 'home_ownership': 16.0, 'an
nual_inc': 6.0, 'verification_status': 4.0, 'pymnt_plan': 0.0, 'purpose': 5.
0, 'dti': 5.0, 'initial_list_status': 0.0}
```

In [ ]: