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 [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

loan = pd.read_csv("loan.csv", sep=",")
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Columns: 111 entries, id to total_il_high_credit_limit
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB
```

```
/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/IPython/co
re/interactiveshell.py:2728: DtypeWarning: Columns (47) have mixed types. Specify dtype
option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Data Understanding

In [ ]:

```
# let's look at the first few rows of the df
loan.head()
```

Out[ ]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grad
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	162.87	I
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	59.83	(
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	84.33	(
3	1076863	1277178	10000	10000	10000.0	36 months	13.49%	339.31	(
4	1075358	1311748	3000	3000	3000.0	60 months	12.69%	67.79	I

5 rows × 111 columns

In [ ]:
```
# Looking at all the column names
loan.columns
```

Out[ ]:
```
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 [ ]:
```
# summarising number of missing values in each column
loan.isnull().sum()
```

Out[ ]:
```
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_title                      2459
emp_length                     1075
home_ownership                    0
annual_inc                        0
verification_status               0
issue_d                           0
loan_status                       0
pymnt_plan                        0
url                               0
desc                          12940
purpose                           0
title                            11
zip_code                          0
addr_state                        0
dti                               0
delinq_2yrs                       0
earliest_cr_line                  0
```

```
inq_last_6mths                        0
mths_since_last_delinq            25682
mths_since_last_record            36931
                                  ...
mo_sin_old_rev_tl_op              39717
mo_sin_rcnt_rev_tl_op             39717
mo_sin_rcnt_tl                    39717
mort_acc                          39717
mths_since_recent_bc              39717
mths_since_recent_bc_dlq          39717
mths_since_recent_inq             39717
mths_since_recent_revol_delinq    39717
num_accts_ever_120_pd             39717
num_actv_bc_tl                    39717
num_actv_rev_tl                   39717
num_bc_sats                       39717
num_bc_tl                         39717
num_il_tl                         39717
num_op_rev_tl                     39717
num_rev_accts                     39717
num_rev_tl_bal_gt_0               39717
num_sats                          39717
num_tl_120dpd_2m                  39717
num_tl_30dpd                      39717
num_tl_90g_dpd_24m                39717
num_tl_op_past_12m                39717
pct_tl_nvr_dlq                    39717
percent_bc_gt_75                  39717
pub_rec_bankruptcies                697
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 [ ]:

```
# percentage of missing values in each column
round(loan.isnull().sum()/len(loan.index), 2)*100
```

Out[ ]:

```
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_title                           6.0
emp_length                          3.0
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
desc                               33.0
```

```
purpose                          0.0
title                            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
mths_since_last_delinq          65.0
mths_since_last_record          93.0
                                ...
mo_sin_old_rev_tl_op           100.0
mo_sin_rcnt_rev_tl_op          100.0
mo_sin_rcnt_tl                 100.0
mort_acc                       100.0
mths_since_recent_bc           100.0
mths_since_recent_bc_dlq       100.0
mths_since_recent_inq          100.0
mths_since_recent_revol_delinq 100.0
num_accts_ever_120_pd          100.0
num_actv_bc_tl                 100.0
num_actv_rev_tl                100.0
num_bc_sats                    100.0
num_bc_tl                      100.0
num_il_tl                      100.0
num_op_rev_tl                  100.0
num_rev_accts                  100.0
num_rev_tl_bal_gt_0            100.0
num_sats                       100.0
num_tl_120dpd_2m               100.0
num_tl_30dpd                   100.0
num_tl_90g_dpd_24m             100.0
num_tl_op_past_12m             100.0
pct_tl_nvr_dlq                 100.0
percent_bc_gt_75               100.0
pub_rec_bankruptcies             2.0
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 [ ]:

```python
# removing the columns having more than 90% missing values
missing_columns = loan.columns[100*(loan.isnull().sum()/len(loan.index)) > 90]
print(missing_columns)
```

```
Index(['mths_since_last_record', 'next_pymnt_d', 'mths_since_last_major_derog',
       '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_op',
       '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 [ ]:

```
loan = loan.drop(missing_columns, axis=1)
print(loan.shape)
```

```
(39717, 55)
```

In [ ]:

```
# summarise number of missing values again
100*(loan.isnull().sum()/len(loan.index))
```

Out[ ]:

```
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 [ ]:

```
# There are now 2 columns having approx 32 and 64% missing values -
# description and months since last delinquent

# let's have a look at a few entries in the columns
loan.loc[:, ['desc', 'mths_since_last_delinq']].head()
```

Out[ ]:

	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 [ ]:

```
# dropping the two columns
loan = loan.drop(['desc', 'mths_since_last_delinq'], axis=1)
```

In [ ]:

```
# summarise number of missing values again
100*(loan.isnull().sum()/len(loan.index))
```

Out[ ]:

```
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 ntot 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 [ ]:

```python
# missing values in rows
loan.isnull().sum(axis=1)
```

Out[ ]:

```
0        1
1        0
2        1
3        0
4        0
5        0
6        0
7        0
8        1
9        0
10       0
11       0
12       0
13       0
14       0
15       0
16       0
17       0
18       0
19       0
20       0
21       0
22       0
23       0
24       0
25       0
26       1
27       0
28       0
29       0
        ..
39687    4
39688    4
39689    4
39690    4
39691    4
39692    4
39693    4
39694    4
39695    4
39696    4
39697    4
39698    4
39699    4
39700    5
39701    4
39702    4
39703    4
39704    5
39705    4
39706    5
39707    4
39708    4
39709    4
39710    4
```

```
39711    4
39712    4
39713    4
39714    5
39715    5
39716    4
Length: 39717, dtype: int64
```

In [ ]:

```
# checking whether some rows have more than 5 missing values
len(loan[loan.isnull().sum(axis=1) > 5].index)
```

Out[ ]:

```
0
```

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

In [ ]:

```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 53 columns):
id                            39717 non-null int64
member_id                     39717 non-null int64
loan_amnt                     39717 non-null int64
funded_amnt                   39717 non-null int64
funded_amnt_inv               39717 non-null float64
term                          39717 non-null object
int_rate                      39717 non-null object
installment                   39717 non-null float64
grade                         39717 non-null object
sub_grade                     39717 non-null object
emp_title                     37258 non-null object
emp_length                    38642 non-null object
home_ownership                39717 non-null object
annual_inc                    39717 non-null float64
verification_status           39717 non-null object
issue_d                       39717 non-null object
loan_status                   39717 non-null object
pymnt_plan                    39717 non-null object
url                           39717 non-null object
purpose                       39717 non-null object
title                         39706 non-null object
zip_code                      39717 non-null object
addr_state                    39717 non-null object
dti                           39717 non-null float64
delinq_2yrs                   39717 non-null int64
earliest_cr_line              39717 non-null object
inq_last_6mths                39717 non-null int64
open_acc                      39717 non-null int64
pub_rec                       39717 non-null int64
revol_bal                     39717 non-null int64
revol_util                    39667 non-null object
total_acc                     39717 non-null int64
initial_list_status           39717 non-null object
out_prncp                     39717 non-null float64
out_prncp_inv                 39717 non-null float64
total_pymnt                   39717 non-null float64
total_pymnt_inv               39717 non-null float64
total_rec_prncp               39717 non-null float64
```

```
total_rec_int                 39717 non-null float64
total_rec_late_fee            39717 non-null float64
recoveries                    39717 non-null float64
collection_recovery_fee       39717 non-null float64
last_pymnt_d                  39646 non-null object
last_pymnt_amnt               39717 non-null float64
last_credit_pull_d            39715 non-null object
collections_12_mths_ex_med    39661 non-null float64
policy_code                   39717 non-null int64
application_type              39717 non-null object
acc_now_delinq                39717 non-null int64
chargeoff_within_12_mths      39661 non-null float64
delinq_amnt                   39717 non-null int64
pub_rec_bankruptcies          39020 non-null float64
tax_liens                     39678 non-null float64
dtypes: float64(18), int64(13), object(22)
memory usage: 16.1+ MB
```

In [ ]:

```
# The column int_rate is character type, let's convert it to float
loan['int_rate'] = loan['int_rate'].apply(lambda x: pd.to_numeric(x.split("%")[0]))
```

In [ ]:

```
# checking the data types
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 53 columns):
id                            39717 non-null int64
member_id                     39717 non-null int64
loan_amnt                     39717 non-null int64
funded_amnt                   39717 non-null int64
funded_amnt_inv               39717 non-null float64
term                          39717 non-null object
int_rate                      39717 non-null float64
installment                   39717 non-null float64
grade                         39717 non-null object
sub_grade                     39717 non-null object
emp_title                     37258 non-null object
emp_length                    38642 non-null object
home_ownership                39717 non-null object
annual_inc                    39717 non-null float64
verification_status           39717 non-null object
issue_d                       39717 non-null object
loan_status                   39717 non-null object
pymnt_plan                    39717 non-null object
url                           39717 non-null object
purpose                       39717 non-null object
title                         39706 non-null object
zip_code                      39717 non-null object
addr_state                    39717 non-null object
dti                           39717 non-null float64
delinq_2yrs                   39717 non-null int64
earliest_cr_line              39717 non-null object
inq_last_6mths                39717 non-null int64
open_acc                      39717 non-null int64
pub_rec                       39717 non-null int64
revol_bal                     39717 non-null int64
revol_util                    39667 non-null object
```

```
total_acc                     39717 non-null int64
initial_list_status           39717 non-null object
out_prncp                     39717 non-null float64
out_prncp_inv                 39717 non-null float64
total_pymnt                   39717 non-null float64
total_pymnt_inv               39717 non-null float64
total_rec_prncp               39717 non-null float64
total_rec_int                 39717 non-null float64
total_rec_late_fee            39717 non-null float64
recoveries                    39717 non-null float64
collection_recovery_fee       39717 non-null float64
last_pymnt_d                  39646 non-null object
last_pymnt_amnt               39717 non-null float64
last_credit_pull_d            39715 non-null object
collections_12_mths_ex_med    39661 non-null float64
policy_code                   39717 non-null int64
application_type              39717 non-null object
acc_now_delinq                39717 non-null int64
chargeoff_within_12_mths      39661 non-null float64
delinq_amnt                   39717 non-null int64
pub_rec_bankruptcies          39020 non-null float64
tax_liens                     39678 non-null float64
dtypes: float64(19), int64(13), object(21)
memory usage: 16.1+ MB
```

In [ ]:

```
# also, lets extract the numeric part from the variable employment length

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

# using regular expression to extract numeric values from the string
import re
loan['emp_length'] = loan['emp_length'].apply(lambda x: re.findall('\d+', str(x))[0])

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

In [ ]:

```
# looking at type of the columns again
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38642 entries, 0 to 39716
Data columns (total 53 columns):
id                            38642 non-null int64
member_id                     38642 non-null int64
loan_amnt                     38642 non-null int64
funded_amnt                   38642 non-null int64
funded_amnt_inv               38642 non-null float64
term                          38642 non-null object
int_rate                      38642 non-null float64
installment                   38642 non-null float64
grade                         38642 non-null object
sub_grade                     38642 non-null object
emp_title                     37202 non-null object
emp_length                    38642 non-null int64
home_ownership                38642 non-null object
annual_inc                    38642 non-null float64
verification_status           38642 non-null object
```

```
 issue_d                     38642 non-null object
 loan_status                 38642 non-null object
 pymnt_plan                  38642 non-null object
 url                         38642 non-null object
 purpose                     38642 non-null object
 title                       38632 non-null object
 zip_code                    38642 non-null object
 addr_state                  38642 non-null object
 dti                         38642 non-null float64
 delinq_2yrs                 38642 non-null int64
 earliest_cr_line            38642 non-null object
 inq_last_6mths              38642 non-null int64
 open_acc                    38642 non-null int64
 pub_rec                     38642 non-null int64
 revol_bal                   38642 non-null int64
 revol_util                  38595 non-null object
 total_acc                   38642 non-null int64
 initial_list_status         38642 non-null object
 out_prncp                   38642 non-null float64
 out_prncp_inv               38642 non-null float64
 total_pymnt                 38642 non-null float64
 total_pymnt_inv             38642 non-null float64
 total_rec_prncp             38642 non-null float64
 total_rec_int               38642 non-null float64
 total_rec_late_fee          38642 non-null float64
 recoveries                  38642 non-null float64
 collection_recovery_fee     38642 non-null float64
 last_pymnt_d                38576 non-null object
 last_pymnt_amnt             38642 non-null float64
 last_credit_pull_d          38640 non-null object
 collections_12_mths_ex_med  38586 non-null float64
 policy_code                 38642 non-null int64
 application_type            38642 non-null object
 acc_now_delinq              38642 non-null int64
 chargeoff_within_12_mths    38586 non-null float64
 delinq_amnt                 38642 non-null int64
 pub_rec_bankruptcies        37945 non-null float64
 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.),

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

2. 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 [ ]:

```
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[ ]:

```
['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 [ ]:

```
# let's now remove the behaviour variables from analysis
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):
id                            38642 non-null int64
member_id                     38642 non-null int64
loan_amnt                     38642 non-null int64
funded_amnt                   38642 non-null int64
funded_amnt_inv               38642 non-null float64
term                          38642 non-null object
int_rate                      38642 non-null float64
installment                   38642 non-null float64
grade                         38642 non-null object
sub_grade                     38642 non-null object
emp_title                     37202 non-null object
emp_length                    38642 non-null int64
home_ownership                38642 non-null object
annual_inc                    38642 non-null float64
verification_status           38642 non-null object
issue_d                       38642 non-null object
loan_status                   38642 non-null object
pymnt_plan                    38642 non-null object
url                           38642 non-null object
purpose                       38642 non-null object
title                         38632 non-null object
zip_code                      38642 non-null object
addr_state                    38642 non-null object
dti                           38642 non-null float64
initial_list_status           38642 non-null object
collections_12_mths_ex_med    38586 non-null float64
policy_code                   38642 non-null int64
acc_now_delinq                38642 non-null int64
chargeoff_within_12_mths      38586 non-null float64
delinq_amnt                   38642 non-null int64
pub_rec_bankruptcies          37945 non-null float64
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 [ ]:

```
# 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

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 [ ]:

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

Out[ ]:

```
Fully Paid     32145
Charged Off     5399
Current         1098
Name: loan_status, dtype: int64
```

You 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 [ ]:
```
# filtering only fully paid or charged-off
df = df[df['loan_status'] != 'Current']
df['loan_status'] = df['loan_status'].apply(lambda x: 0 if x=='Fully Paid' else 1)

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

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

Out[ ]:
```
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.

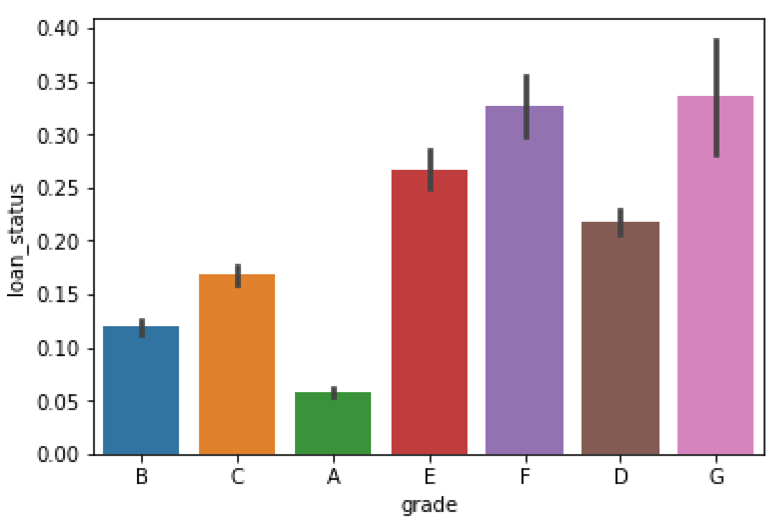
In [ ]:
```
# default rate
round(np.mean(df['loan_status']), 2)
```

Out[ ]:
```
0.14000000000000001
```

The overall default rate is about 14%.

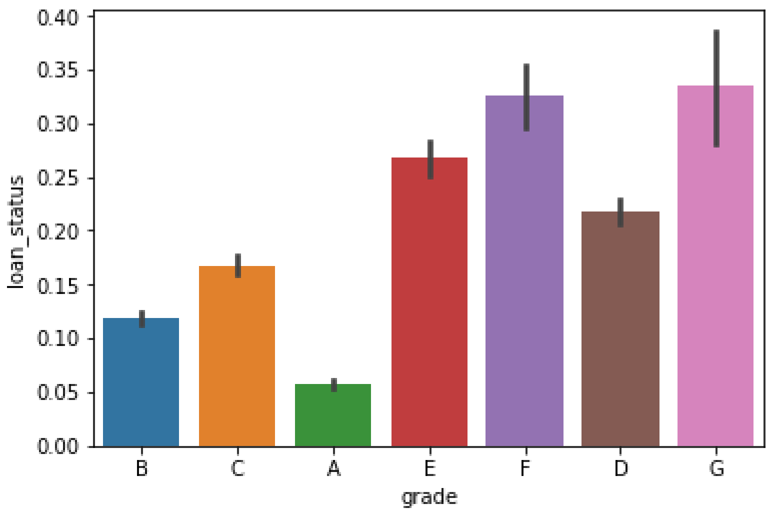
Let's first visualise the average default rates across categorical variables.

In [ ]:
```
# plotting default rates across grade of the loan
sns.barplot(x='grade', y='loan_status', data=df)
plt.show()
```



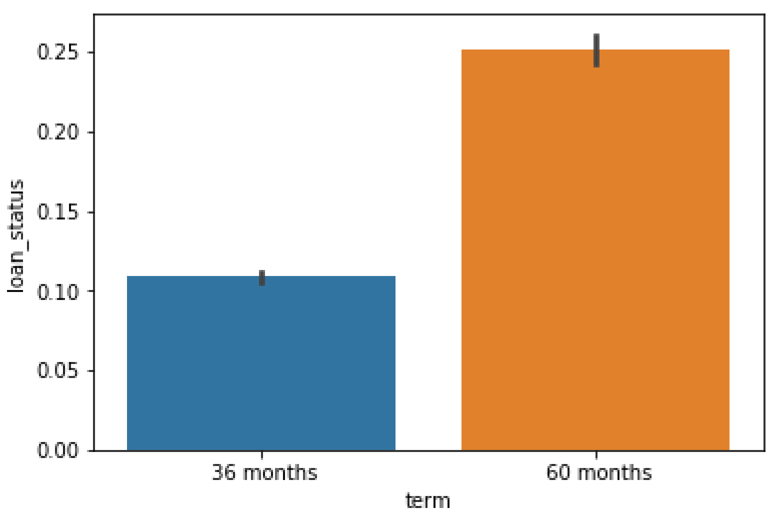
In [ ]:
```
# lets define a function to plot loan_status across categorical variables
def plot_cat(cat_var):
    sns.barplot(x=cat_var, y='loan_status', data=df)
    plt.show()
```

In [ ]:
```
# compare default rates across grade of loan
plot_cat('grade')
```

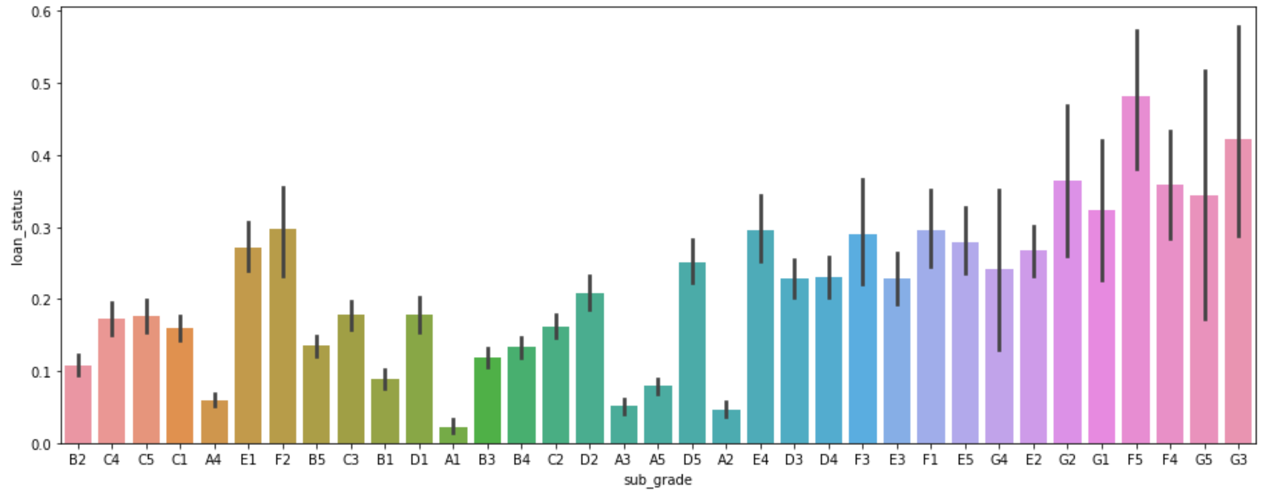


Clearly, as the grade of loan goes from A to G, the default rate increases. This is expected because the grade is decided by Lending Club based on the riskiness of the loan.

In [ ]:
```
# term: 60 months loans default more than 36 months loans
plot_cat('term')
```

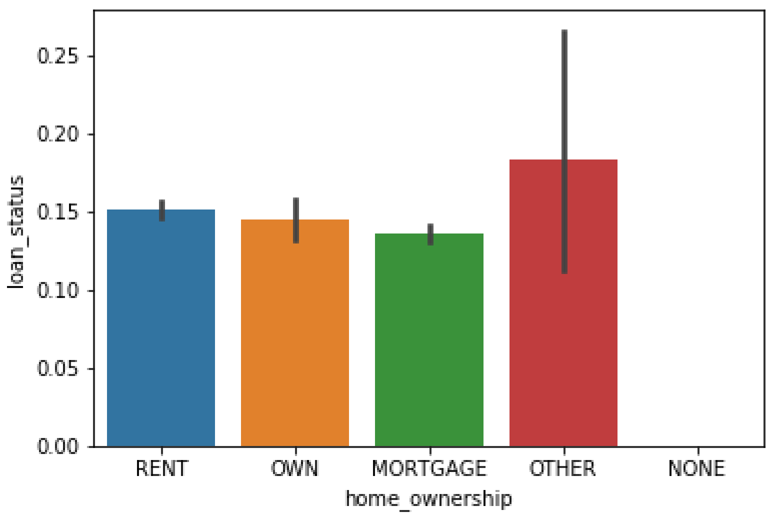


In [ ]:
```
# sub-grade: as expected - A1 is better than A2 better than A3 and so on
plt.figure(figsize=(16, 6))
plot_cat('sub_grade')
```



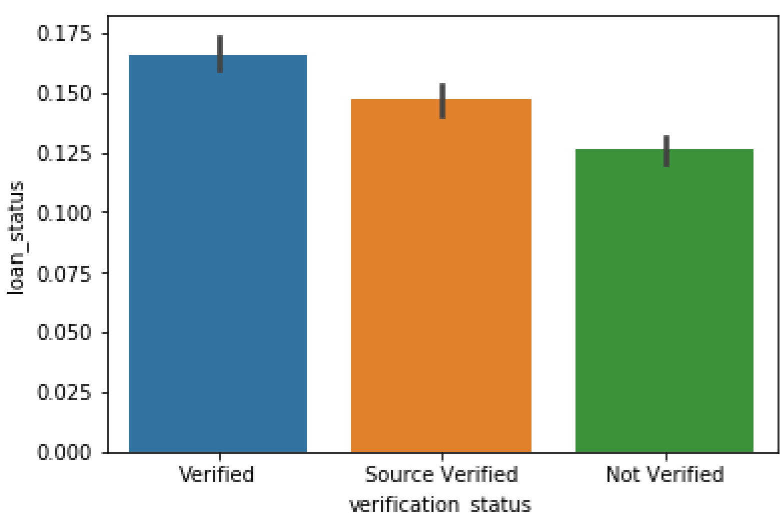
In [ ]:

```
# home ownership: not a great discriminator
plot_cat('home_ownership')
```

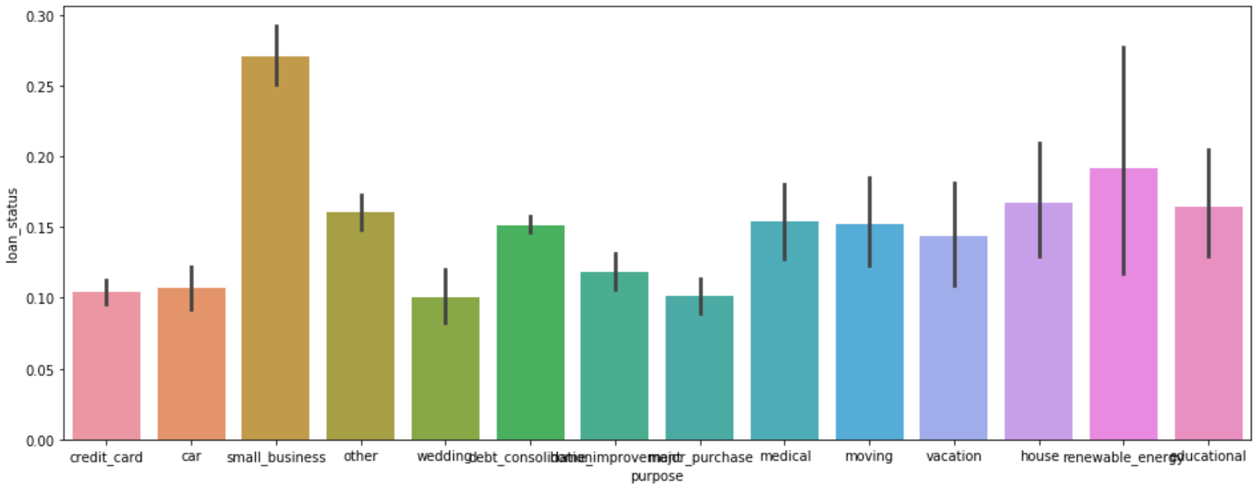


In [ ]:

```
# verification_status: surprisingly, verified loans default more than not verifiedb
plot_cat('verification_status')
```



In [ ]:
```
# purpose: small business loans defualt the most, then renewable energy and education
plt.figure(figsize=(16, 6))
plot_cat('purpose')
```



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

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

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

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

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

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

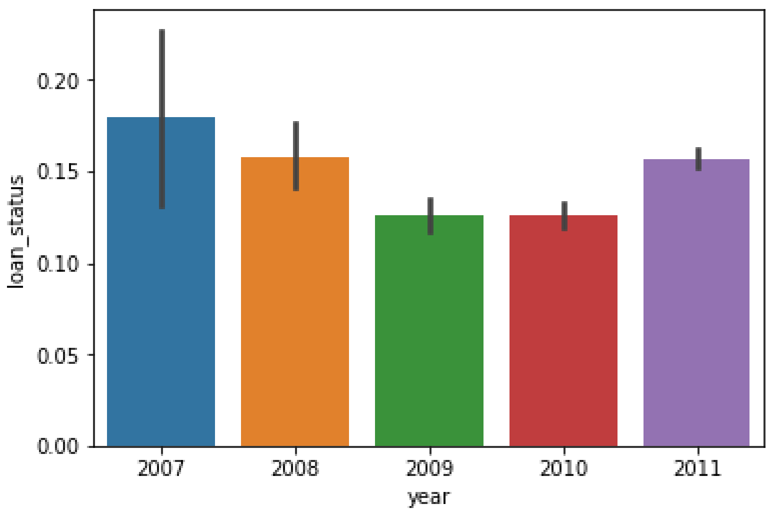
You can see that the number of loans has increased steadily across years.

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

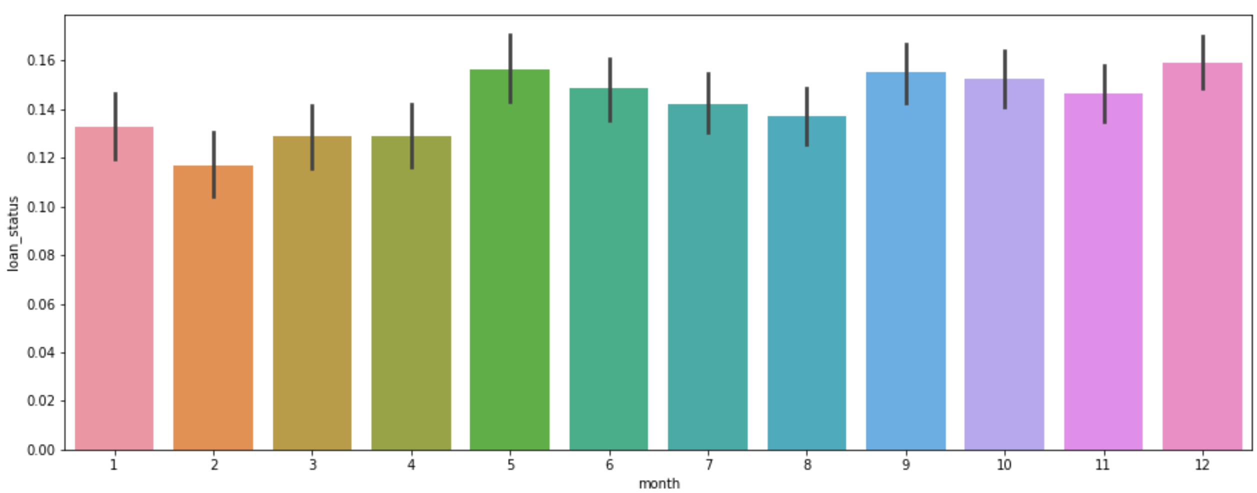
Out[ ]:
```
month
1     2331
2     2278
3     2632
4     2756
5     2838
6     3094
7     3253
8     3321
9     3394
10    3637
11    3890
12    4120
Name: month, dtype: int64
```

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

In [ ]:
```
# lets compare the default rates across years
# the default rate had suddenly increased in 2011, inspite of reducing from 2008 till 2
plot_cat('year')
```

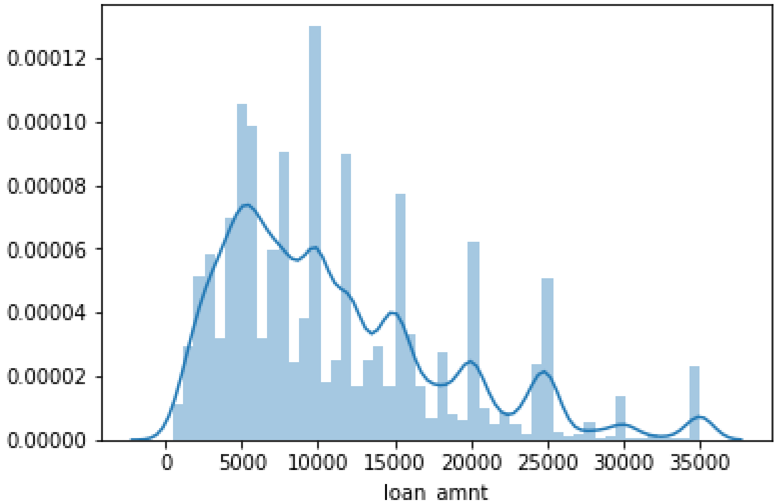


In [ ]:
```
# comparing default rates across months: not much variation across months
plt.figure(figsize=(16, 6))
plot_cat('month')
```



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

In [ ]:
```
# loan amount: the median loan amount is around 10,000
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.

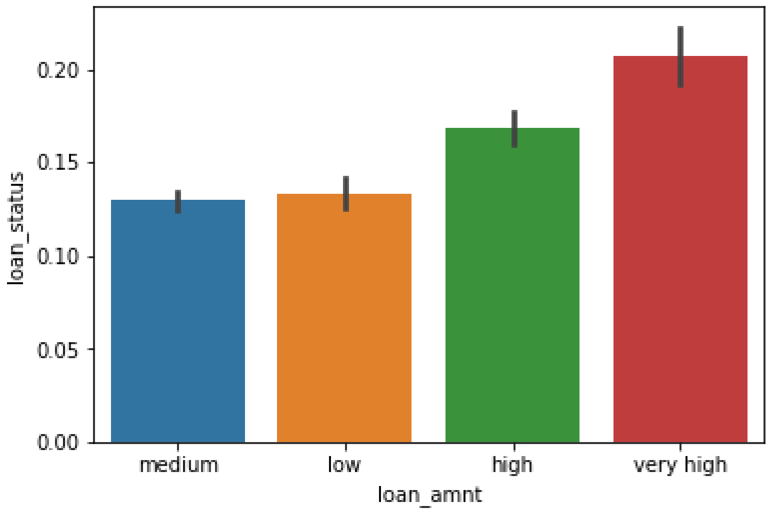
Let's bin the loan amount variable into small, medium, high, very high.

In [ ]:
```
# binning 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'

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

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

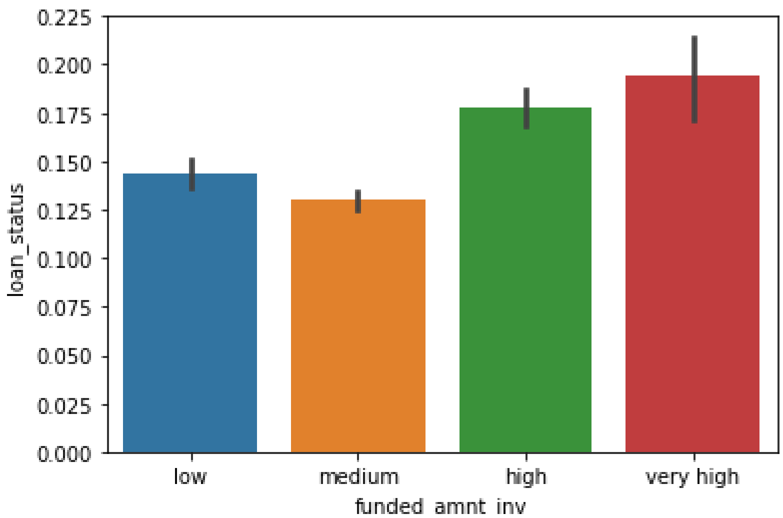
Out[ ]:
```
medium       20157
high          7572
low           7095
very high     2720
Name: loan_amnt, dtype: int64
```

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



In [ ]:
```
# let's also convert funded amount invested to bins
df['funded_amnt_inv'] = df['funded_amnt_inv'].apply(lambda x: loan_amount(x))
```

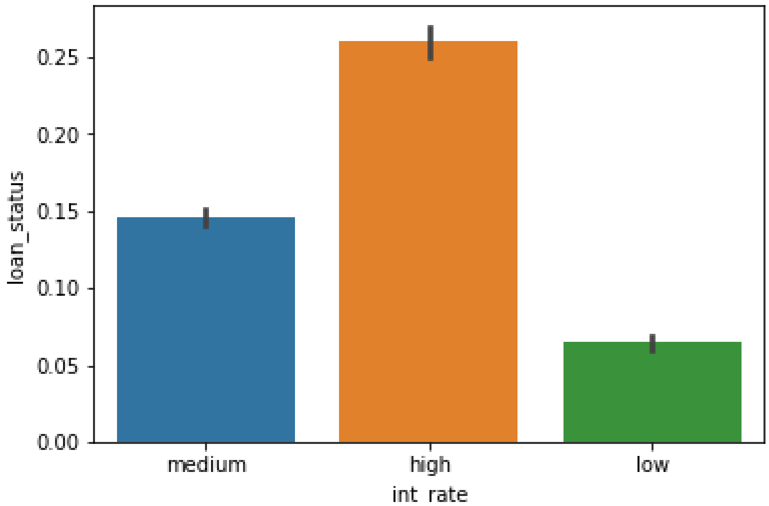
In [ ]:
```
# funded amount invested
plot_cat('funded_amnt_inv')
```



In [ ]:
```
# 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'


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

In [ ]:
```
# comparing default rates across rates of interest
# high interest rates default more, as expected
```

```
plot_cat('int_rate')
```

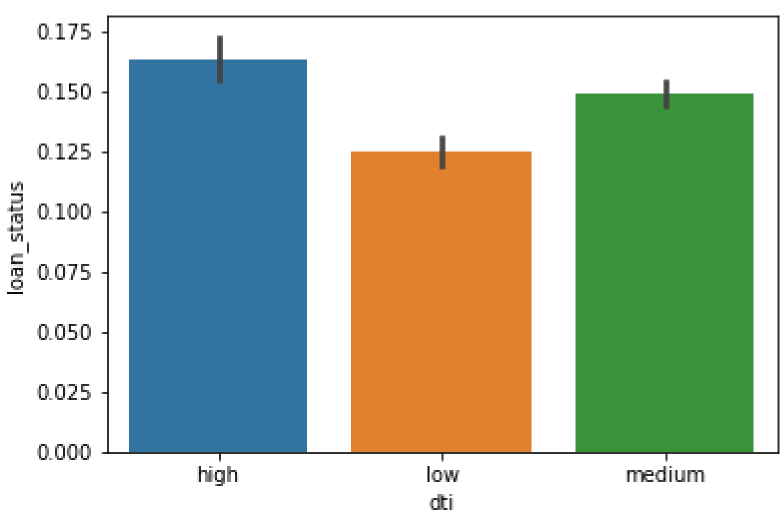


In [ ]:

```
# debt to income ratio
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 [ ]:

```
# comparing default rates across debt to income ratio
# high dti translates into higher default rates, as expected
plot_cat('dti')
```



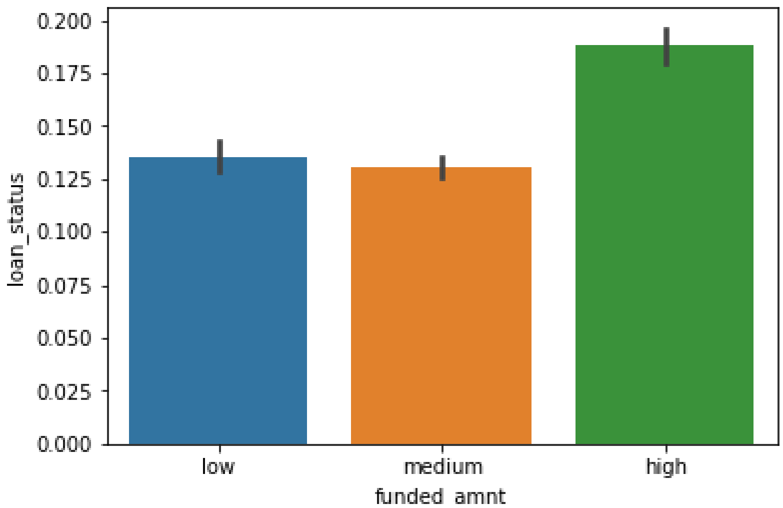
In [ ]:

```
# funded amount
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 [ ]:

```
plot_cat('funded_amnt')
```

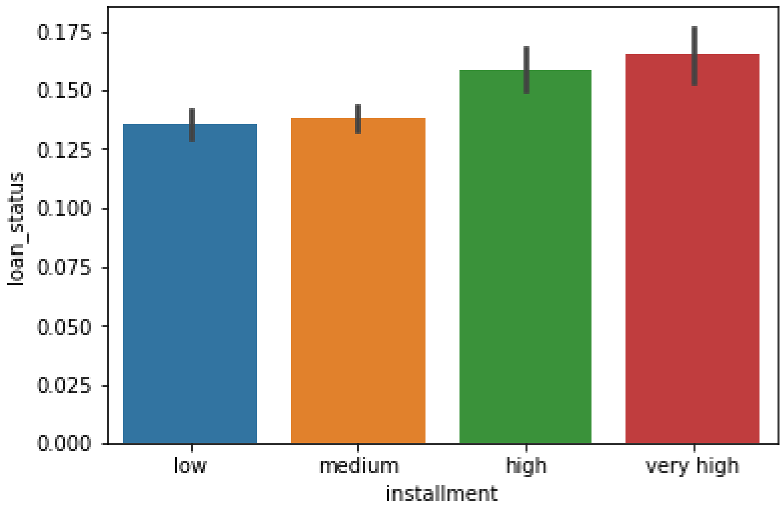


In [ ]:

```
# 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'

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

In [ ]:

```
# comparing default rates across installment
# the higher the installment amount, the higher the default rate
plot_cat('installment')
```

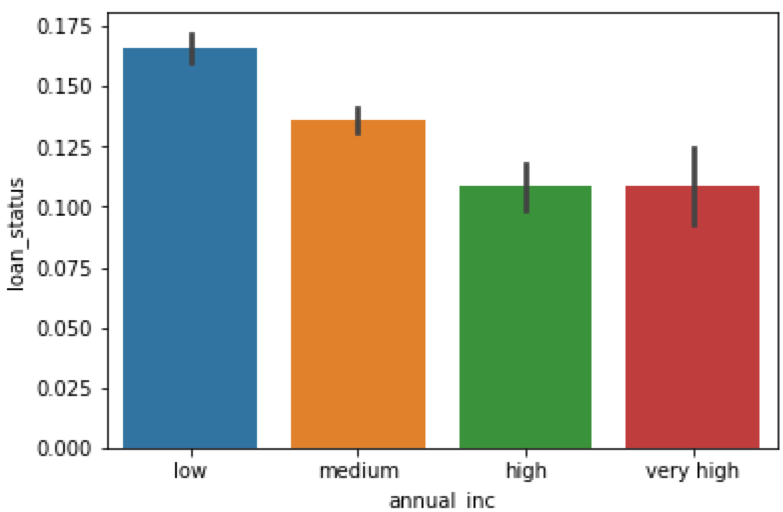


In [ ]:

```
# 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'

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

In [ ]:

```
# annual income and default rate
# lower the annual income, higher the default rate
plot_cat('annual_inc')
```



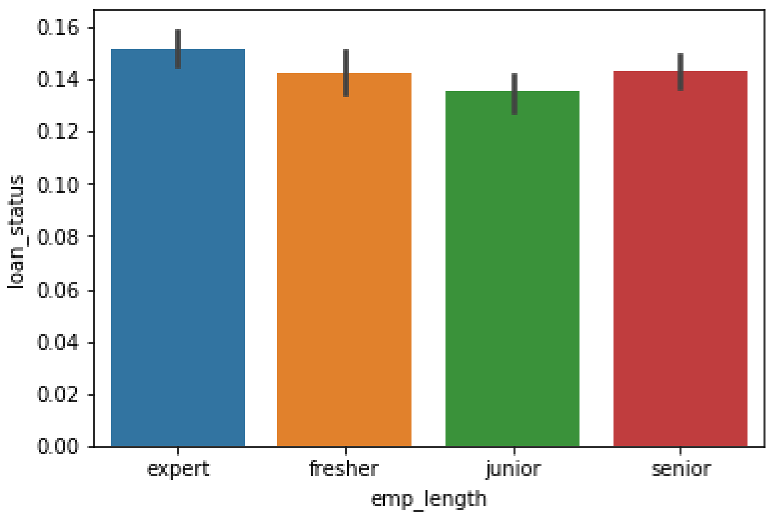
In [ ]:

```
# employment length
# first, let's drop the missing value observations in emp length
df = df[~df['emp_length'].isnull()]

# binning the variable
```

```
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 [ ]:

```
# emp_length and default rate
# not much of a predictor of default
plot_cat('emp_length')
```



Segmented Univariate Analysis

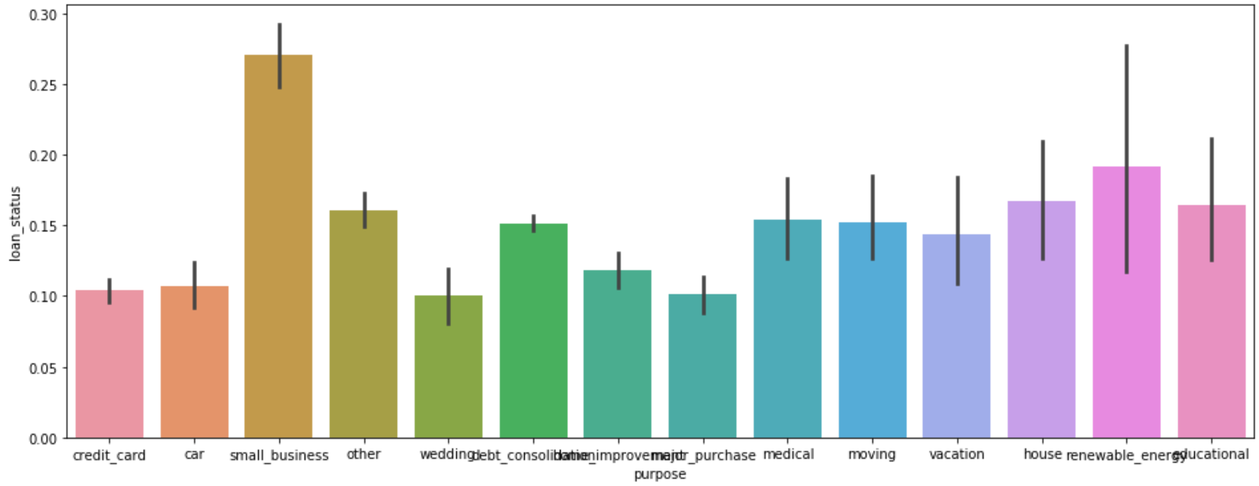
We have now compared the default rates across various variables, and some of the important predictors are purpose of the loan, interest rate, annual income, grade etc.

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.

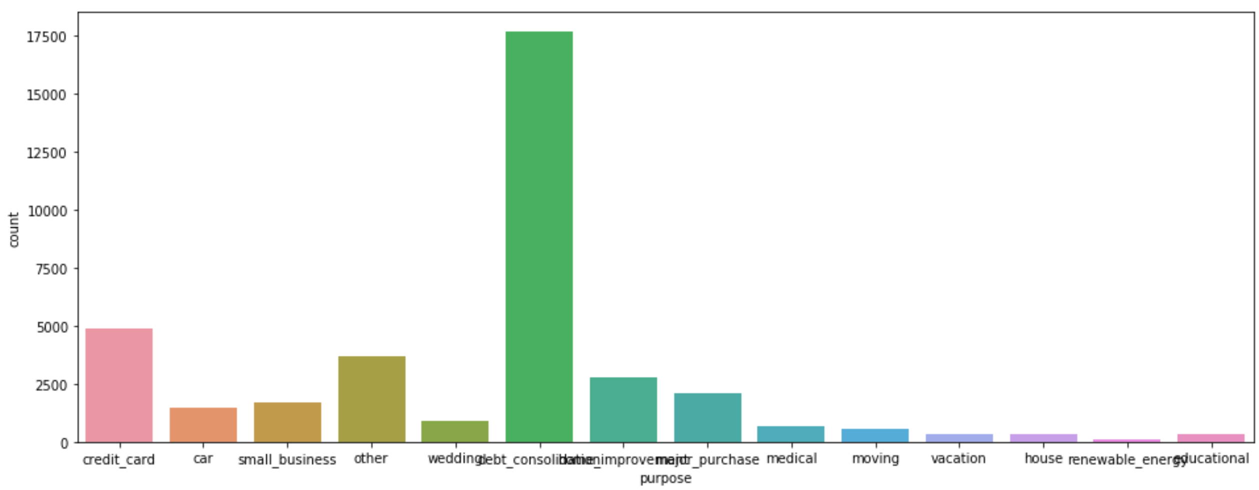
In [ ]:

```
# purpose: small business loans defualt the most, then renewable energy and education
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 [ ]:
```
# lets first look at the number of loans for each type (purpose) of the loan
# most loans are debt consolidation (to repay otehr debts), then credit card, major pur
plt.figure(figsize=(16, 6))
sns.countplot(x='purpose', data=df)
plt.show()
```



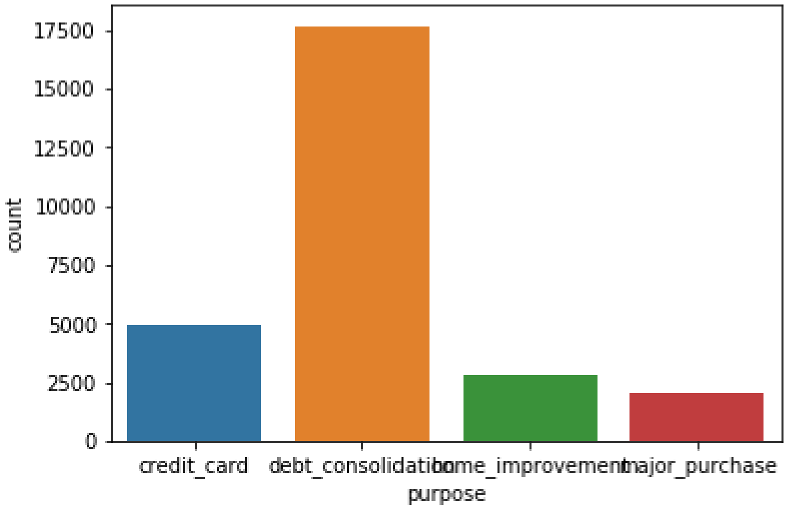
Let's analyse the top 4 types of loans based on purpose: consolidation, credit card, home improvement and major purchase.

In [ ]:
```
# filtering the df for the 4 types of loans mentioned above
main_purposes = ["credit_card","debt_consolidation","home_improvement","major_purchase"
df = df[df['purpose'].isin(main_purposes)]
df['purpose'].value_counts()
```

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

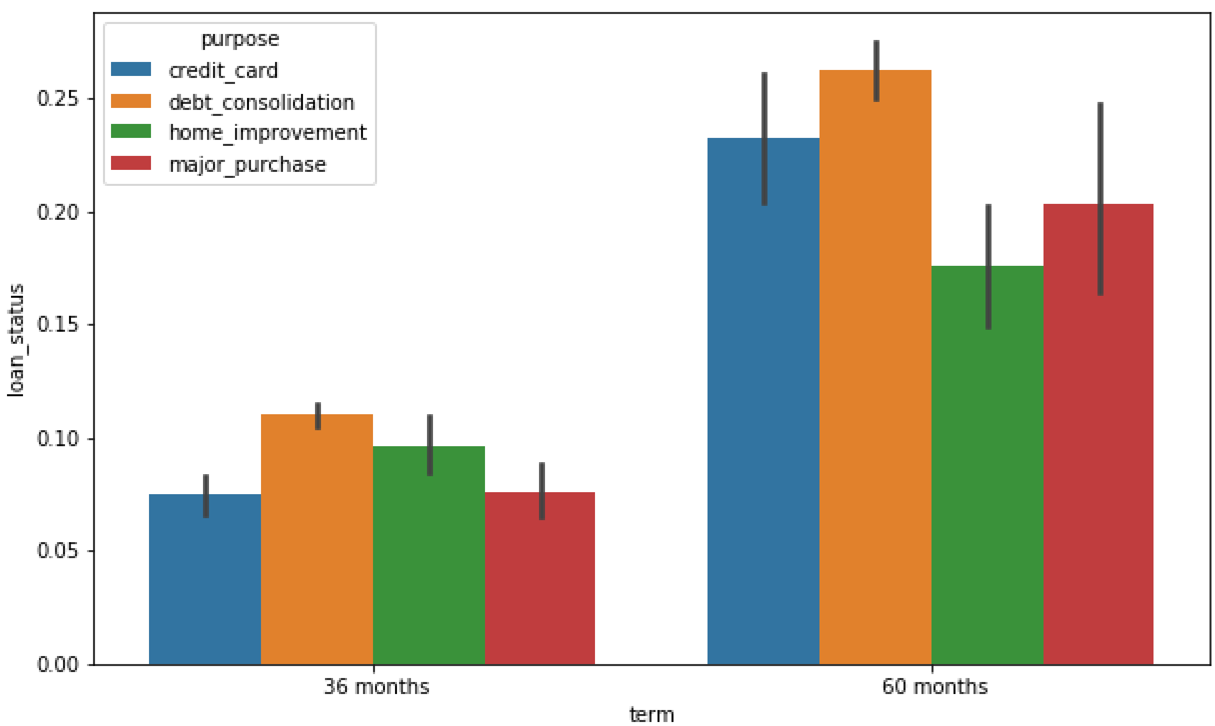
In [ ]:

```
# plotting number of loans by purpose
sns.countplot(x=df['purpose'])
plt.show()
```



In [ ]:

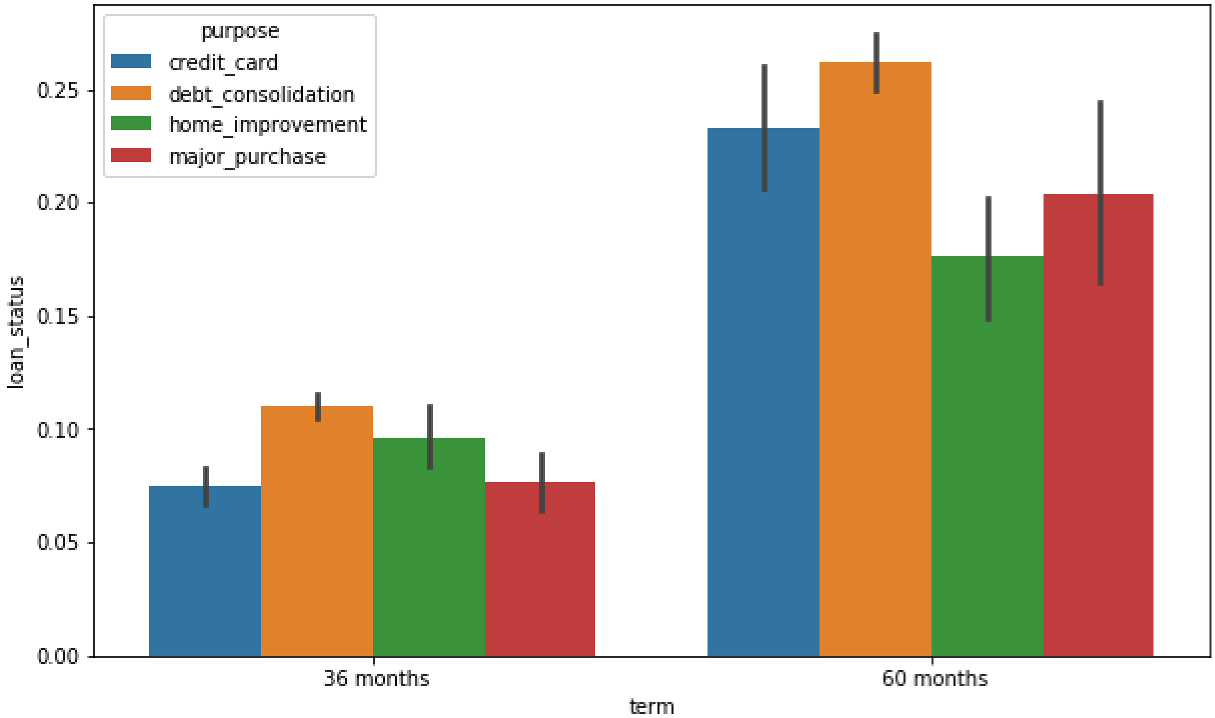
```
# let's now compare the default rates across two types of categorical variables
# purpose of loan (constant) and another categorical variable (which changes)

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



In [ ]:

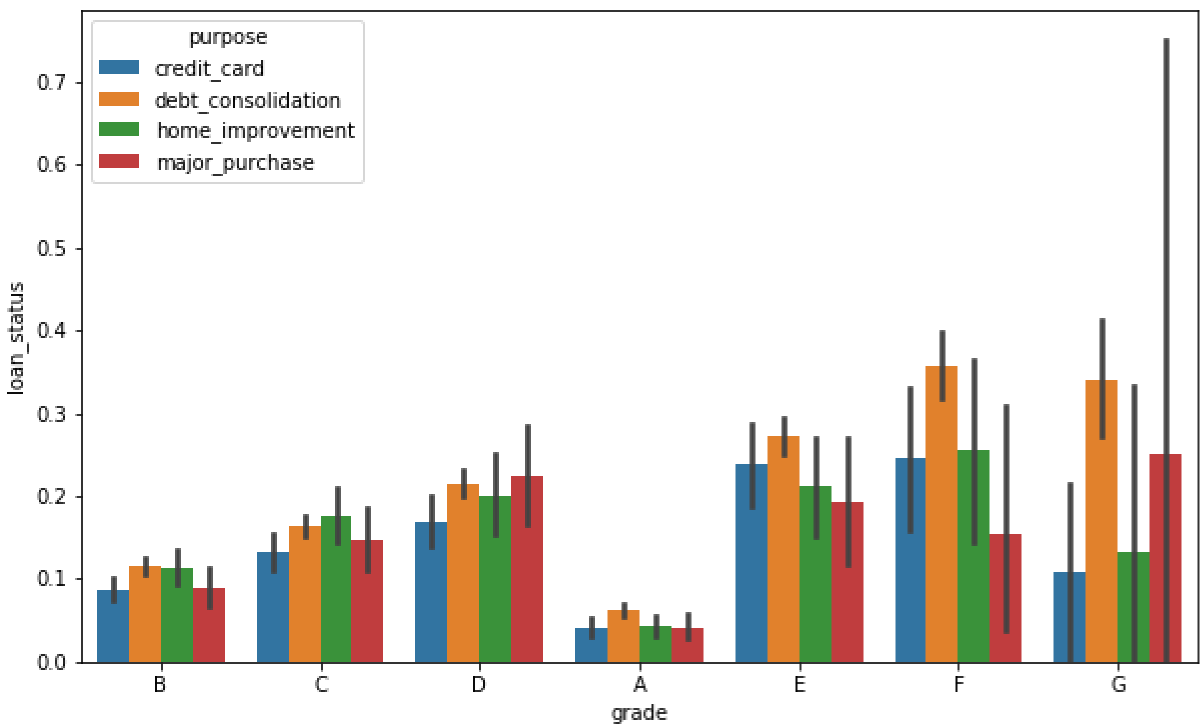
```
# lets write a function which takes a categorical variable and plots the default rate
# segmented by purpose

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')
```



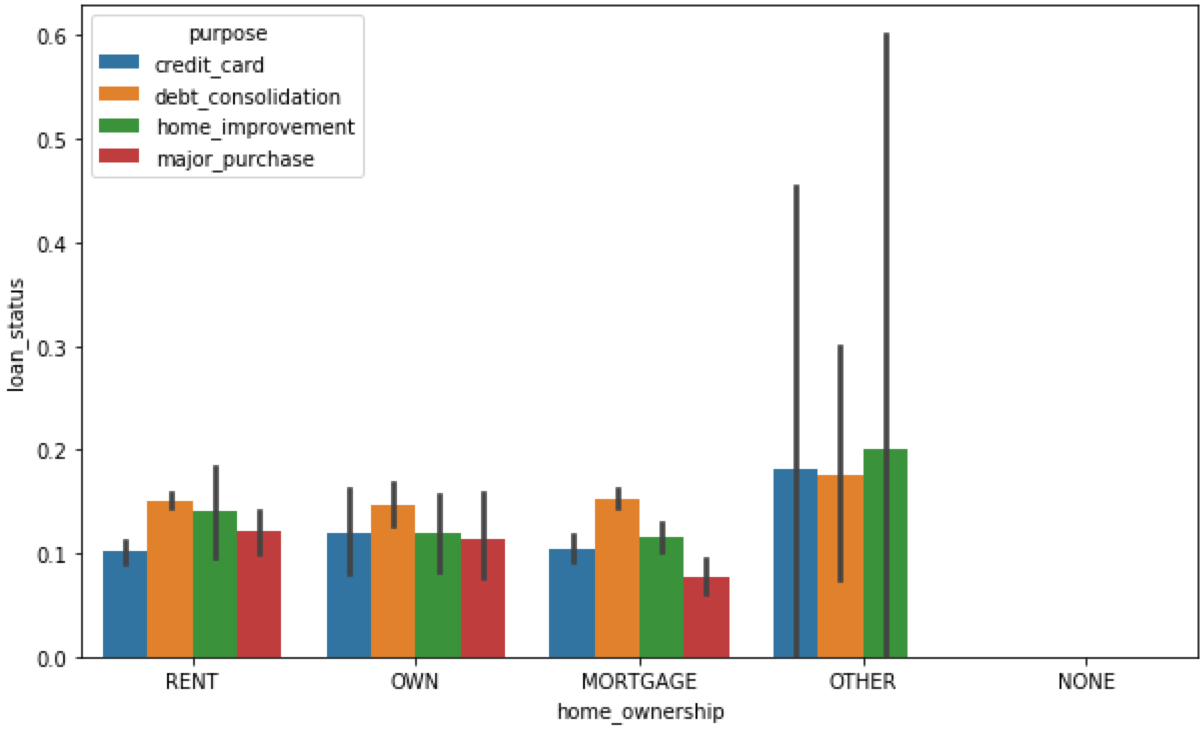
In [ ]:

```
# grade of loan
plot_segmented('grade')
```



In [ ]:

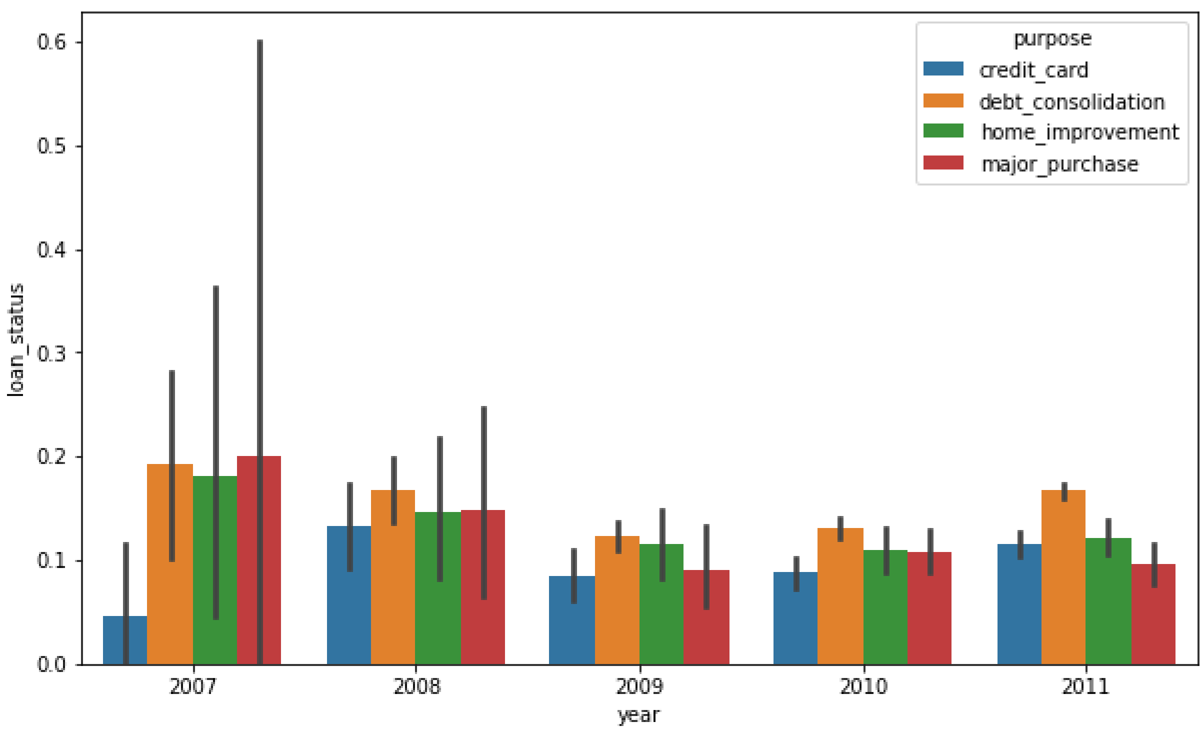
```
# home ownership
plot_segmented('home_ownership')
```



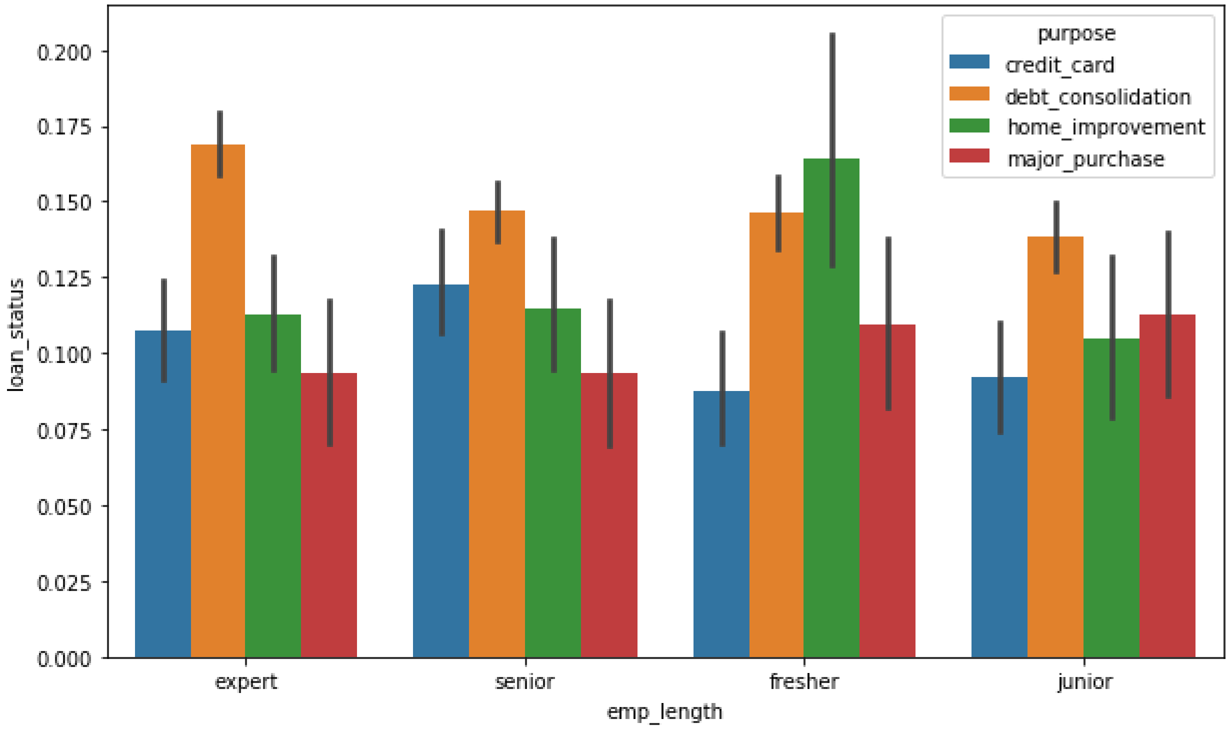
In general, debt consolidation loans have the highest default rates. Lets compare across other categories as well.

In [ ]:

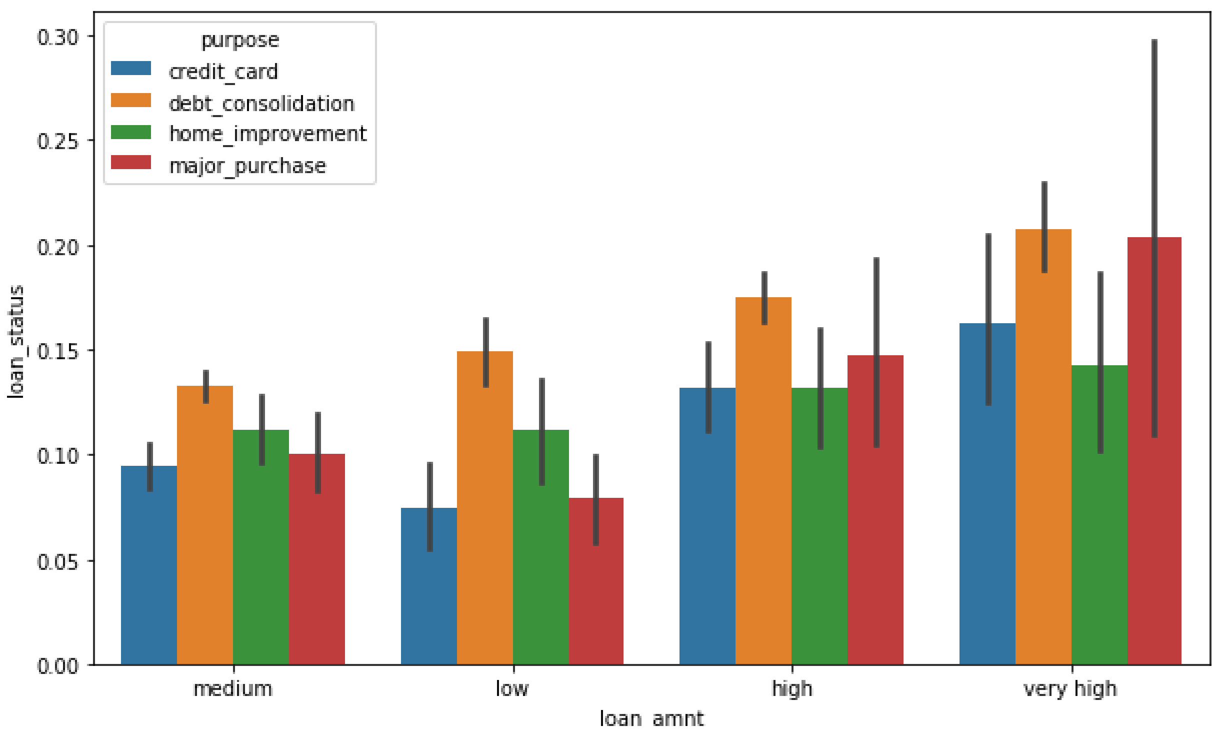
```
# year
plot_segmented('year')
```



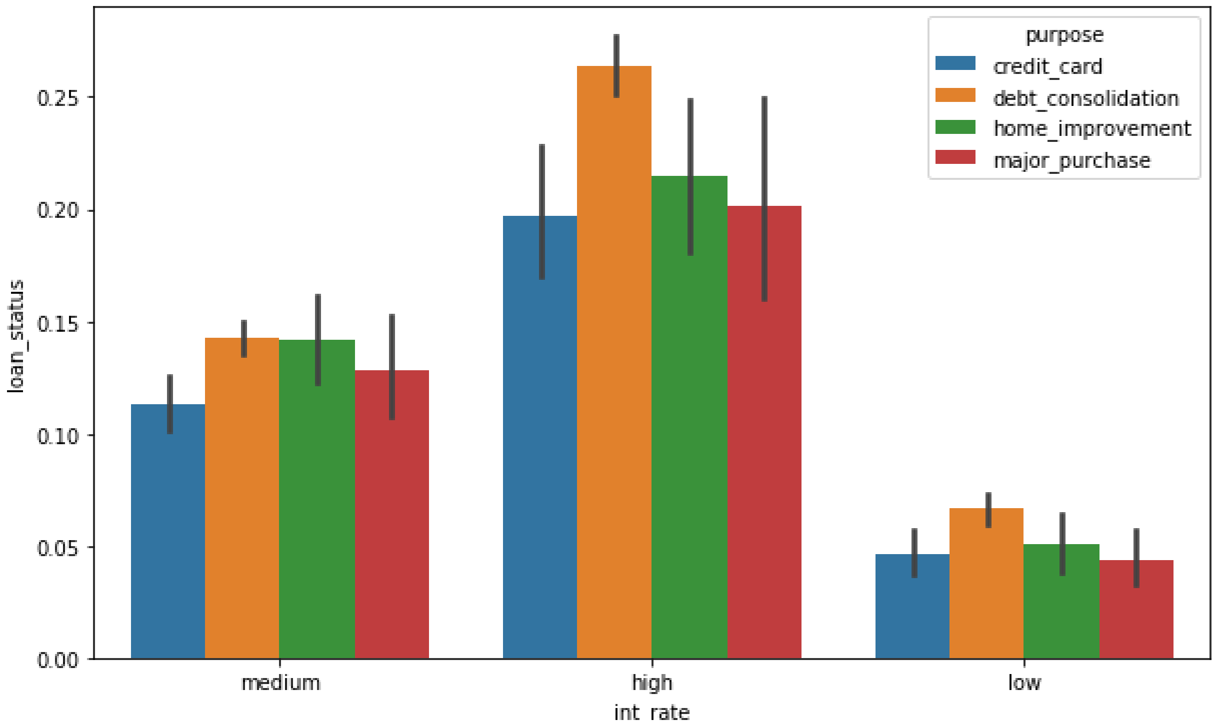
In [ ]:
```
# emp_length
plot_segmented('emp_length')
```



In [ ]:
```
# loan_amnt: same trend across loan purposes
plot_segmented('loan_amnt')
```

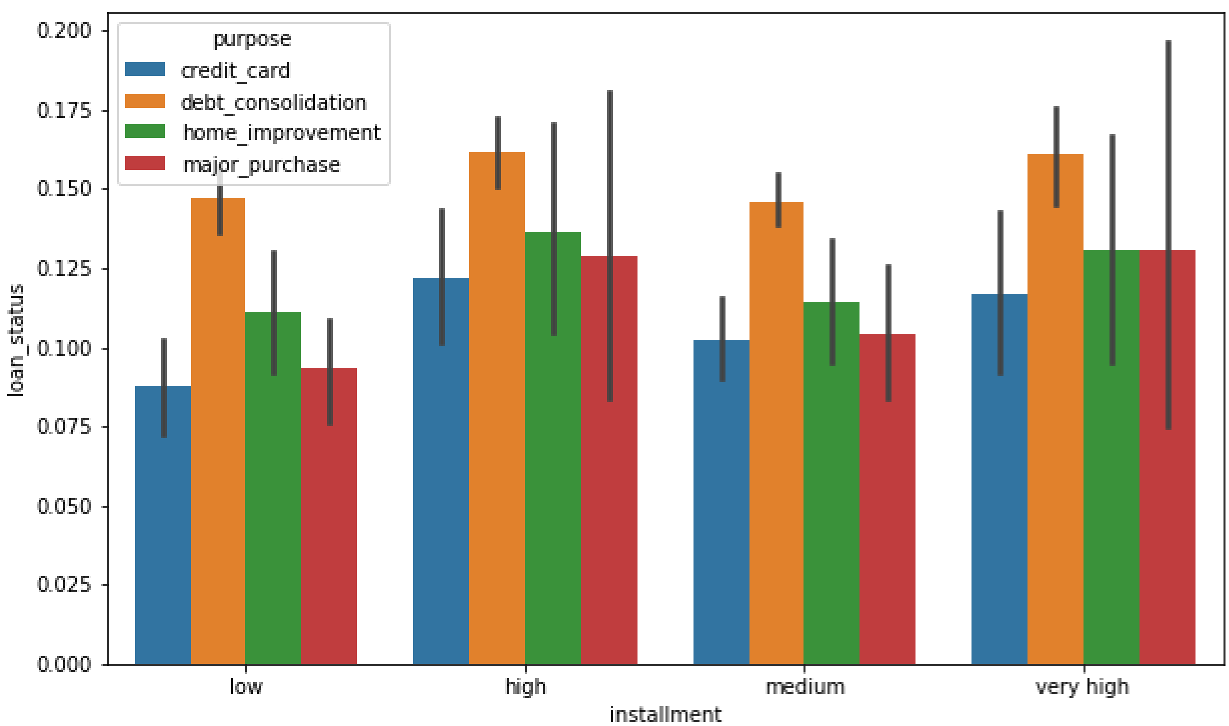


In [ ]:
```
# interest rate
plot_segmented('int_rate')
```



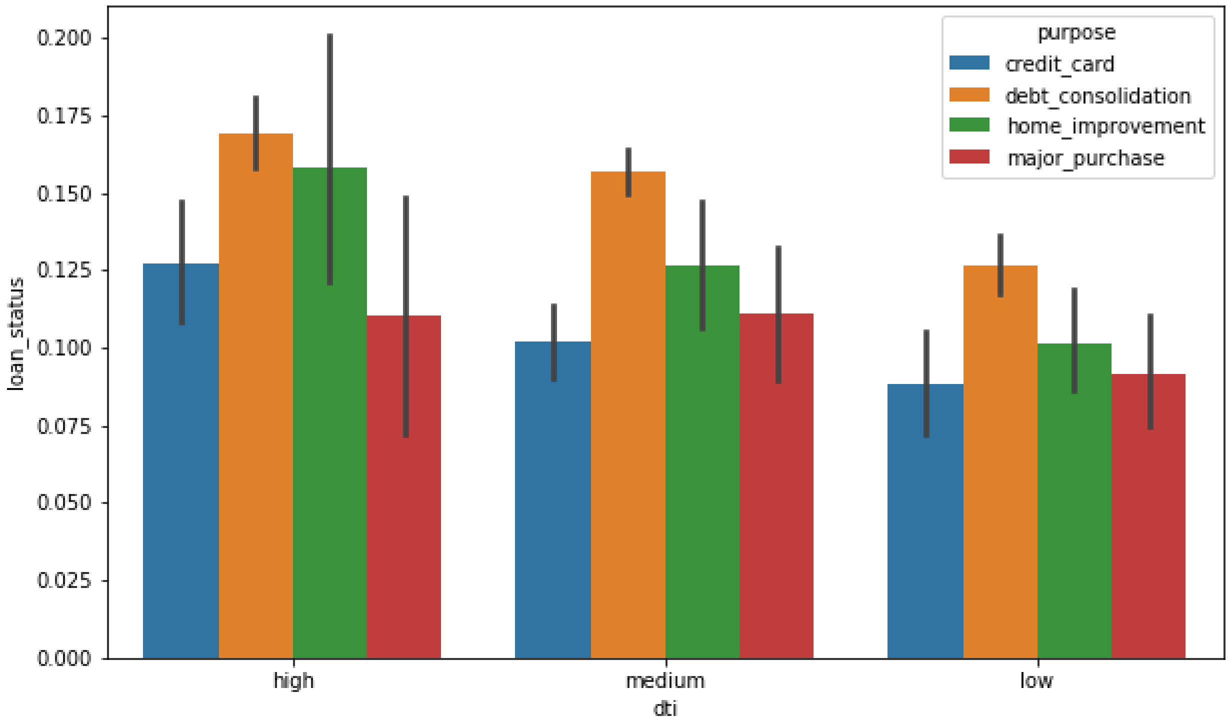
In [ ]:

```
# installment
plot_segmented('installment')
```

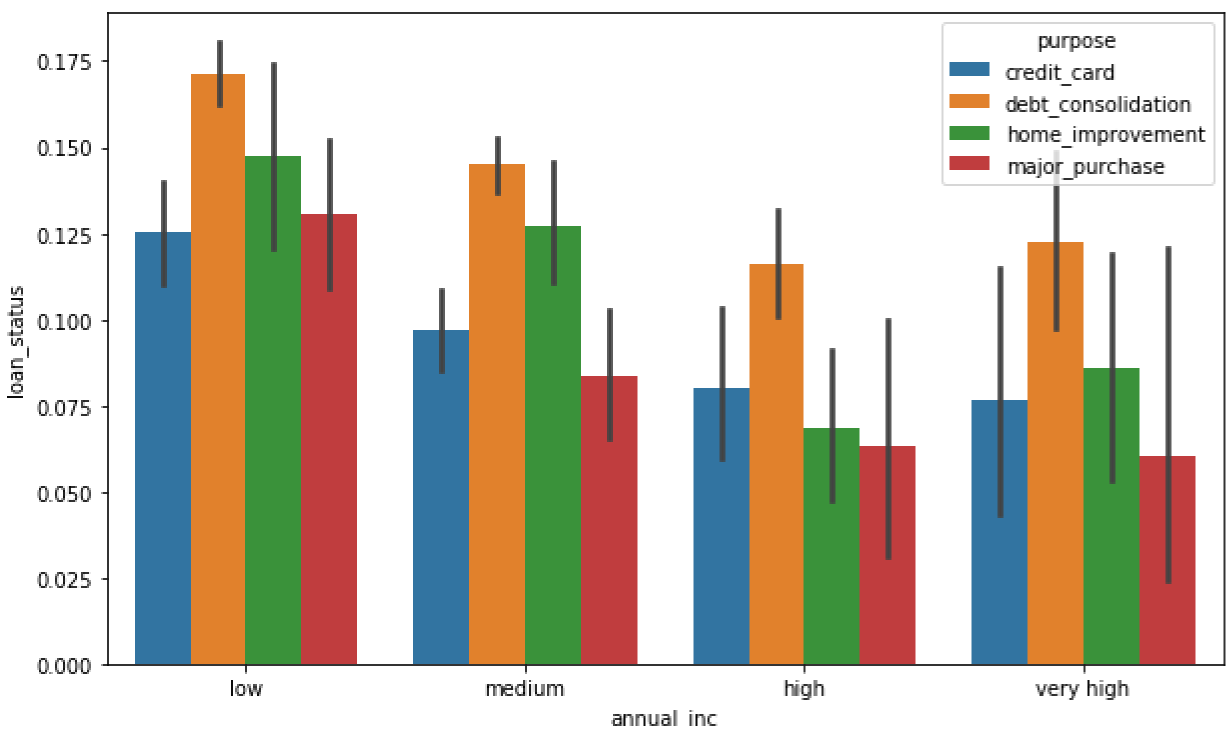


In [ ]:

```
# debt to income ratio
plot_segmented('dti')
```



In [ ]:
```
# annual income
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'.

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

In [ ]:
```
# variation of default rate across annual_inc
df.groupby('annual_inc').loan_status.mean().sort_values(ascending=False)
```

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

In [ ]:
```
# one can write a function which takes in a categorical variable and computed the avera
# default rate across the categories
# It can also compute the 'difference between the highest and the lowest default rate'
# categories, which is a decent metric indicating the effect of the varaible on default

def diff_rate(cat_var):
    default_rates = df.groupby(cat_var).loan_status.mean().sort_values(ascending=False)
    return (round(default_rates, 2), round(default_rates[0] - default_rates[-1], 2))

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
```

Thus, there is a 6% increase in default rate as you go from high to low annual income. We can compute this difference for all the variables and roughly identify the ones that affect default rate the most.

In [ ]:
```
# filtering all the object type variables
df_categorical = df.loc[:, df.dtypes == object]
df_categorical['loan_status'] = df['loan_status']

# Now, for each variable, we can compute the incremental diff in default rates
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', 'annua
l_inc', 'verification_status', 'issue_d', 'loan_status', 'pymnt_plan', 'purpose', 'dti',
'initial_list_status', 'collections_12_mths_ex_med', 'policy_code', 'acc_now_delinq', 'c
hargeoff_within_12_mths', 'delinq_amnt', 'pub_rec_bankruptcies', 'tax_liens', 'month',
'year']
```

```
/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/ipykernel_
launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexi
ng.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [ ]:
```
# storing the diff of default rates for each column in a dict
d = {key: diff_rate(key)[1]*100 for key in df_categorical.columns if key != 'loan_statu
```

```
print(d)
```

```
{'loan_amnt': 7.0000000000000009, 'funded_amnt_inv': 6.0, 'pymnt_plan': 0.0, 'verificati
on_status': 4.0, 'emp_title': 100.0, 'dti': 5.0, 'home_ownership': 16.0, 'purpose': 5.0,
'sub_grade': 46.0, 'grade': 27.0, 'funded_amnt': 5.0, 'installment': 3.0, 'initial_list_
status': 0.0, 'int_rate': 19.0, 'term': 15.0, 'annual_inc': 6.0, 'emp_length': 2.0}
```