Problem statement

Using Exploratory Data Analysis (EDA) for basic understanding the risk analytics in banking and financial services and understand how data is used to minimise the risk of losing money while lending to customers.

For the analysis, we follow the below five steps of EDA.

1. Data sourcing
2. Data cleaning
3. Univariate analysis
4. Bivariate analysis
5. Derived metrics

1. Data sourcing

Data is the key for any analysis. Data comes from various sources and your first job as a data analyst is to procure the data from them. Data is of two types

1. **Public Data :** A large amount of data collected by the government or other public agencies is made public for the purposes of research. Such data sets do not require special permission for access and are therefore called public data.
2. **Private Data :** Private data is the data which is sensitive to the organisation and hence it does not available in public domain.

Our data for the analysis is present in the form of CSV file in the file named **loan.csv** in the loan directory. Let us first get the data for our analysis.

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

In [2]:
```
loan_data = pd.read_csv("loan/loan.csv")
loan_data.head()
```

```
C:\Users\sbondugula\AppData\Local\Temp\ipykernel_10748\296403435.py:1: DtypeWarning: Col
umns (47) have mixed types. Specify dtype option on import or set low_memory=False.
  loan_data = pd.read_csv("loan/loan.csv")
```

Out[2]:

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

5 rows × 111 columns

In [3]:
```
loan_data.tail()
```

Out[3]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_
39712	92187	92174	2500	2500	1075.0	36 months	8.07%	78.42	A	
39713	90665	90607	8500	8500	875.0	36 months	10.28%	275.38	C	
39714	90395	90390	5000	5000	1325.0	36 months	8.07%	156.84	A	
39715	90376	89243	5000	5000	650.0	36 months	7.43%	155.38	A	
39716	87023	86999	7500	7500	800.0	36 months	13.75%	255.43	E	

5 rows × 111 columns

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

Out[4]:
```
(39717, 111)
```

Here we can see our data contains 39717 rows and 111 columns.

2. Data cleaning

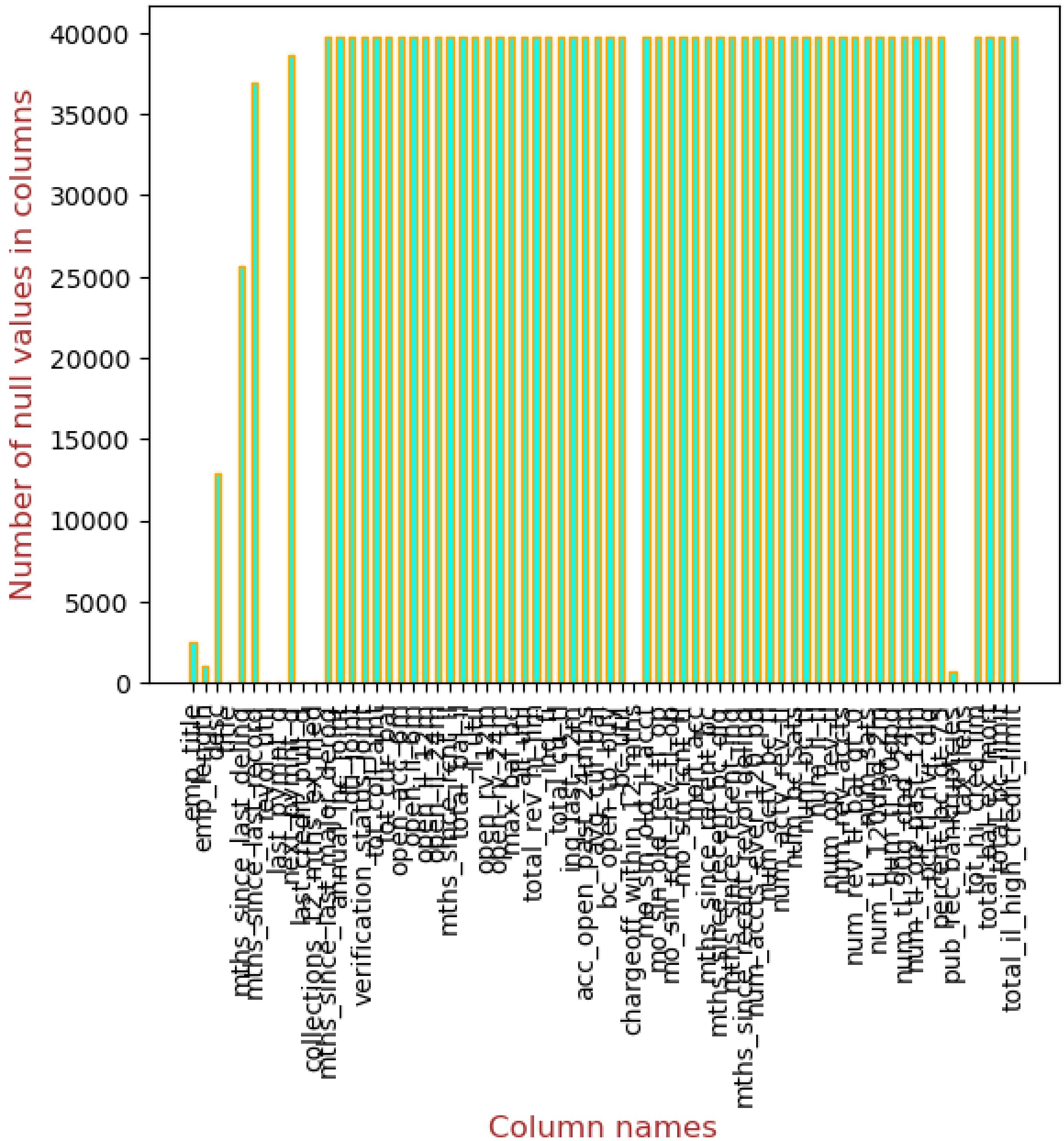
Our next step is to clean the data which we have read. Cleaning can be done in many ways as follows.

Fixing missing values

In [5]:
```
loan_data.isna().sum()[loan_data.isna().sum()>0]
```

Out[5]:
```
emp_title                    2459
emp_length                   1075
desc                        12940
title                          11
mths_since_last_delinq      25682
                            ...
tax_liens                      39
tot_hi_cred_lim             39717
total_bal_ex_mort           39717
total_bc_limit              39717
total_il_high_credit_limit  39717
Length: 68, dtype: int64
```

In [6]:
```
plt.bar(loan_data.isna().sum()[loan_data.isna().sum()>0].index,loan_data.isna().sum()[lo
plt.xlabel("Column names", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'
plt.ylabel("Number of null values in columns", fontdict={'fontsize': 12, 'fontweight' :
plt.xticks(rotation=90)
plt.show()
```



In [7]:
```
loan_data.isna().sum()[loan_data.isna().sum()>0].count()
```

Out[7]:
```
68
```

It is observed that there are 68 columns which have atleast a missing value and also some columns have almost all null values. From the above bar graph, we can see most of the columns have atleast 10000 missing values out of 39717 rows(which is almost 25 %).Let us make a cutoff of 10000 and delete all the columns which have more than 10000 null values/missing values.
For remaining columns, we can analyse further if we can clean them or not.

In [8]:
```
loan_data.isna().sum()[loan_data.isna().sum()>10000].index
```

Out[8]:
```
Index(['desc', 'mths_since_last_delinq', '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 [9]:
```
loan_data.drop(columns=loan_data.isna().sum()[loan_data.isna().sum()>10000].index,inplac
```

In [10]:
```
loan_data.shape
```

Out[10]:
```
(39717, 53)
```

We can observe there are 53 columns remaining.

Let us analysize missing values in these columns.

In [11]:
```
loan_data.isna().sum()[loan_data.isna().sum()>0]
```

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

Let us analyze the columns which have missing values.

In [12]:
```
loan_data[loan_data.isna().sum()[loan_data.isna().sum()>0].index]
```

Out[12]:

	emp_title	emp_length	title	revol_util	last_pymnt_d	last_credit_pull_d	collections_12_mths_ex_r
0	NaN	10+ years	Computer	83.70%	Jan-15	May-16	
1	Ryder	< 1 year	bike	9.40%	Apr-13	Sep-13	
2	NaN	10+ years	real estate business	98.50%	Jun-14	May-16	
3	AIR RESOURCES BOARD	10+ years	personel	21%	Jan-15	Apr-16	
4	University Medical Group	1 year	Personal	53.90%	May-16	May-16	
...	...	...	...	...	...	...	
39712	FiSite Research	4 years	Home Improvement	13.10%	Jul-10	Jun-10	I
39713	Squarewave Solutions, Ltd.	3 years	Retiring credit card debt	26.90%	Jul-10	Jul-10	I
39714	NaN	< 1 year	MBA Loan Consolidation	19.40%	Apr-08	Jun-07	I
39715	NaN	< 1 year	JAL Loan	0.70%	Jan-08	Jun-07	I

39716	Evergreen Center	< 1 year	Consolidation Loan	51.50%	Jun-10	Jun-10	I

39717 rows × 10 columns

Let us analyze the number of unique values in each of the above columns.

In [13]:
```
loan_data[loan_data.isna().sum()[loan_data.isna().sum()>0].index].nunique()
```

Out[13]:
```
emp_title                     28820
emp_length                       11
title                         19615
revol_util                     1089
last_pymnt_d                    101
last_credit_pull_d              106
collections_12_mths_ex_med        1
chargeoff_within_12_mths          1
pub_rec_bankruptcies              3
tax_liens                         1
dtype: int64
```

emp_title and title columns have more unique values which does not have any impact on analysis. Let us remove them.

In [14]:
```
loan_data.drop(columns=["emp_title","title"],inplace=True)
```

In [15]:
```
loan_data.shape
```

Out[15]:
```
(39717, 51)
```

Some columns have only one unique value. They cannot be used for any analysis. Let us remove them.

In [16]:
```
loan_data.drop(columns=loan_data.nunique()[loan_data.nunique()==1].index,inplace=True)
```

In [17]:
```
loan_data.shape
```

Out[17]:
```
(39717, 42)
```

In [18]:
```
loan_data.head()
```

Out[18]:

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

5 rows × 42 columns

In [19]:
```
loan_data.columns
```

Out[19]:
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'url', 'purpose', 'zip_code', 'addr_state', 'dti',
       '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', 'pub_rec_bankruptcies'],
      dtype='object')
```

From the above columns - total_rec_int, total_rec_prncp, total_rec_late_fee, last_credit_pull_d, recoveries, collection_recovery_fee, last_pymnt_d,out_prncp,out_prncp_inv are the features of post loan approval, which may not need for us to analyse loan approval.

In [20]:
```
loan_data.drop(columns=['total_rec_int', 'total_rec_prncp', 'total_rec_late_fee', 'last_
```

In [21]:
```
loan_data.shape
```

Out[21]:
```
(39717, 33)
```

In [22]:
```
loan_data.nunique()
```

Out[22]:
```
id                         39717
member_id                  39717
loan_amnt                    885
funded_amnt                 1041
funded_amnt_inv             8205
term                           2
int_rate                     371
installment                15383
grade                          7
sub_grade                     35
emp_length                    11
home_ownership                 5
annual_inc                  5318
verification_status            3
issue_d                       55
loan_status                    3
url                        39717
purpose                       14
zip_code                     823
addr_state                    50
dti                         2868
delinq_2yrs                   11
earliest_cr_line             526
inq_last_6mths                 9
open_acc                      40
pub_rec                        5
revol_bal                  21711
revol_util                  1089
total_acc                     82
total_pymnt                37850
total_pymnt_inv            37518
last_pymnt_amnt            34930
pub_rec_bankruptcies           3
dtype: int64
```

As we can see, id,member_id and url are unique for each borrower. So we can drop those columns as they are not needed for further analysis.

In [23]:
```
loan_data.drop(columns=["id","member_id","url"],inplace=True)
```

In [24]:
```
loan_data.nunique()
```

Out[24]:
```
loan_amnt                  885
funded_amnt               1041
funded_amnt_inv           8205
term                         2
int_rate                   371
installment              15383
grade                        7
sub_grade                   35
emp_length                  11
home_ownership               5
annual_inc                5318
verification_status          3
issue_d                     55
loan_status                  3
purpose                     14
zip_code                   823
addr_state                  50
dti                       2868
delinq_2yrs                 11
earliest_cr_line           526
inq_last_6mths               9
open_acc                    40
pub_rec                      5
revol_bal                21711
revol_util                1089
total_acc                   82
total_pymnt              37850
total_pymnt_inv          37518
last_pymnt_amnt          34930
pub_rec_bankruptcies         3
dtype: int64
```

In [25]:
```
loan_data.isna().sum()
```

Out[25]:
```
loan_amnt                  0
funded_amnt                0
funded_amnt_inv            0
term                       0
int_rate                   0
installment                0
grade                      0
sub_grade                  0
emp_length              1075
home_ownership             0
annual_inc                 0
verification_status        0
issue_d                    0
loan_status                0
purpose                    0
zip_code                   0
addr_state                 0
dti                        0
delinq_2yrs                0
earliest_cr_line           0
inq_last_6mths             0
open_acc                   0
pub_rec                    0
revol_bal                  0
revol_util                50
total_acc                  0
total_pymnt                0
total_pymnt_inv            0
last_pymnt_amnt            0
```

```
pub_rec_bankruptcies      697
dtype: int64
```

Now we have only three columns which have missing values. Let us now find what percent of rows, these columns have missing values. If the percentage is less, we can delete those rows.

In [26]:
```
percent_of_missing_values = 100*loan_data.isna().sum()/len(loan_data)
percent_of_missing_values[100*loan_data.isna().sum()/len(loan_data)>0]
```

Out[26]:
```
emp_length              2.706650
revol_util              0.125891
pub_rec_bankruptcies    1.754916
dtype: float64
```

As we can clearly see the percentage is less than 3, we can delete those rows with missing values.

In [27]:
```
loan_data.dropna(subset=['emp_length', 'revol_util', 'pub_rec_bankruptcies'],axis=0, inp
```

In [28]:
```
loan_data.isna().sum()
```

Out[28]:
```
loan_amnt               0
funded_amnt             0
funded_amnt_inv         0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
zip_code                0
addr_state              0
dti                     0
delinq_2yrs             0
earliest_cr_line        0
inq_last_6mths          0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
total_pymnt             0
total_pymnt_inv         0
last_pymnt_amnt         0
pub_rec_bankruptcies    0
dtype: int64
```

Now we have removed the missing / NAN values from the data.

In [29]:
```
loan_data.shape
```

Out[29]:
```
(37898, 30)
```

In [30]:
```
loan_data.head()
```

Out[30]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_length	hom
0	5000	5000	4975.0	36	10.65%	162.87	B	B2	10+ years	

				term					
				months					
1	2500	2500	2500.0	60 months	15.27%	59.83	C	C4	< 1 year
2	2400	2400	2400.0	36 months	15.96%	84.33	C	C5	10+ years
3	10000	10000	10000.0	36 months	13.49%	339.31	C	C1	10+ years
4	3000	3000	3000.0	60 months	12.69%	67.79	B	B5	1 year

5 rows × 30 columns

In [31]:
```
loan_data.loan_status.unique()
```
Out[31]:
```
array(['Fully Paid', 'Charged Off', 'Current'], dtype=object)
```

The column loan_status has three values (Fully Paid, Charged Off, Current). Here we cannot analyse whether the applicant is likely to default or not based on the Current status. So we can remove the rows with loan_status is 'Current'

In [32]:
```
loan_data=loan_data[loan_data.loan_status!='Current']
loan_data.head()
```
Out[32]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_length	hom
0	5000	5000	4975.0	36 months	10.65%	162.87	B	B2	10+ years	
1	2500	2500	2500.0	60 months	15.27%	59.83	C	C4	< 1 year	
2	2400	2400	2400.0	36 months	15.96%	84.33	C	C5	10+ years	
3	10000	10000	10000.0	36 months	13.49%	339.31	C	C1	10+ years	
5	5000	5000	5000.0	36 months	7.90%	156.46	A	A4	3 years	

5 rows × 30 columns

In [33]:
```
loan_data.loan_status.unique()
```
Out[33]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [34]:
```
loan_data.shape
```
Out[34]:
```
(36800, 30)
```

Standardising values

Let us check the datatypes of the columns

In [35]:
```
loan_data.info()
```
```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 36800 entries, 0 to 39680
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   loan_amnt             36800 non-null  int64  
 1   funded_amnt           36800 non-null  int64  
 2   funded_amnt_inv       36800 non-null  float64
 3   term                  36800 non-null  object 
 4   int_rate              36800 non-null  object 
 5   installment           36800 non-null  float64
 6   grade                 36800 non-null  object 
 7   sub_grade             36800 non-null  object 
 8   emp_length            36800 non-null  object 
 9   home_ownership        36800 non-null  object 
 10  annual_inc            36800 non-null  float64
 11  verification_status   36800 non-null  object 
 12  issue_d               36800 non-null  object 
 13  loan_status           36800 non-null  object 
 14  purpose               36800 non-null  object 
 15  zip_code              36800 non-null  object 
 16  addr_state            36800 non-null  object 
 17  dti                   36800 non-null  float64
 18  delinq_2yrs           36800 non-null  int64  
 19  earliest_cr_line      36800 non-null  object 
 20  inq_last_6mths        36800 non-null  int64  
 21  open_acc              36800 non-null  int64  
 22  pub_rec               36800 non-null  int64  
 23  revol_bal             36800 non-null  int64  
 24  revol_util            36800 non-null  object 
 25  total_acc             36800 non-null  int64  
 26  total_pymnt           36800 non-null  float64
 27  total_pymnt_inv       36800 non-null  float64
 28  last_pymnt_amnt       36800 non-null  float64
 29  pub_rec_bankruptcies  36800 non-null  float64
dtypes: float64(8), int64(8), object(14)
memory usage: 8.7+ MB
```

In [36]:
```
loan_data.int_rate.unique()
```

Out[36]:
```
array(['10.65%', '15.27%', '15.96%', '13.49%', '7.90%', '18.64%',
       '21.28%', '12.69%', '14.65%', '9.91%', '16.29%', '6.03%', '11.71%',
       '12.42%', '14.27%', '16.77%', '7.51%', '8.90%', '18.25%', '6.62%',
       '19.91%', '17.27%', '17.58%', '21.67%', '19.42%', '20.89%',
       '20.30%', '23.91%', '19.03%', '23.13%', '22.74%', '22.35%',
       '22.06%', '24.11%', '6.00%', '23.52%', '22.11%', '7.49%', '11.99%',
       '5.99%', '10.99%', '9.99%', '18.79%', '11.49%', '15.99%', '16.49%',
       '6.99%', '12.99%', '15.23%', '14.79%', '5.42%', '8.49%', '10.59%',
       '17.49%', '15.62%', '19.29%', '13.99%', '18.39%', '16.89%',
       '17.99%', '20.99%', '22.85%', '19.69%', '20.62%', '20.25%',
       '21.36%', '23.22%', '21.74%', '22.48%', '23.59%', '12.62%',
       '18.07%', '11.63%', '7.91%', '7.42%', '11.14%', '20.20%', '12.12%',
       '19.39%', '16.11%', '17.54%', '22.64%', '16.59%', '17.19%',
       '12.87%', '20.69%', '9.67%', '21.82%', '19.79%', '18.49%',
       '13.84%', '22.94%', '24.40%', '21.48%', '14.82%', '7.29%',
       '17.88%', '20.11%', '16.02%', '13.43%', '14.91%', '13.06%',
       '15.28%', '15.65%', '17.14%', '11.11%', '10.37%', '14.17%',
       '16.40%', '17.51%', '7.66%', '10.74%', '5.79%', '6.92%', '10.00%',
       '9.63%', '14.54%', '12.68%', '18.62%', '19.36%', '13.80%',
       '18.99%', '21.59%', '20.85%', '21.22%', '19.74%', '20.48%',
       '6.91%', '12.23%', '12.61%', '10.36%', '6.17%', '6.54%', '9.25%',
       '16.69%', '15.95%', '8.88%', '13.35%', '9.62%', '16.32%', '12.98%',
       '14.83%', '13.72%', '14.09%', '14.46%', '20.03%', '17.80%',
       '15.20%', '15.57%', '18.54%', '19.66%', '17.06%', '18.17%',
       '17.43%', '20.40%', '20.77%', '18.91%', '21.14%', '17.44%',
       '13.23%', '11.12%', '7.88%', '13.61%', '10.38%', '17.56%',
       '17.93%', '15.58%', '13.98%', '14.84%', '15.21%', '6.76%', '6.39%',
```

```
'11.86%', '7.14%', '14.35%', '16.82%', '10.75%', '14.72%',
'16.45%', '20.53%', '19.41%', '20.16%', '21.27%', '18.30%',
'18.67%', '19.04%', '20.90%', '21.64%', '12.73%', '10.25%',
'13.11%', '10.62%', '13.48%', '14.59%', '16.07%', '15.70%',
'9.88%', '11.36%', '15.33%', '13.85%', '14.96%', '14.22%', '7.74%',
'13.22%', '13.57%', '8.59%', '17.04%', '14.61%', '8.94%', '12.18%',
'11.83%', '11.48%', '16.35%', '13.92%', '15.31%', '14.26%',
'19.13%', '12.53%', '16.70%', '16.00%', '17.39%', '18.09%',
'7.40%', '18.43%', '17.74%', '7.05%', '20.52%', '20.86%', '19.47%',
'18.78%', '21.21%', '19.82%', '20.17%', '13.16%', '8.00%',
'13.47%', '12.21%', '16.63%', '9.32%', '12.84%', '11.26%',
'15.68%', '15.37%', '10.95%', '11.89%', '14.11%', '13.79%',
'7.68%', '11.58%', '7.37%', '16.95%', '15.05%', '18.53%', '14.74%',
'14.42%', '18.21%', '17.26%', '18.84%', '17.90%', '19.16%',
'13.67%', '9.38%', '12.72%', '13.36%', '11.46%', '10.51%', '9.07%',
'13.04%', '11.78%', '12.41%', '10.83%', '12.09%', '17.46%',
'14.30%', '17.15%', '15.25%', '10.20%', '15.88%', '14.93%',
'16.20%', '18.72%', '14.62%', '8.32%', '14.12%', '10.96%',
'10.33%', '10.01%', '12.86%', '11.28%', '11.59%', '8.63%',
'12.54%', '12.22%', '11.91%', '15.38%', '16.96%', '9.70%',
'16.33%', '14.75%', '13.17%', '15.07%', '16.01%', '10.71%',
'10.64%', '9.76%', '11.34%', '10.39%', '13.87%', '11.03%',
'11.66%', '13.24%', '10.08%', '9.45%', '13.55%', '12.29%',
'11.97%', '12.92%', '15.45%', '14.50%', '14.18%', '15.13%',
'16.08%', '15.76%', '17.03%', '10.46%', '13.93%', '10.78%',
'9.51%', '12.36%', '13.30%', '9.83%', '9.01%', '10.91%', '10.28%',
'12.49%', '11.22%'], dtype=object)
```

As we can see above, the int_rate column is in the form of object appended with '%', we can remove it and convert to float, so that it can be used for our analysis.

In [37]:
```
loan_data.int_rate=loan_data.int_rate.apply(lambda x:float(str(x).rstrip('%')))
```

In [38]:
```
loan_data.int_rate.unique()
```

Out[38]:
```
array([10.65, 15.27, 15.96, 13.49,  7.9 , 18.64, 21.28, 12.69, 14.65,
        9.91, 16.29,  6.03, 11.71, 12.42, 14.27, 16.77,  7.51,  8.9 ,
       18.25,  6.62, 19.91, 17.27, 17.58, 21.67, 19.42, 20.89, 20.3 ,
       23.91, 19.03, 23.13, 22.74, 22.35, 22.06, 24.11,  6.  , 23.52,
       22.11,  7.49, 11.99,  5.99, 10.99,  9.99, 18.79, 11.49, 15.99,
       16.49,  6.99, 12.99, 15.23, 14.79,  5.42,  8.49, 10.59, 17.49,
       15.62, 19.29, 13.99, 18.39, 16.89, 17.99, 20.99, 22.85, 19.69,
       20.62, 20.25, 21.36, 23.22, 21.74, 22.48, 23.59, 12.62, 18.07,
       11.63,  7.91,  7.42, 11.14, 20.2 , 12.12, 19.39, 16.11, 17.54,
       22.64, 16.59, 17.19, 12.87, 20.69,  9.67, 21.82, 19.79, 18.49,
       13.84, 22.94, 24.4 , 21.48, 14.82,  7.29, 17.88, 20.11, 16.02,
       13.43, 14.91, 13.06, 15.28, 15.65, 17.14, 11.11, 10.37, 14.17,
       16.4 , 17.51,  7.66, 10.74,  5.79,  6.92, 10.  ,  9.63, 14.54,
       12.68, 18.62, 19.36, 13.8 , 18.99, 21.59, 20.85, 21.22, 19.74,
       20.48,  6.91, 12.23, 12.61, 10.36,  6.17,  6.54,  9.25, 16.69,
       15.95,  8.88, 13.35,  9.62, 16.32, 12.98, 14.83, 13.72, 14.09,
       14.46, 20.03, 17.8 , 15.2 , 15.57, 18.54, 19.66, 17.06, 18.17,
       17.43, 20.4 , 20.77, 18.91, 21.14, 17.44, 13.23, 11.12,  7.88,
       13.61, 10.38, 17.56, 17.93, 15.58, 13.98, 14.84, 15.21,  6.76,
        6.39, 11.86,  7.14, 14.35, 16.82, 10.75, 14.72, 16.45, 20.53,
       19.41, 20.16, 21.27, 18.3 , 18.67, 19.04, 20.9 , 21.64, 12.73,
       10.25, 13.11, 10.62, 13.48, 14.59, 16.07, 15.7 ,  9.88, 11.36,
       15.33, 13.85, 14.96, 14.22,  7.74, 13.22, 13.57,  8.59, 17.04,
       14.61,  8.94, 12.18, 11.83, 11.48, 16.35, 13.92, 15.31, 14.26,
       19.13, 12.53, 16.7 , 16.  , 17.39, 18.09,  7.4 , 18.43, 17.74,
        7.05, 20.52, 20.86, 19.47, 18.78, 21.21, 19.82, 20.17, 13.16,
        8.  , 13.47, 12.21, 16.63,  9.32, 12.84, 11.26, 15.68, 15.37,
       10.95, 11.89, 14.11, 13.79,  7.68, 11.58,  7.37, 16.95, 15.05,
       18.53, 14.74, 14.42, 18.21, 17.26, 18.84, 17.9 , 19.16, 13.67,
        9.38, 12.72, 13.36, 11.46, 10.51,  9.07, 13.04, 11.78, 12.41,
```

```
       10.83, 12.09, 17.46, 14.3 , 17.15, 15.25, 10.2 , 15.88, 14.93,
       16.2 , 18.72, 14.62,  8.32, 14.12, 10.96, 10.33, 10.01, 12.86,
       11.28, 11.59,  8.63, 12.54, 12.22, 11.91, 15.38, 16.96,  9.7 ,
       16.33, 14.75, 13.17, 15.07, 16.01, 10.71, 10.64,  9.76, 11.34,
       10.39, 13.87, 11.03, 11.66, 13.24, 10.08,  9.45, 13.55, 12.29,
       11.97, 12.92, 15.45, 14.5 , 14.18, 15.13, 16.08, 15.76, 17.03,
       10.46, 13.93, 10.78,  9.51, 12.36, 13.3 ,  9.83,  9.01, 10.91,
       10.28, 12.49, 11.22])
```

int_rate is now standardised to float for statistical computation.

In [39]:
```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36800 entries, 0 to 39680
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   loan_amnt             36800 non-null  int64  
 1   funded_amnt           36800 non-null  int64  
 2   funded_amnt_inv       36800 non-null  float64
 3   term                  36800 non-null  object 
 4   int_rate              36800 non-null  float64
 5   installment           36800 non-null  float64
 6   grade                 36800 non-null  object 
 7   sub_grade             36800 non-null  object 
 8   emp_length            36800 non-null  object 
 9   home_ownership        36800 non-null  object 
 10  annual_inc            36800 non-null  float64
 11  verification_status   36800 non-null  object 
 12  issue_d               36800 non-null  object 
 13  loan_status           36800 non-null  object 
 14  purpose               36800 non-null  object 
 15  zip_code              36800 non-null  object 
 16  addr_state            36800 non-null  object 
 17  dti                   36800 non-null  float64
 18  delinq_2yrs           36800 non-null  int64  
 19  earliest_cr_line      36800 non-null  object 
 20  inq_last_6mths        36800 non-null  int64  
 21  open_acc              36800 non-null  int64  
 22  pub_rec               36800 non-null  int64  
 23  revol_bal             36800 non-null  int64  
 24  revol_util            36800 non-null  object 
 25  total_acc             36800 non-null  int64  
 26  total_pymnt           36800 non-null  float64
 27  total_pymnt_inv       36800 non-null  float64
 28  last_pymnt_amnt       36800 non-null  float64
 29  pub_rec_bankruptcies  36800 non-null  float64
dtypes: float64(9), int64(8), object(13)
memory usage: 8.7+ MB
```

In [40]:
```
loan_data.issue_d.unique()
```

Out[40]:
```
array(['Dec-11', 'Nov-11', 'Oct-11', 'Sep-11', 'Aug-11', 'Jul-11',
       'Jun-11', 'May-11', 'Apr-11', 'Mar-11', 'Feb-11', 'Jan-11',
       'Dec-10', 'Nov-10', 'Oct-10', 'Sep-10', 'Aug-10', 'Jul-10',
       'Jun-10', 'May-10', 'Apr-10', 'Mar-10', 'Feb-10', 'Jan-10',
       'Dec-09', 'Nov-09', 'Oct-09', 'Sep-09', 'Aug-09', 'Jul-09',
       'Jun-09', 'May-09', 'Apr-09', 'Mar-09', 'Feb-09', 'Jan-09',
       'Dec-08', 'Nov-08', 'Oct-08', 'Sep-08', 'Aug-08', 'Jul-08',
       'Jun-08', 'May-08', 'Apr-08', 'Mar-08', 'Feb-08', 'Jan-08',
       'Dec-07', 'Nov-07', 'Oct-07', 'Aug-07'], dtype=object)
```

As we can see above, the column "issue_d" which is the combination of month and date can be split into two separate columns for month and year.

The original column issue_d can be removed after creating new columns.

In [41]:
```
#Converting month to its respective number so that it can be ordered categorical variabl
month_num_dict = {"Jan":1,"Feb":2,"Mar":3,"Apr":4,"May":5,"Jun":6,"Jul":7,"Aug":8,"Sep":

loan_data['issue_id_month']=loan_data.issue_d.apply(lambda x:str(x).split('-')[0])
loan_data['issue_id_year']=loan_data.issue_d.apply(lambda x:int(str(x).split('-')[1]))
loan_data['issue_id_month_number']=loan_data.issue_id_month.apply(lambda x : month_num_d
```

In [42]:
```
loan_data.drop(columns=["issue_d"],inplace=True)
```

In [43]:
```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36800 entries, 0 to 39680
Data columns (total 32 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   loan_amnt              36800 non-null  int64  
 1   funded_amnt            36800 non-null  int64  
 2   funded_amnt_inv        36800 non-null  float64
 3   term                   36800 non-null  object 
 4   int_rate               36800 non-null  float64
 5   installment            36800 non-null  float64
 6   grade                  36800 non-null  object 
 7   sub_grade              36800 non-null  object 
 8   emp_length             36800 non-null  object 
 9   home_ownership         36800 non-null  object 
 10  annual_inc             36800 non-null  float64
 11  verification_status    36800 non-null  object 
 12  loan_status            36800 non-null  object 
 13  purpose                36800 non-null  object 
 14  zip_code               36800 non-null  object 
 15  addr_state             36800 non-null  object 
 16  dti                    36800 non-null  float64
 17  delinq_2yrs            36800 non-null  int64  
 18  earliest_cr_line       36800 non-null  object 
 19  inq_last_6mths         36800 non-null  int64  
 20  open_acc               36800 non-null  int64  
 21  pub_rec                36800 non-null  int64  
 22  revol_bal              36800 non-null  int64  
 23  revol_util             36800 non-null  object 
 24  total_acc              36800 non-null  int64  
 25  total_pymnt            36800 non-null  float64
 26  total_pymnt_inv        36800 non-null  float64
 27  last_pymnt_amnt        36800 non-null  float64
 28  pub_rec_bankruptcies   36800 non-null  float64
 29  issue_id_month         36800 non-null  object 
 30  issue_id_year          36800 non-null  int64  
 31  issue_id_month_number  36800 non-null  int64  
dtypes: float64(9), int64(10), object(13)
memory usage: 9.3+ MB
```

In [44]:
```
loan_data.earliest_cr_line.unique()
```

Out[44]:
```
array(['Jan-85', 'Apr-99', 'Nov-01', 'Feb-96', 'Nov-04', 'Jul-05',
       'Jan-07', 'Apr-04', 'Sep-04', 'Jan-98', 'Oct-89', 'Jul-03',
       'May-91', 'Sep-07', 'Oct-98', 'Aug-93', 'Oct-03', 'Jan-01',
       'Nov-97', 'Feb-83', 'Jul-85', 'Apr-03', 'Jun-01', 'Feb-02',
       'Aug-84', 'Nov-06', 'Dec-87', 'Nov-81', 'Apr-05', 'Oct-07',
       'Dec-00', 'Apr-07', 'Jan-03', 'Mar-94', 'Sep-98', 'Jun-04',
       'Nov-95', 'Jul-99', 'Jun-95', 'Sep-92', 'Jan-02', 'Apr-92',
       'Oct-06', 'May-00', 'Dec-98', 'Dec-04', 'Oct-00', 'May-02',
       'May-06', 'Jul-02', 'Jul-06', 'May-97', 'Oct-05', 'Apr-95',
```

```
'Oct-02', 'Jan-00', 'Apr-00', 'Dec-94', 'Sep-05', 'Dec-84',
'Dec-99', 'Nov-03', 'Jun-89', 'Jun-03', 'Oct-96', 'May-03',
'Jun-02', 'Jun-07', 'Dec-96', 'Sep-02', 'Jan-86', 'May-98',
'Jan-97', 'Jun-05', 'Feb-90', 'Mar-04', 'Jul-95', 'Aug-94',
'Jun-92', 'Mar-97', 'Apr-06', 'Apr-90', 'Aug-99', 'Sep-00',
'Feb-01', 'Dec-88', 'Feb-99', 'Dec-91', 'Aug-00', 'Oct-04',
'Aug-04', 'Feb-05', 'Nov-05', 'Nov-00', 'May-07', 'Jan-91',
'Jun-00', 'Aug-06', 'Dec-02', 'Jun-93', 'Jun-06', 'Feb-04',
'Dec-90', 'Mar-00', 'Feb-95', 'Jul-01', 'Apr-02', 'Dec-01',
'Sep-06', 'May-99', 'Aug-98', 'Dec-05', 'May-04', 'Oct-01',
'Jun-83', 'Mar-86', 'Apr-80', 'Jul-04', 'Jul-08', 'May-96',
'Jan-04', 'Nov-02', 'Aug-02', 'Aug-01', 'Mar-91', 'Sep-89',
'Sep-94', 'Sep-03', 'Sep-99', 'Aug-05', 'Dec-86', 'Nov-98',
'Feb-06', 'May-94', 'Nov-07', 'Feb-93', 'Nov-91', 'May-05',
'Mar-90', 'Mar-96', 'Oct-79', 'Jun-81', 'Mar-01', 'Apr-01',
'Jun-99', 'Nov-93', 'Jan-06', 'Dec-97', 'Nov-94', 'Jul-97',
'Oct-91', 'Jun-94', 'Mar-06', 'Sep-96', 'Apr-91', 'Jul-93',
'Jan-95', 'Sep-87', 'Mar-03', 'Oct-99', 'Jul-96', 'Dec-03',
'Aug-88', 'Mar-98', 'Feb-07', 'Dec-92', 'Jul-98', 'Jul-89',
'May-90', 'Jul-94', 'Sep-01', 'Mar-84', 'Nov-99', 'Mar-07',
'Mar-08', 'Apr-94', 'Jan-05', 'Jul-86', 'Aug-90', 'May-92',
'Jul-00', 'Mar-88', 'May-83', 'Jul-78', 'Mar-95', 'Feb-00',
'Mar-92', 'Jan-81', 'Sep-90', 'Apr-93', 'Jun-98', 'May-93',
'May-01', 'Nov-96', 'Feb-97', 'Jan-92', 'Mar-02', 'Jan-88',
'Aug-97', 'Aug-87', 'Aug-08', 'Oct-94', 'Feb-94', 'Jun-96',
'Feb-98', 'Nov-08', 'Apr-98', 'Jan-93', 'May-87', 'Jul-71',
'Aug-07', 'Jun-97', 'Mar-80', 'Dec-06', 'Jul-07', 'Oct-95',
'Jan-96', 'Jul-91', 'Jul-92', 'Dec-72', 'Dec-93', 'Jan-99',
'Feb-03', 'Apr-97', 'Dec-95', 'Mar-70', 'Nov-84', 'Apr-84',
'Jul-84', 'Aug-95', 'Mar-99', 'Sep-88', 'Mar-89', 'Mar-87',
'Oct-97', 'Dec-80', 'Jan-94', 'Jul-90', 'Aug-03', 'Mar-05',
'Jan-89', 'Apr-96', 'Oct-86', 'Feb-92', 'Jan-90', 'Nov-90',
'Mar-69', 'Jun-75', 'Mar-85', 'Dec-07', 'Sep-95', 'Oct-93',
'Dec-89', 'Sep-80', 'Jun-88', 'May-78', 'Aug-89', 'Oct-90',
'Sep-91', 'Feb-87', 'Nov-85', 'Jan-84', 'Jul-88', 'May-08',
'Oct-85', 'Mar-83', 'Aug-91', 'Jun-90', 'Feb-86', 'Jun-84',
'Sep-81', 'Apr-86', 'Aug-79', 'Aug-80', 'Nov-92', 'Sep-93',
'Jun-87', 'Feb-84', 'Aug-92', 'Aug-85', 'Jul-83', 'Dec-83',
'Jan-87', 'Aug-96', 'Sep-76', 'Nov-86', 'Oct-87', 'Sep-08',
'May-77', 'May-86', 'Mar-81', 'Jan-83', 'Sep-79', 'Oct-83',
'Nov-89', 'Jun-85', 'May-82', 'Feb-88', 'Oct-92', 'Aug-83',
'Sep-97', 'Apr-85', 'Oct-88', 'Oct-81', 'Sep-68', 'Jul-74',
'Jul-79', 'Nov-87', 'May-95', 'Feb-91', 'Mar-93', 'Jun-08',
'Jul-80', 'Dec-82', 'Oct-84', 'Feb-80', 'Nov-88', 'Apr-88',
'Sep-85', 'Sep-71', 'Mar-78', 'Feb-08', 'Jun-79', 'Jun-80',
'Apr-89', 'Sep-83', 'Feb-89', 'Aug-86', 'Oct-80', 'May-88',
'Dec-85', 'Jan-82', 'Sep-77', 'Sep-86', 'Dec-76', 'Apr-82',
'Apr-08', 'Feb-79', 'Jan-08', 'Jul-87', 'May-89', 'Oct-77',
'Dec-75', 'Oct-08', 'Feb-85', 'May-75', 'May-85', 'Feb-71',
'Jun-77', 'Dec-81', 'Apr-81', 'May-79', 'Feb-82', 'Jan-72',
'Jun-86', 'Sep-67', 'Apr-78', 'Feb-65', 'Nov-75', 'Jun-67',
'Dec-79', 'Aug-67', 'Apr-71', 'Sep-84', 'Aug-82', 'May-81',
'May-84', 'Dec-70', 'Oct-73', 'Jan-71', 'Apr-74', 'Jan-80',
'Apr-75', 'Mar-77', 'Nov-69', 'Jan-76', 'Nov-83', 'Apr-87',
'Nov-82', 'May-74', 'Aug-74', 'Jun-91', 'Jun-72', 'Mar-63',
'Aug-69', 'Jul-72', 'Aug-75', 'Sep-82', 'Sep-74', 'Aug-81',
'Nov-76', 'Mar-73', 'Dec-77', 'Oct-76', 'Jan-74', 'Jan-70',
'Aug-68', 'Apr-83', 'Oct-82', 'Jan-75', 'Dec-74', 'Feb-73',
'Jun-82', 'Jun-74', 'May-65', 'Apr-76', 'Oct-71', 'Apr-77',
'Oct-78', 'Feb-81', 'Jan-77', 'Aug-77', 'Dec-78', 'Aug-76',
'Jun-68', 'Jan-78', 'Dec-73', 'Sep-78', 'Nov-70', 'Nov-78',
'May-80', 'Jan-79', 'Oct-65', 'Nov-74', 'Apr-66', 'Feb-72',
'Mar-76', 'Sep-73', 'Aug-78', 'Mar-79', 'Jul-76', 'Jul-82',
'Apr-73', 'Apr-67', 'Oct-72', 'Mar-75', 'Mar-82', 'Oct-63',
'Feb-70', 'Jul-73', 'Feb-78', 'Nov-71', 'Jun-76', 'Aug-72',
'Jul-77', 'Jul-75', 'Sep-70', 'Jul-81', 'Sep-72', 'Nov-80',
```

```
'Sep-62', 'Apr-70', 'Nov-77', 'Nov-66', 'Jun-78', 'May-71',
'May-70', 'Apr-79', 'May-73', 'Oct-70', 'Feb-75', 'Mar-74',
'Sep-56', 'Jan-46', 'Dec-50', 'Mar-66', 'Jul-69', 'Jan-68',
'Nov-73', 'Jun-70', 'Feb-77', 'Feb-68', 'Feb-74', 'Jun-69',
'Feb-66', 'Sep-64', 'May-76', 'Aug-73', 'Aug-70', 'Jun-73',
'Dec-68', 'Feb-76', 'Sep-75', 'Sep-69', 'Nov-54', 'Mar-72',
'Jan-73', 'Nov-79', 'Dec-65', 'Apr-72', 'Nov-72', 'Nov-67',
'Sep-63', 'Dec-69', 'Apr-69', 'Nov-62', 'Jul-70', 'Jan-63',
'Oct-67', 'May-67', 'Feb-67', 'Jun-71', 'Nov-68', 'Oct-75',
'Mar-71', 'Apr-64', 'Feb-69', 'Aug-71', 'Jul-67', 'Dec-66',
'Oct-68', 'Oct-74', 'May-72'], dtype=object)
```

As we can see above, the column "earliest_cr_line" which is the combination of month and year can be split into two separate columns for date and month.
And the original column can be removed.

In [45]:
```
loan_data['earliest_cr_line_month']=loan_data.earliest_cr_line.apply(lambda x:str(x).spl
loan_data['earliest_cr_line_year']=loan_data.earliest_cr_line.apply(lambda x:int(str(x).
```

In [46]:
```
loan_data.drop(columns=["earliest_cr_line"],inplace=True)
```

In [47]:
```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36800 entries, 0 to 39680
Data columns (total 33 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   loan_amnt               36800 non-null  int64  
 1   funded_amnt             36800 non-null  int64  
 2   funded_amnt_inv         36800 non-null  float64
 3   term                    36800 non-null  object 
 4   int_rate                36800 non-null  float64
 5   installment             36800 non-null  float64
 6   grade                   36800 non-null  object 
 7   sub_grade               36800 non-null  object 
 8   emp_length              36800 non-null  object 
 9   home_ownership          36800 non-null  object 
 10  annual_inc              36800 non-null  float64
 11  verification_status     36800 non-null  object 
 12  loan_status             36800 non-null  object 
 13  purpose                 36800 non-null  object 
 14  zip_code                36800 non-null  object 
 15  addr_state              36800 non-null  object 
 16  dti                     36800 non-null  float64
 17  delinq_2yrs             36800 non-null  int64  
 18  inq_last_6mths          36800 non-null  int64  
 19  open_acc                36800 non-null  int64  
 20  pub_rec                 36800 non-null  int64  
 21  revol_bal               36800 non-null  int64  
 22  revol_util              36800 non-null  object 
 23  total_acc               36800 non-null  int64  
 24  total_pymnt             36800 non-null  float64
 25  total_pymnt_inv         36800 non-null  float64
 26  last_pymnt_amnt         36800 non-null  float64
 27  pub_rec_bankruptcies    36800 non-null  float64
 28  issue_id_month          36800 non-null  object 
 29  issue_id_year           36800 non-null  int64  
 30  issue_id_month_number   36800 non-null  int64  
 31  earliest_cr_line_month  36800 non-null  object 
 32  earliest_cr_line_year   36800 non-null  int64  
dtypes: float64(9), int64(11), object(13)
memory usage: 9.5+ MB
```

In [48]:
```
loan_data.revol_util.unique()
```
Out[48]:
```
array(['83.70%', '9.40%', '98.50%', ..., '49.63%', '0.04%', '7.28%'],
      dtype=object)
```

As we can see above, the revol_util column is in the form of object appended with '%', we can remove it and convert to float, so that it can be used for our analysis.

In [49]:
```
loan_data.revol_util=loan_data.revol_util.apply(lambda x:float(str(x).rstrip('%')))
```

In [50]:
```
loan_data.revol_util.unique()
```
Out[50]:
```
array([8.370e+01, 9.400e+00, 9.850e+01, ..., 4.963e+01, 4.000e-02,
       7.280e+00])
```

In [51]:
```
loan_data.emp_length.unique()
```
Out[51]:
```
array(['10+ years', '< 1 year', '3 years', '8 years', '9 years',
       '4 years', '5 years', '1 year', '6 years', '2 years', '7 years'],
      dtype=object)
```

As we see above, emp_length does not contain data in numeric format, we can convert it to numeric format so that it can be helpful for us in further analysis.

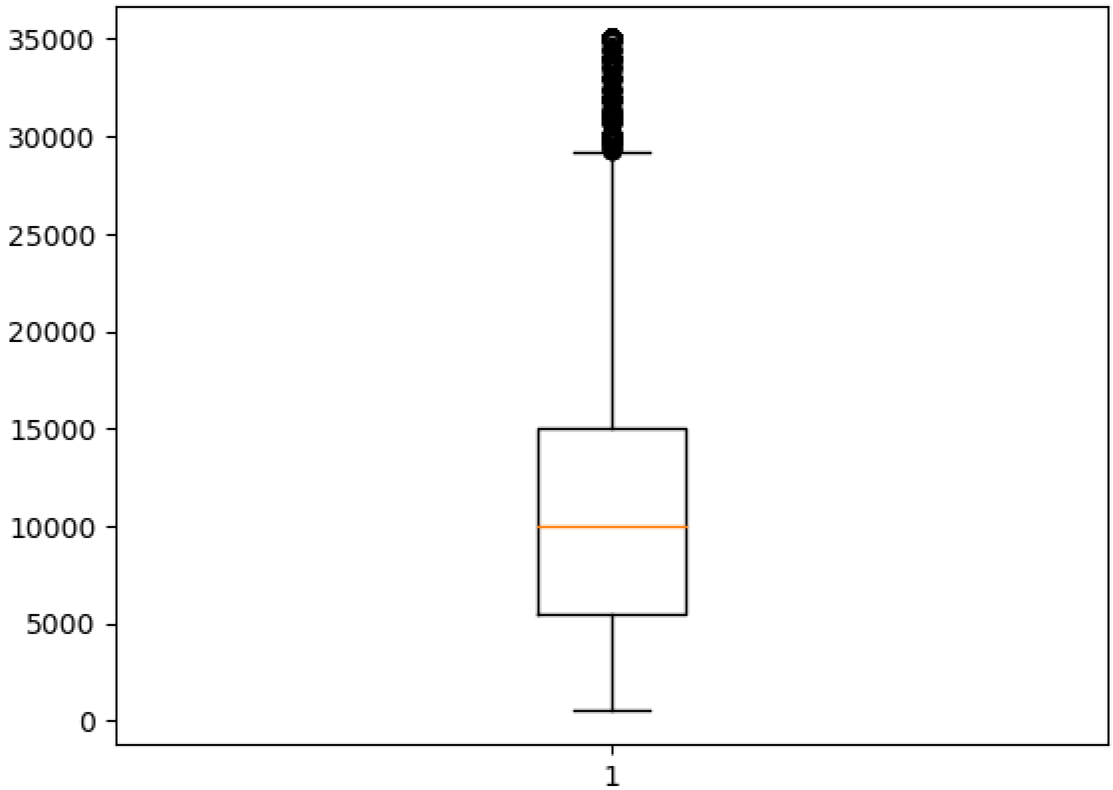
In [52]:
```
import re
loan_data.emp_length=loan_data.emp_length.apply(lambda x:str(x).replace("< 1 year","0"))
loan_data.emp_length=loan_data.emp_length.apply(lambda x:int(re.findall(r'\d+', str(x))[
```

In [53]:
```
loan_data.emp_length.unique()
```
Out[53]:
```
array([10,  0,  3,  8,  9,  4,  5,  1,  6,  2,  7], dtype=int64)
```

Let us now detect outliers.

Let us consider loan_amnt first.

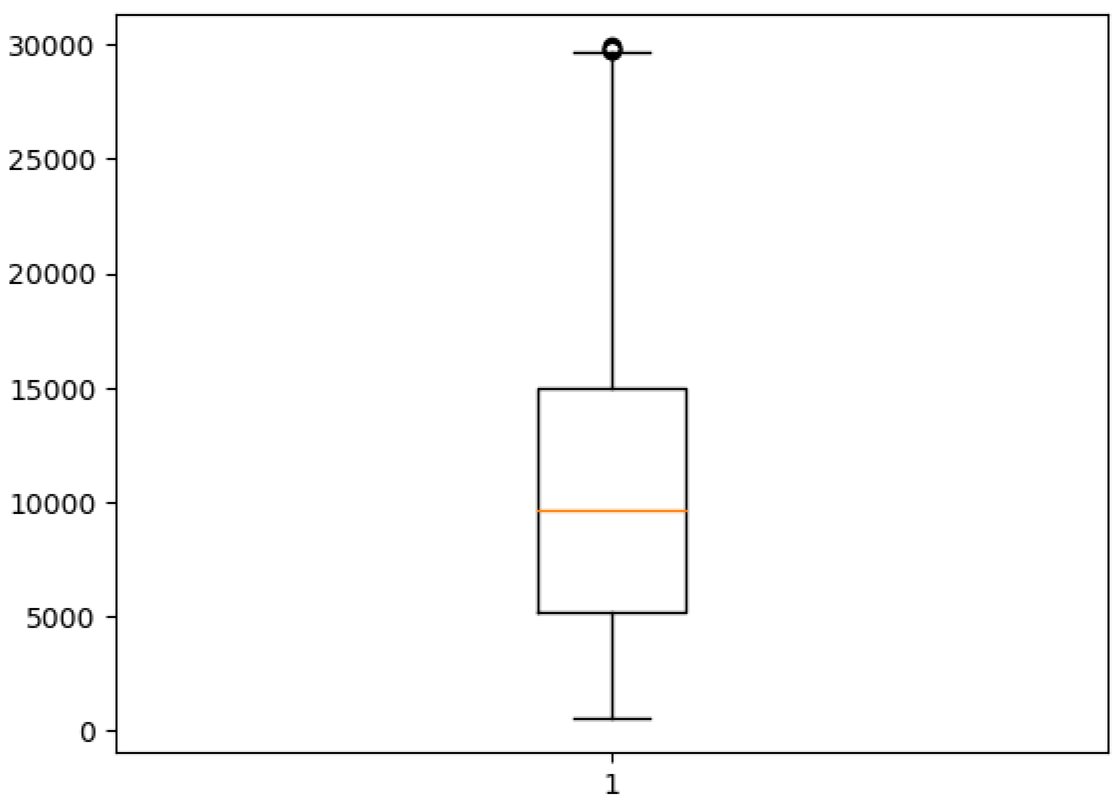
In [54]:
```
plt.boxplot(loan_data.loan_amnt)
plt.show()
```



From the above boxplot, we can see there are some outliers above 30000. Let us remove them.

In [55]:
```
loan_data=loan_data[loan_data.loan_amnt<30000]
```

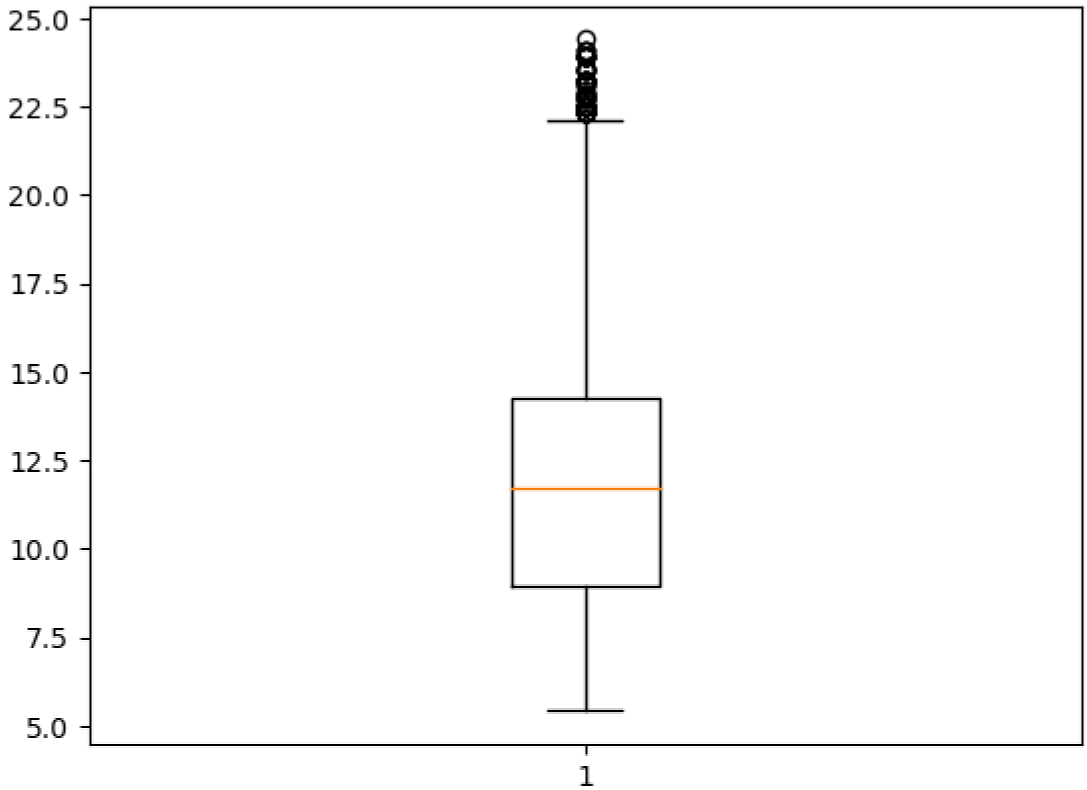
In [56]:
```
plt.boxplot(loan_data.loan_amnt)
plt.show()
```



As we can see outliers are removed for loan_amnt column.

Let us consider int_rate column for detecting if there are any outliers in it.

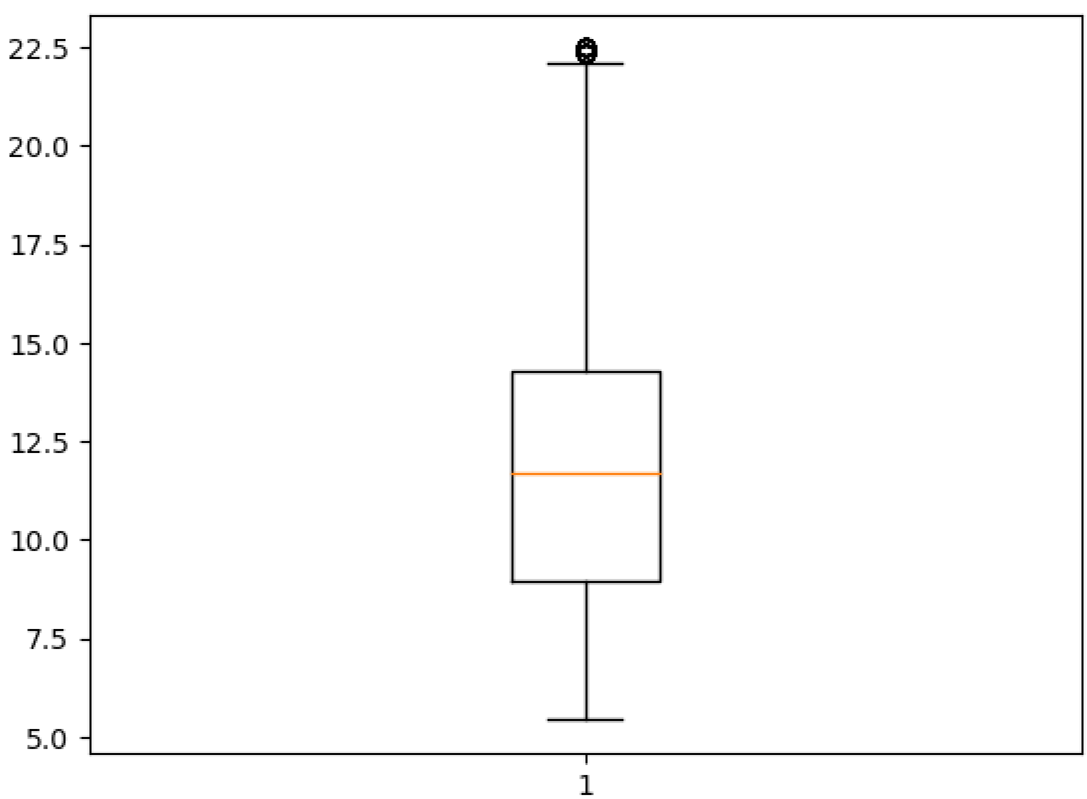
In [57]:
```
plt.boxplot(loan_data.int_rate)
plt.show()
```



From the above boxplot, we can see there are outliers. Let us remove them.

In [58]:
```
loan_data=loan_data[loan_data.int_rate<22.5]
```

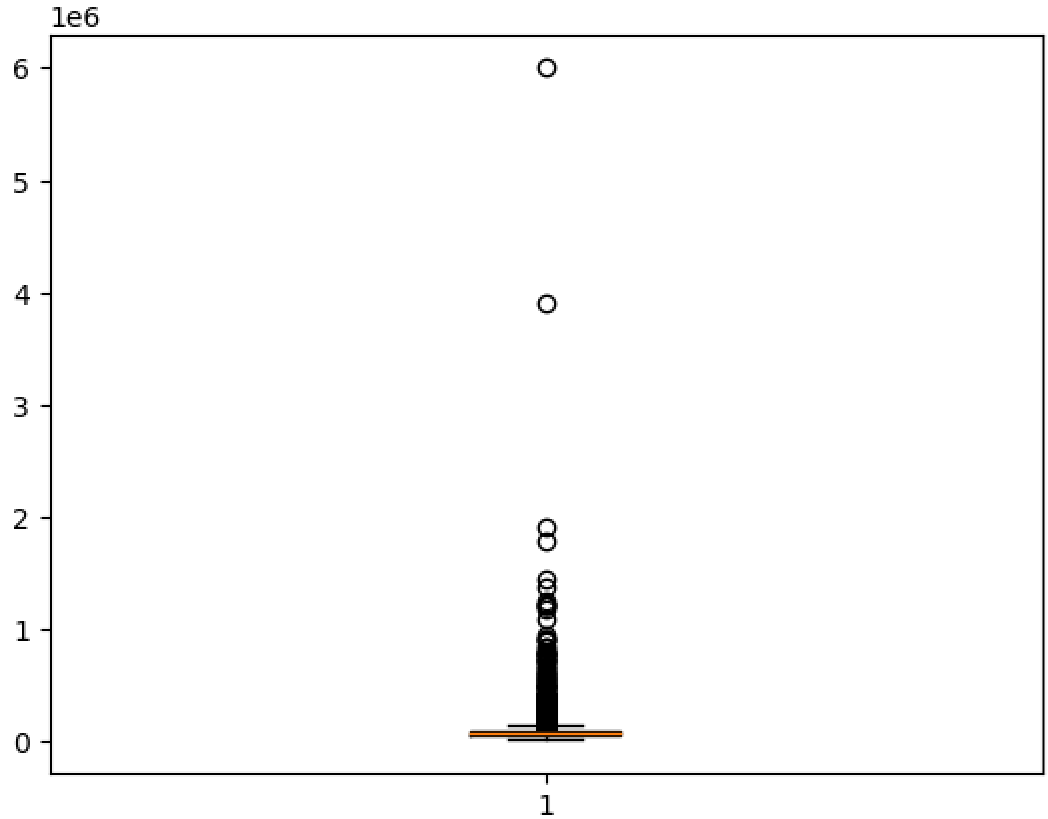
In [59]:
```
plt.boxplot(loan_data.int_rate)
plt.show()
```



We can see the outliers are removed now for int_rate.

Let us consider annual_inc column for detecting if there are any outliers in it.

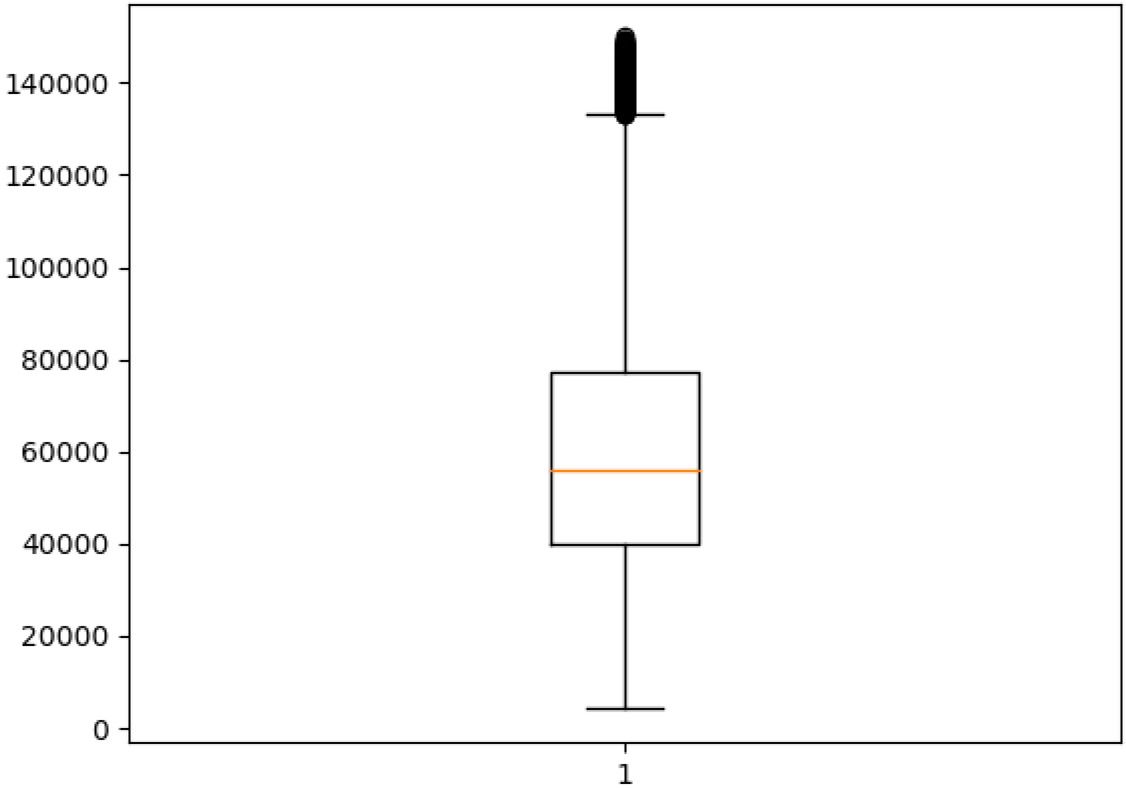
In [60]:
```
plt.boxplot(loan_data.annual_inc)
plt.show()
```



As we can clearly see there are outliers, let us remove outliers now.

In [61]:
```
loan_data = loan_data[loan_data.annual_inc<150000]
```

In [62]:
```
plt.boxplot(loan_data.annual_inc)
plt.show()
```



As we can see from the above boxplot, the outliers are removed.

In [63]:
```
pd.set_option('display.max_columns', None)
loan_data.head()
```

Out[63]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_length	hom
0	5000	5000	4975.0	36 months	10.65	162.87	B	B2	10	
1	2500	2500	2500.0	60 months	15.27	59.83	C	C4	0	
2	2400	2400	2400.0	36 months	15.96	84.33	C	C5	10	
3	10000	10000	10000.0	36 months	13.49	339.31	C	C1	10	
5	5000	5000	5000.0	36 months	7.90	156.46	A	A4	3	

In [64]:
```
loan_data.shape
```

Out[64]:
```
(34295, 33)
```

3. Univariate analysis

Univariate analysis is the process of analysing one variable at a time. This can be done in following steps.

- Analysing unordered categorical variables
- Analysing ordered categorical variables
- Analysing quantitative variables
- Segmented univariate analysis

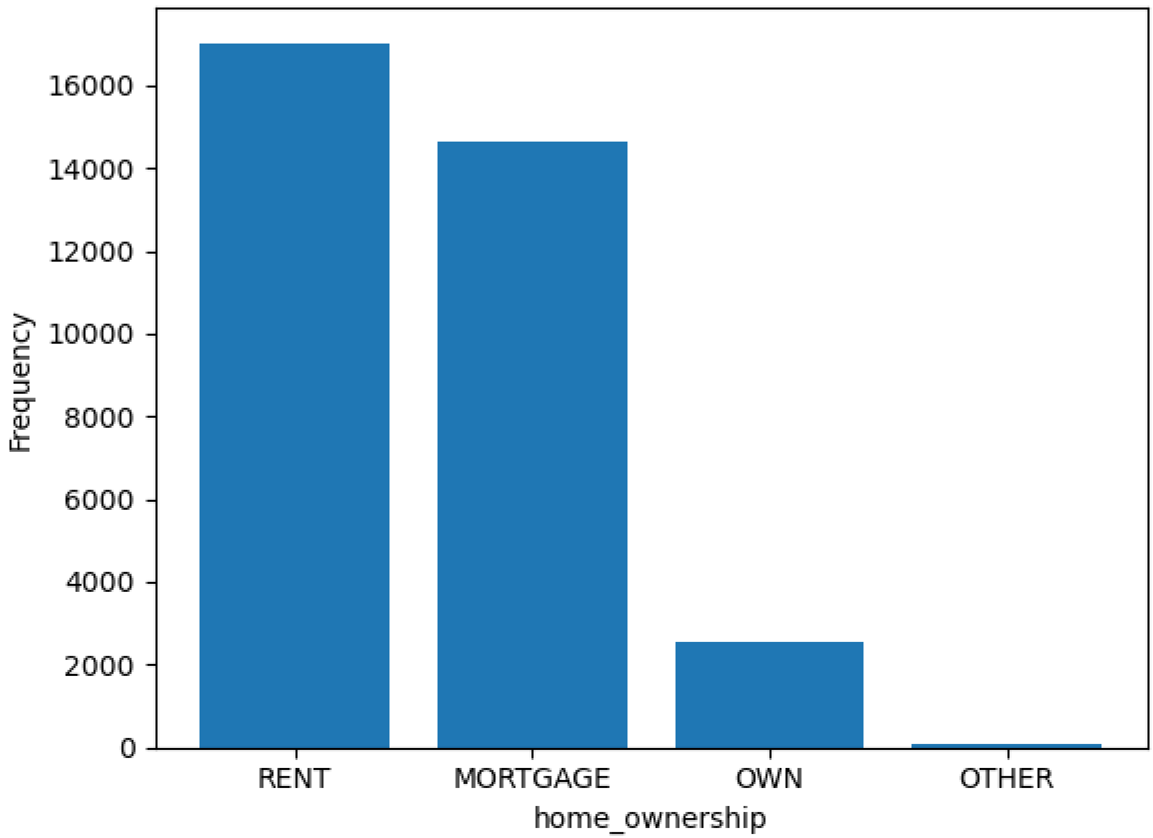
Analysing unordered categorical variables

Let the now analyse the unordered categorical variable - home_ownership.

In [65]:
```
ua_ucv_home_ownership = loan_data.groupby(by="home_ownership").size()
```

In [66]:
```
ua_ucv_home_ownership.sort_values(ascending=False,inplace=True)
```

In [67]:
```
plt.bar(ua_ucv_home_ownership.index,ua_ucv_home_ownership.values)
plt.ylabel("Frequency")
plt.xlabel("home_ownership")
plt.show()
```



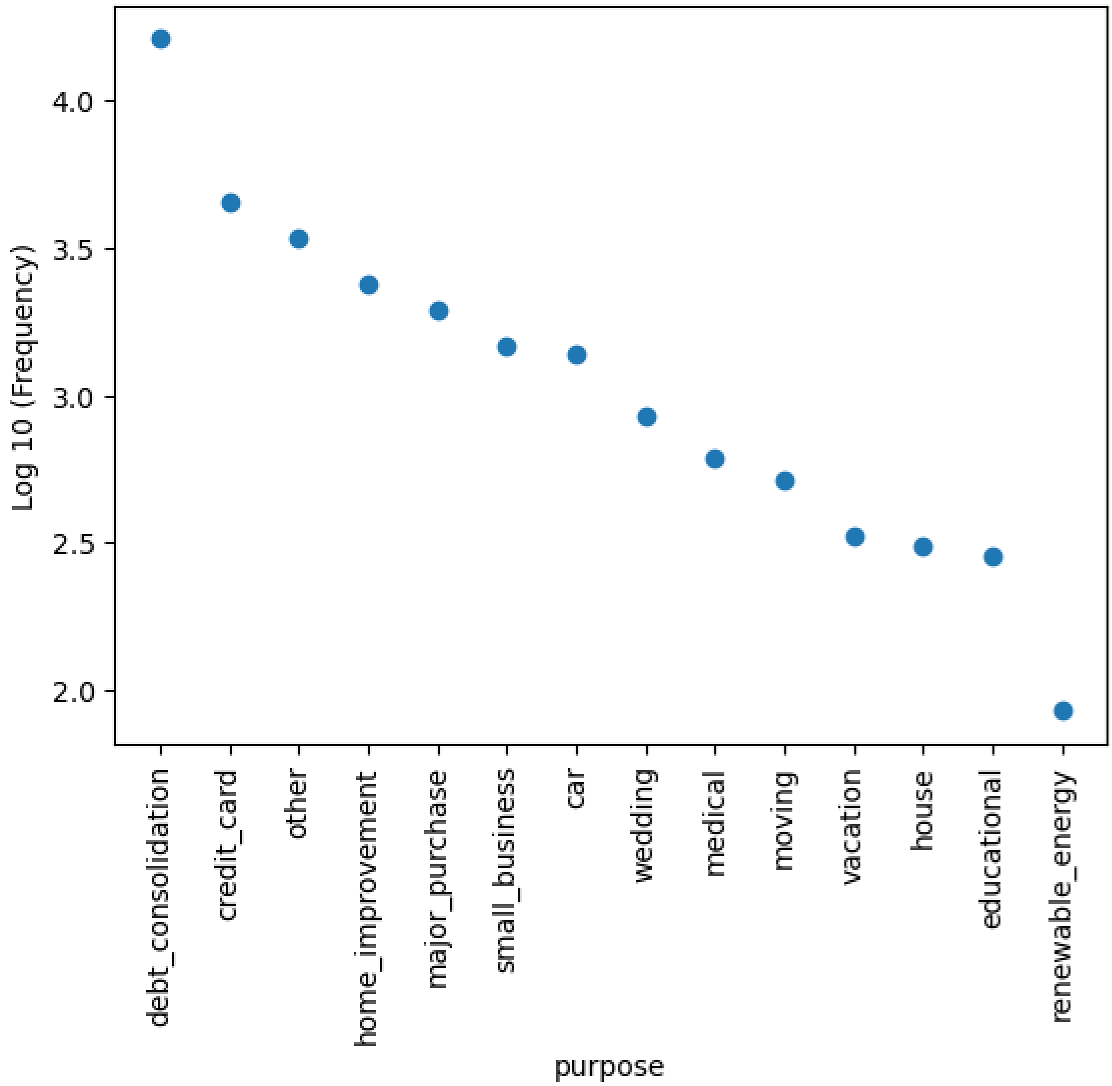
From the above scatter plot, we can observe the loan applicants are more for home_ownership of "RENT" and "MORTGAGE".

Let the now analyse the another unordered categorical variable - purpose

In [68]:
```
ua_ucv_purpose = loan_data.groupby(by="purpose").size()
```

In [69]:
```
ua_ucv_purpose.sort_values(ascending=False,inplace=True)
```

In [70]:
```
plt.scatter(ua_ucv_purpose.index,np.log10(ua_ucv_purpose.values))
plt.ylabel("Log 10 (Frequency)")
plt.xlabel("purpose")
plt.xticks(rotation=90)
plt.show()
```



We can see the "purpose" column follows **"Power law distribution"**.
We can see the loan applicants are more for purpose "debt_consolidation" followed by "credit_card" and "other".

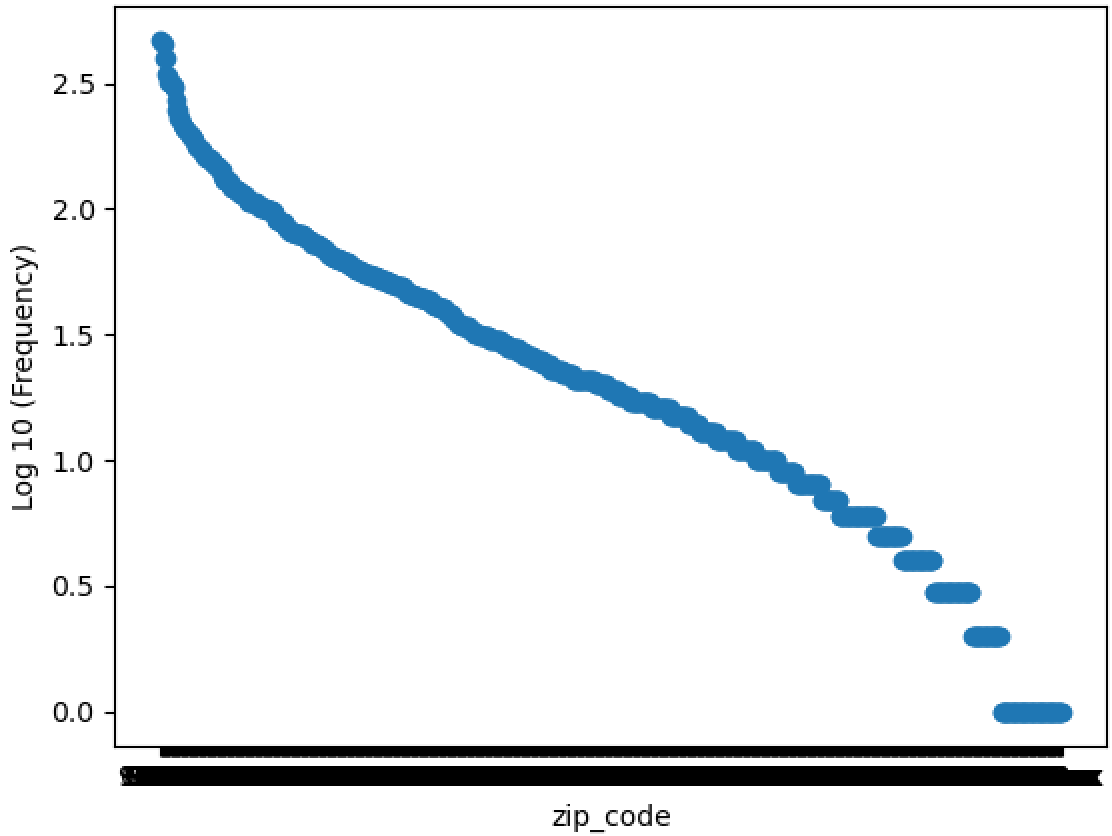
Let the now analyse the another unordered categorical variable - zip_code.

In [71]:
```
ua_ucv_zip_code = loan_data.groupby(by="zip_code").size()
```

In [72]:
```
ua_ucv_zip_code.sort_values(ascending=False,inplace=True)
```

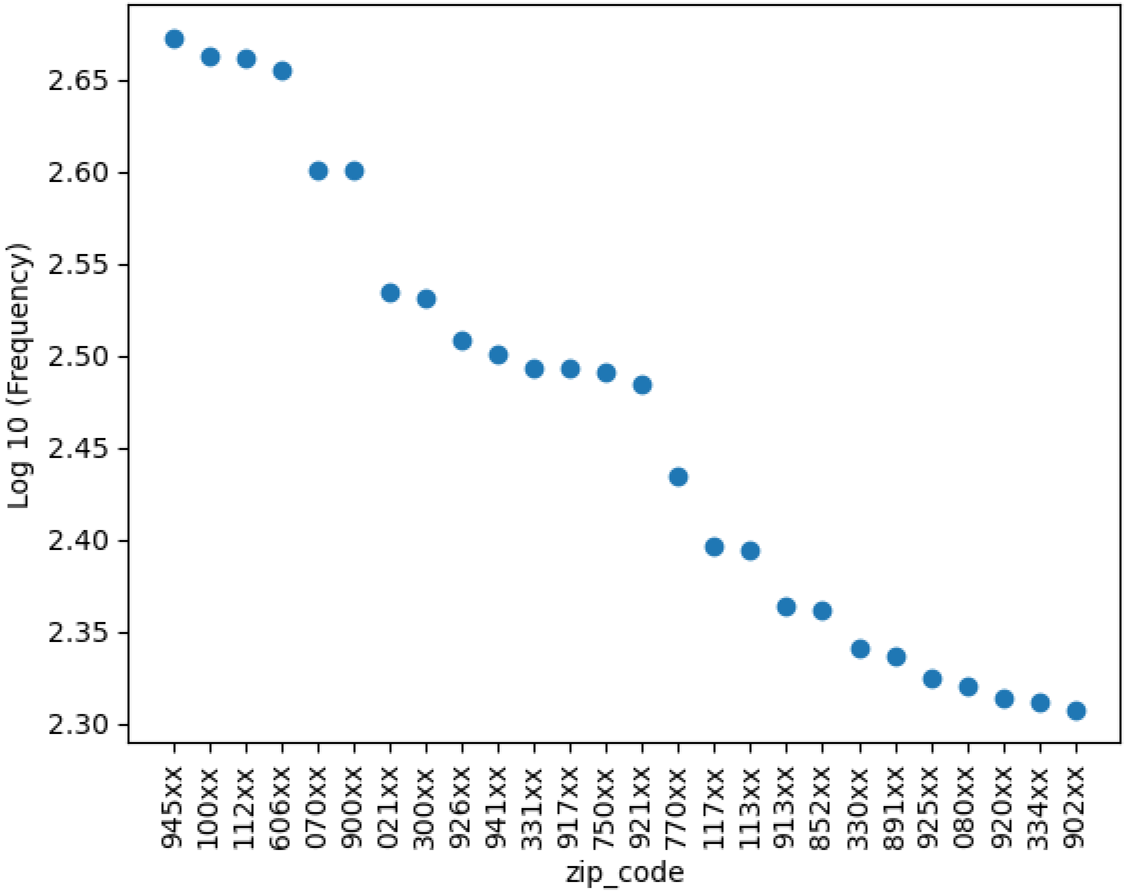
In [73]:
```
plt.scatter(ua_ucv_zip_code.index,np.log10(ua_ucv_zip_code.values))
plt.ylabel("Log 10 (Frequency)")
plt.xlabel("zip_code")

plt.show()
```



The above plot says zip_code also follows **"Power law distribution"**. Let us plot it by not considering the minimum frequency ones.

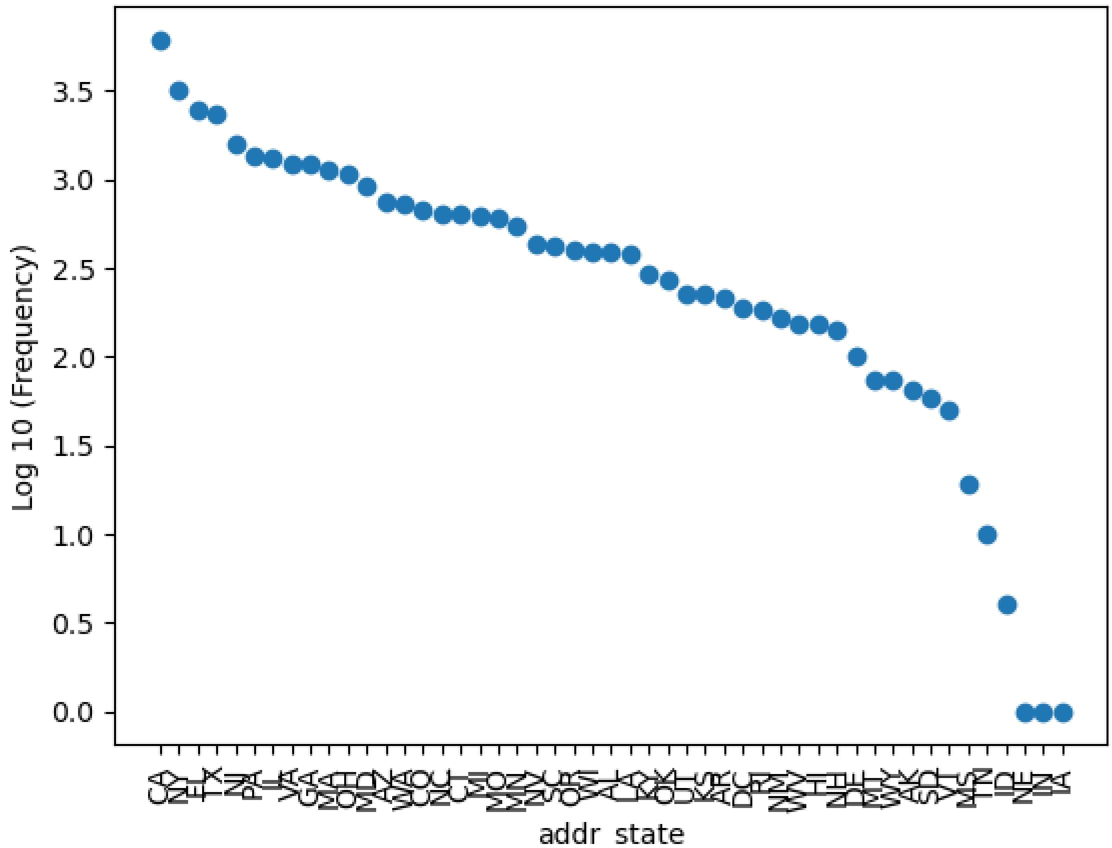
In [74]:
```
ua_ucv_zip_code = ua_ucv_zip_code[ua_ucv_zip_code>200]
ua_ucv_zip_code.sort_values(ascending=False,inplace=True)
plt.scatter(ua_ucv_zip_code.index,np.log10(ua_ucv_zip_code.values))
plt.ylabel("Log 10 (Frequency)")
plt.xlabel("zip_code")
plt.xticks(rotation=90)
plt.show()
```



From the above graph we can conclude, the loan applicants are high for zip_code starting with 945,100,112 and 606.

Let the now analyse the another unordered categorical variable - addr_state.

In [75]:
```
ua_ucv_addr_state = loan_data.groupby(by="addr_state").size()
ua_ucv_addr_state.sort_values(ascending=False,inplace=True)
plt.scatter(ua_ucv_addr_state.index,np.log10(ua_ucv_addr_state.values))
plt.ylabel("Log 10 (Frequency)")
plt.xlabel("addr_state")
plt.xticks(rotation=90)
plt.show()
```

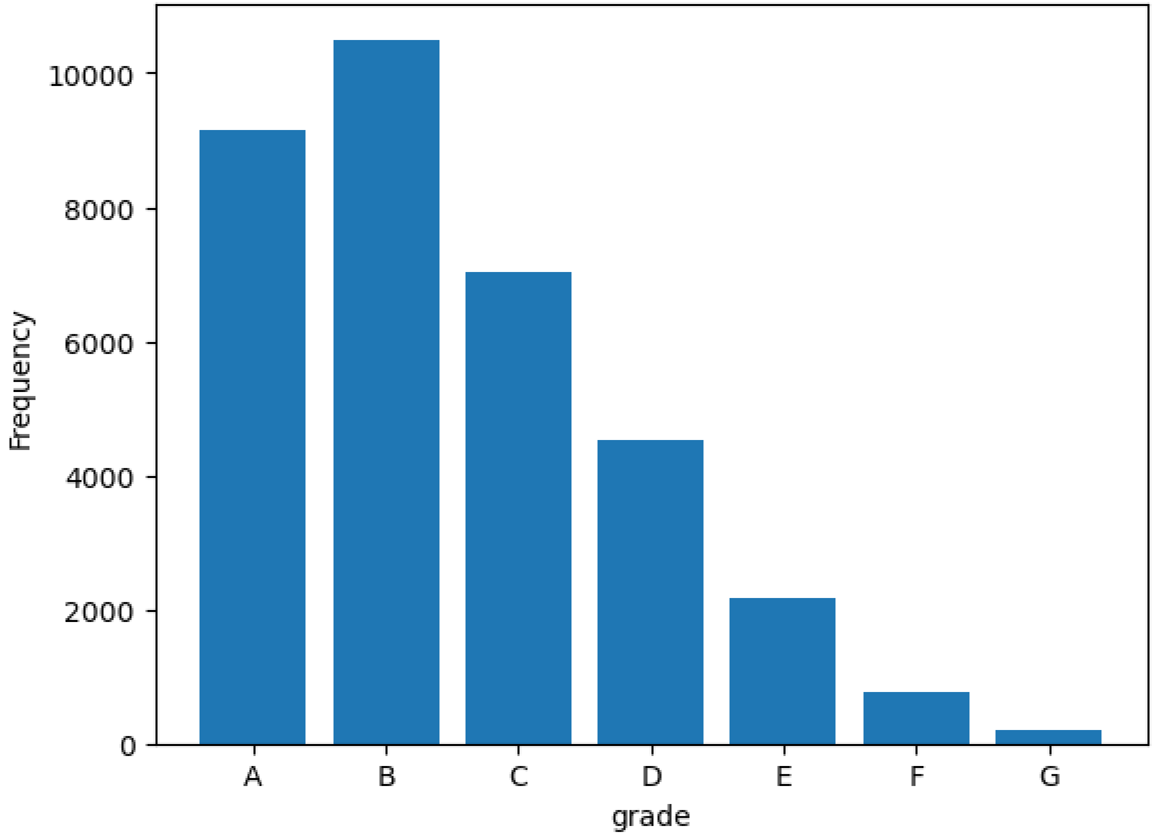


We can see the "addr_state" column follows **"Power law distribution"**. We can see the loan applicants are more for addr_state "CA" followed by "NY" and "FL"

Analysing ordered categorical variables

Let the now analyse the ordered categorical variable - grade.

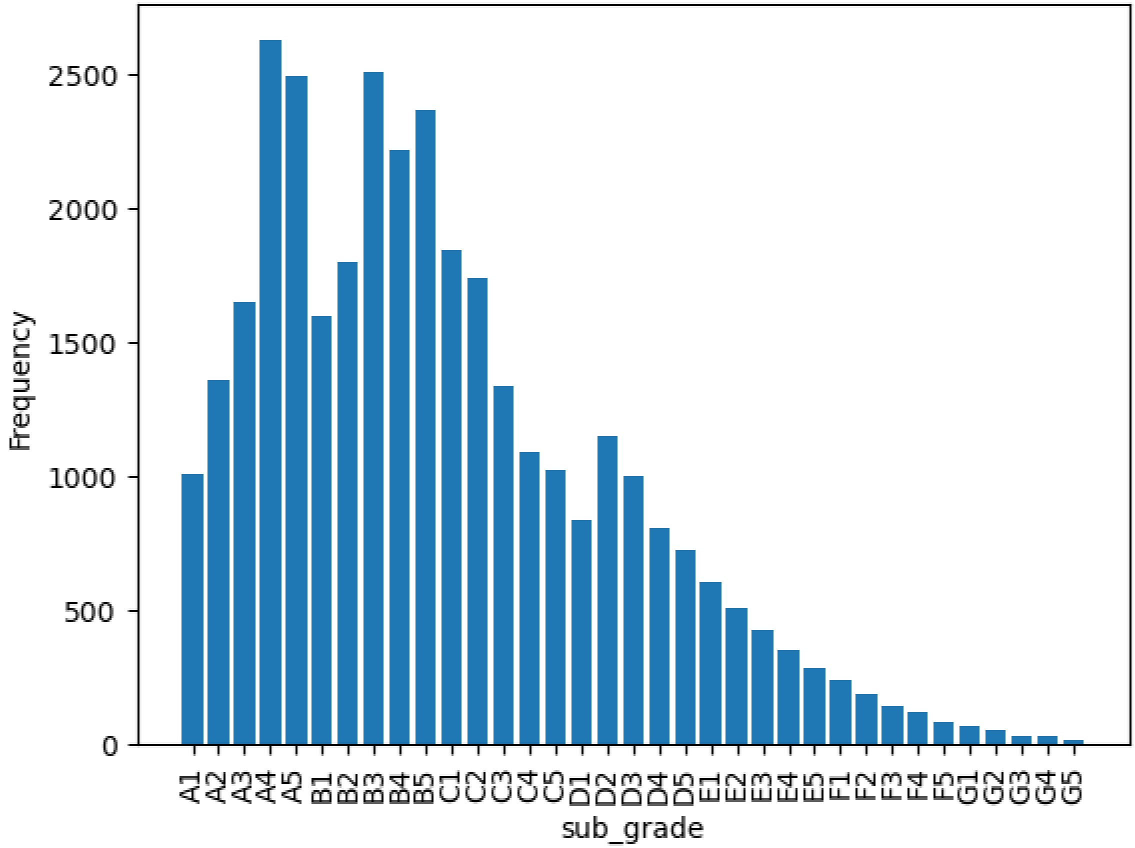
In [76]:
```
ua_ocv_grade = loan_data.groupby(by="grade").size()
plt.bar(ua_ocv_grade.index,ua_ocv_grade.values)
plt.ylabel("Frequency")
plt.xlabel("grade")
plt.show()
```



From the above plot, we can conclude most of the loan applicants are of grades A and B. Also, there are very few applicants with grade G.
Also we can observe as the grade increases, the frequency of loan applicants decreasing.

Let the now analyse the ordered categorical variable - sub_grade.

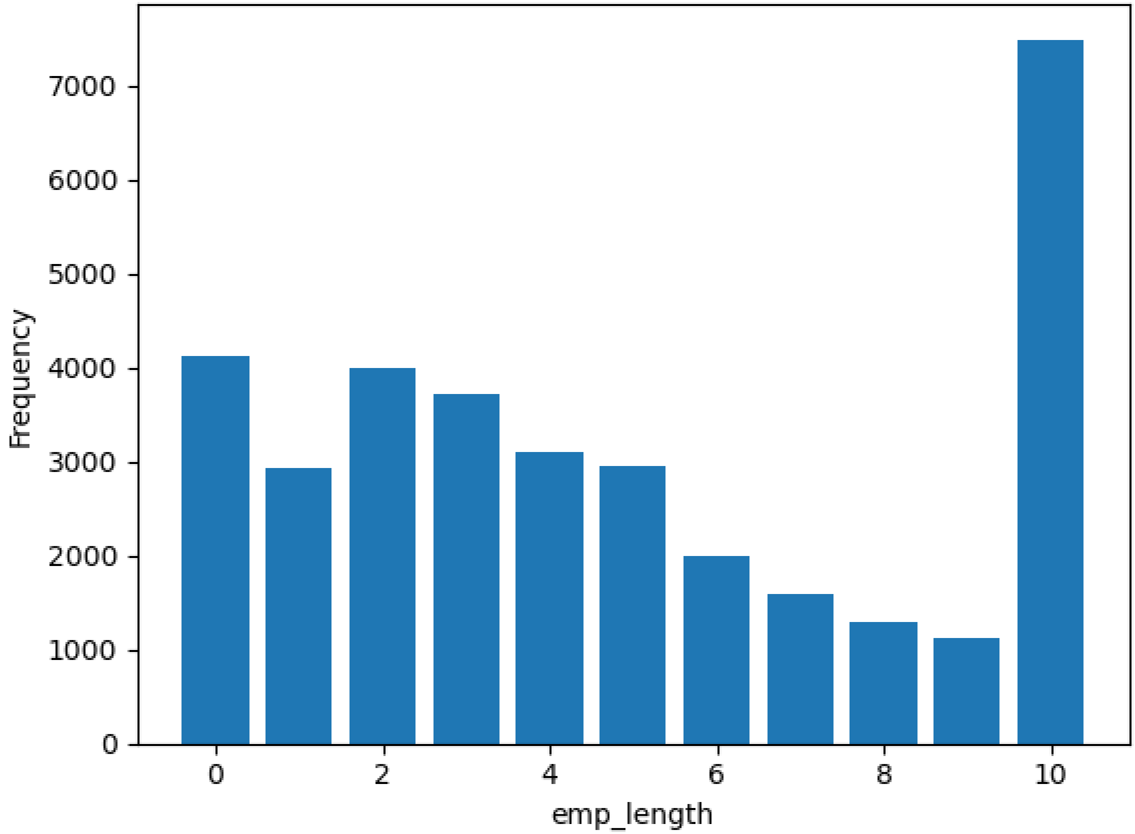
In [77]:
```
ua_ocv_sub_grade = loan_data.groupby(by="sub_grade").size()
plt.bar(ua_ocv_sub_grade.index,ua_ocv_sub_grade.values)
plt.ylabel("Frequency")
plt.xlabel("sub_grade")
plt.xticks(rotation=90)
plt.show()
```



From the above plot, we can conclude most of the loan applicants are of sub_grades belonging to A and B. Also, there are very few applicants with sub_grade belonging to G.

Let the now analyse the ordered categorical variable - emp_length.

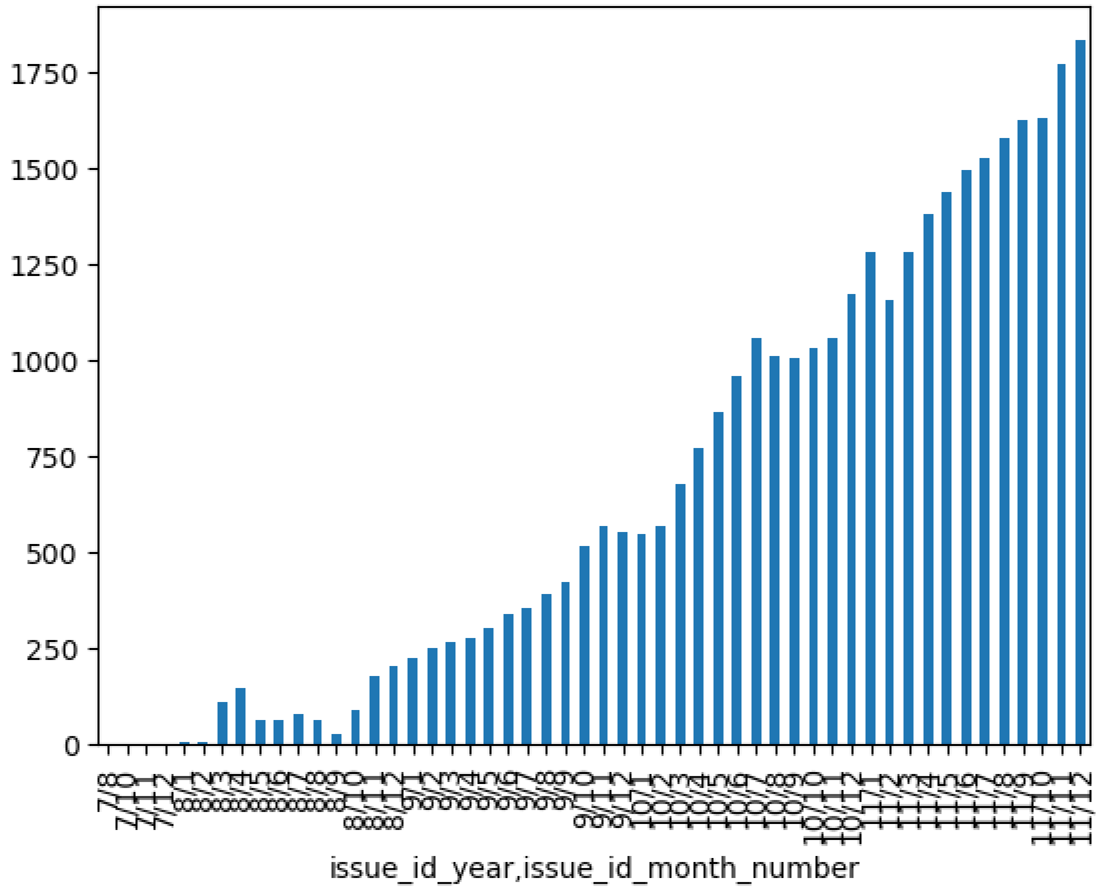
In [78]:
```
ua_ocv_emp_length = loan_data.groupby(by="emp_length").size()
ua_ocv_emp_length.sort_index()
plt.bar(ua_ocv_emp_length.index,ua_ocv_emp_length.values)
plt.ylabel("Frequency")
plt.xlabel("emp_length")
plt.show()
```



From the above plot, we can see the loan applicants are more for 10+ years of emp_length.

In [79]:
```
ua_ocv_issue_d = loan_data.groupby(by=['issue_id_year',"issue_id_month_number"]).size()
ax=ua_ocv_issue_d.plot.bar()
ax.set_xticklabels([f"{x}/{y}" for x,y in ua_ocv_issue_d.index.tolist()])
ax.plot()
```

Out[79]: []

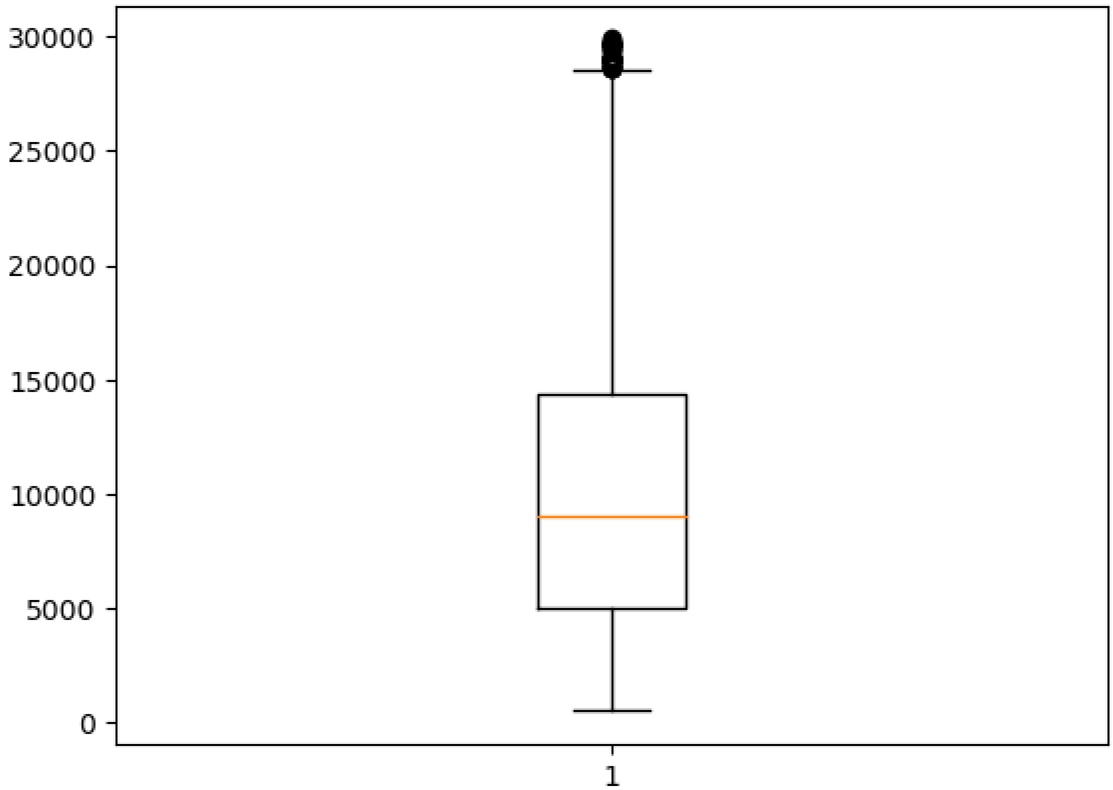


From the above plot, we can see the number of loan applications has been increasing over the time.

Analysing quantitative variables

Let us analyse the quantitative variable - "loan_amnt"

In [80]:
```
plt.boxplot(loan_data.loan_amnt)
plt.show()
```



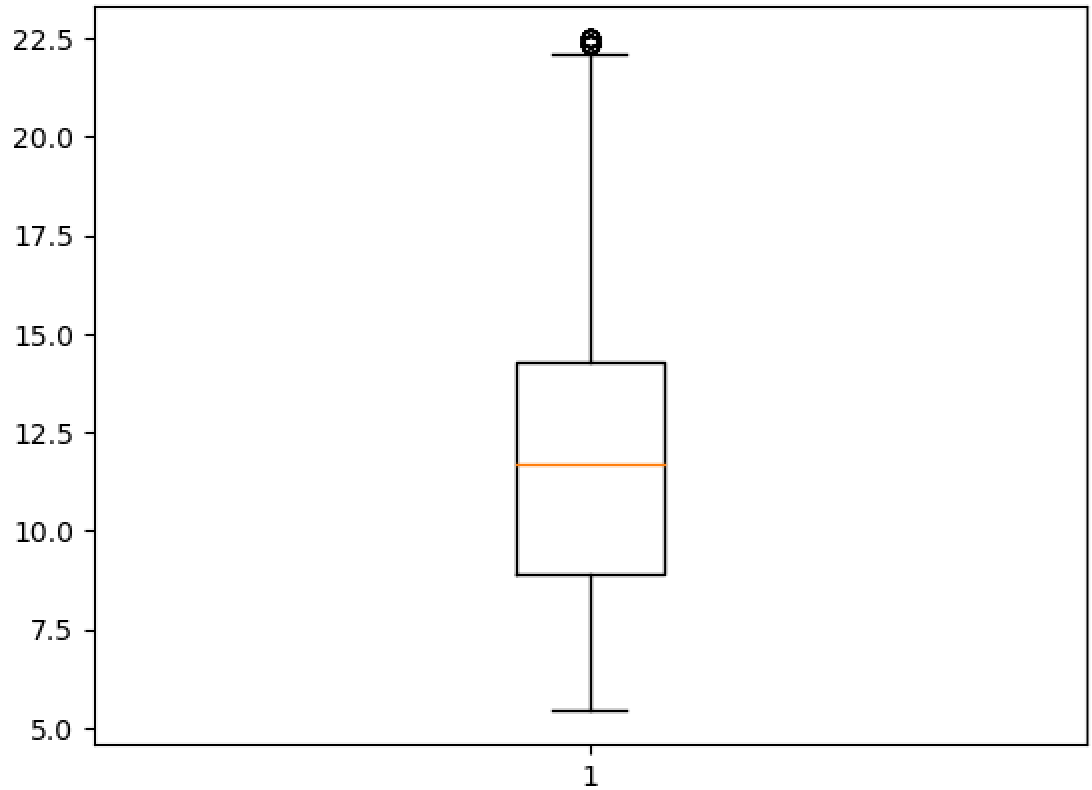
From the above graph, we can see the the loan_amnt is spread widely from 50th percentile to 100th percentile.
Most of the loan applicants are from loan_amnt with below 15000.

In [81]:
```
loan_data.loan_amnt.describe()
```

Out[81]:
```
count    34295.000000
mean     10273.626622
std       6276.333232
min        500.000000
25%       5000.000000
50%       9000.000000
75%      14400.000000
max      29900.000000
Name: loan_amnt, dtype: float64
```

Let us now analyse another quantitative variable - "int_rate".

In [82]:
```
plt.boxplot(loan_data.int_rate)
plt.show()
```



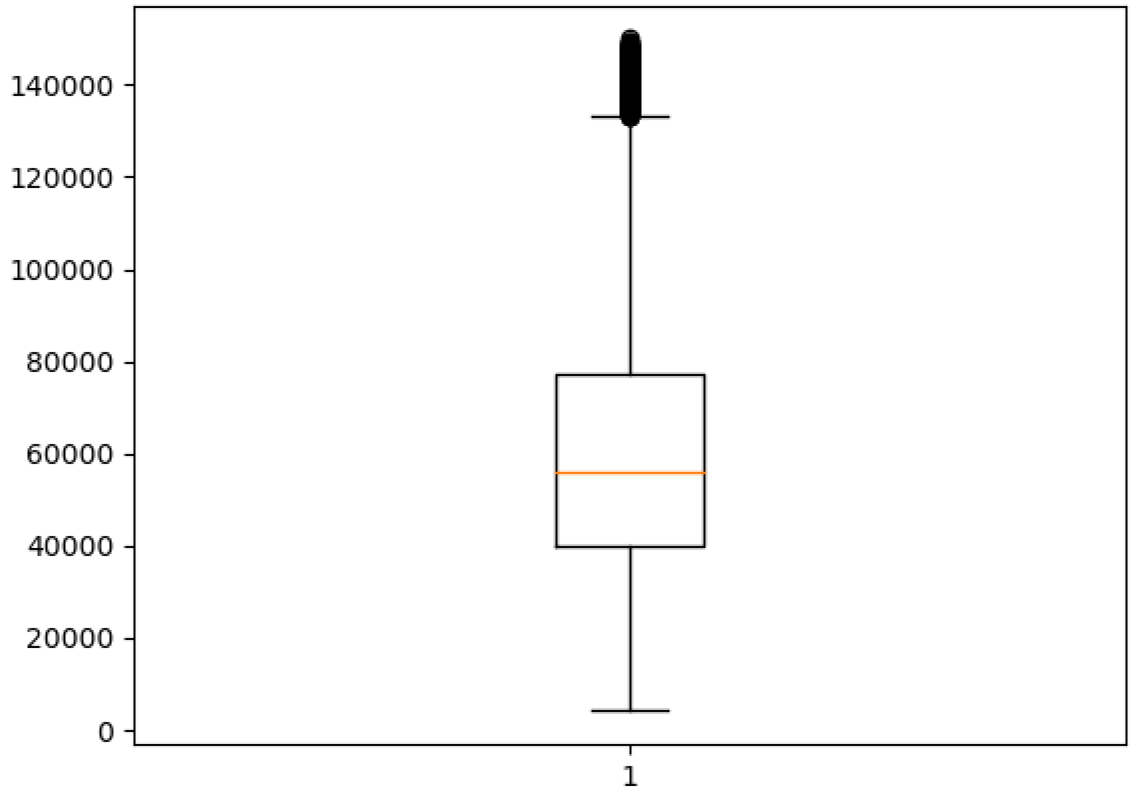
From the above graph, we can see the the int_rate is spread widely from 50th percentile to 100th percentile. Let us describe it.

In [83]:
```
loan_data.int_rate.describe()
```

Out[83]:
```
count    34295.000000
mean        11.836186
std          3.603375
min          5.420000
25%          8.900000
50%         11.710000
75%         14.270000
max         22.480000
Name: int_rate, dtype: float64
```

The average int_rate given to the applicants is 11.83%.

In [84]:
```
plt.boxplot(loan_data.annual_inc)
plt.show()
```



From the above graph, we can see the the annual_inc is spread widely from 50th percentile to 100th percentile. Let us describe it.

In [85]:
```
loan_data.annual_inc.describe()
```

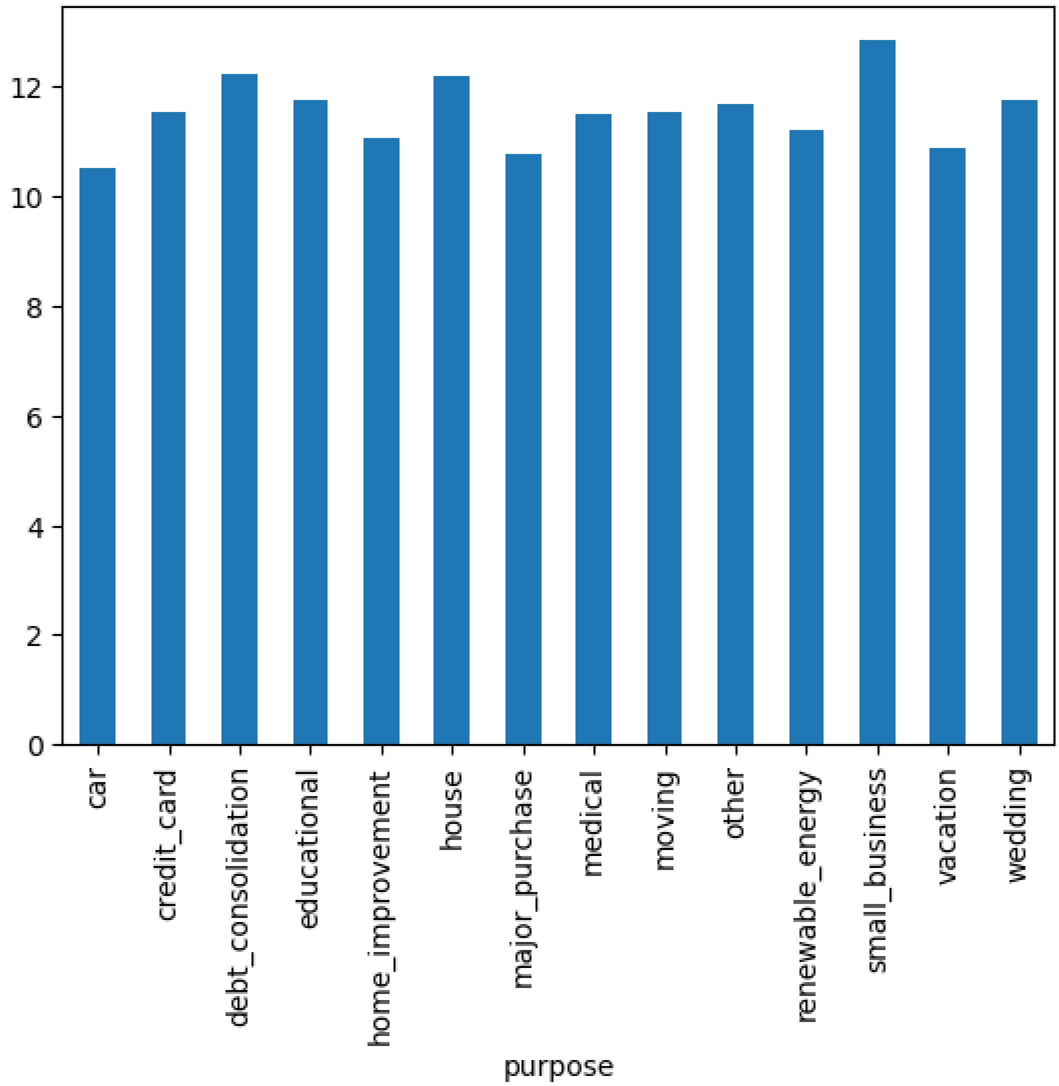
Out[85]:
```
count     34295.000000
mean      61186.056529
std       27911.822309
min        4000.000000
25%       40000.000000
50%       56000.000000
75%       77290.000000
max      149981.000000
Name: annual_inc, dtype: float64
```

Segmented univariate analysis

Let us check how int_rate is distributed on "purpose".

In [86]:
```
loan_data.groupby(by="purpose")["int_rate"].mean().plot.bar()
```

Out[86]:
```
<AxesSubplot:xlabel='purpose'>
```

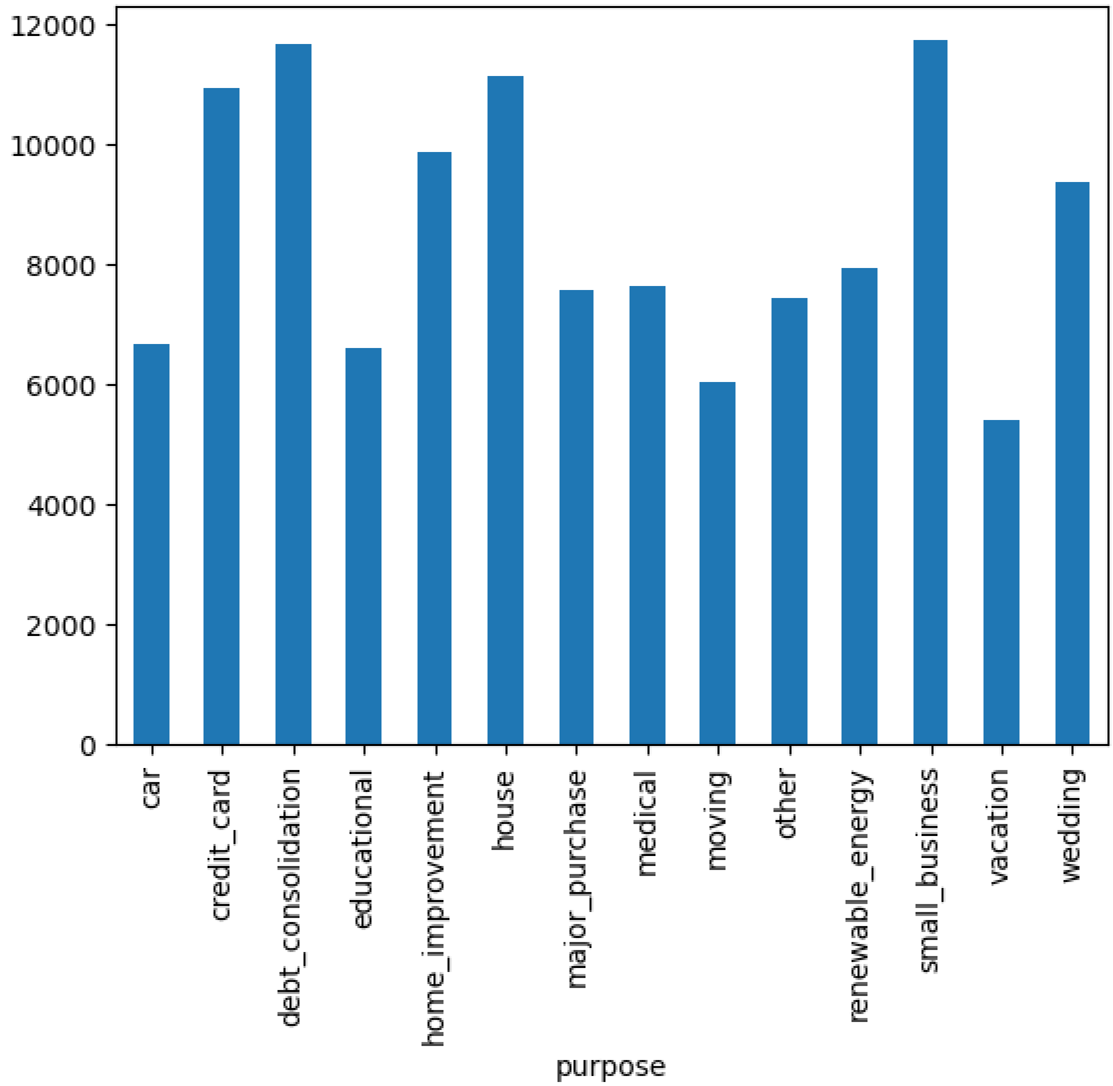


As we can see, interest rate is high for purposes of debt_consolidation,house,small_business and low for car,vacation,major_purchase.

Let us check how loan_amnt is distributed on "purpose".

In [87]:
```
loan_data.groupby(by="purpose")["loan_amnt"].mean().plot.bar()
```

Out[87]:
```
<AxesSubplot:xlabel='purpose'>
```

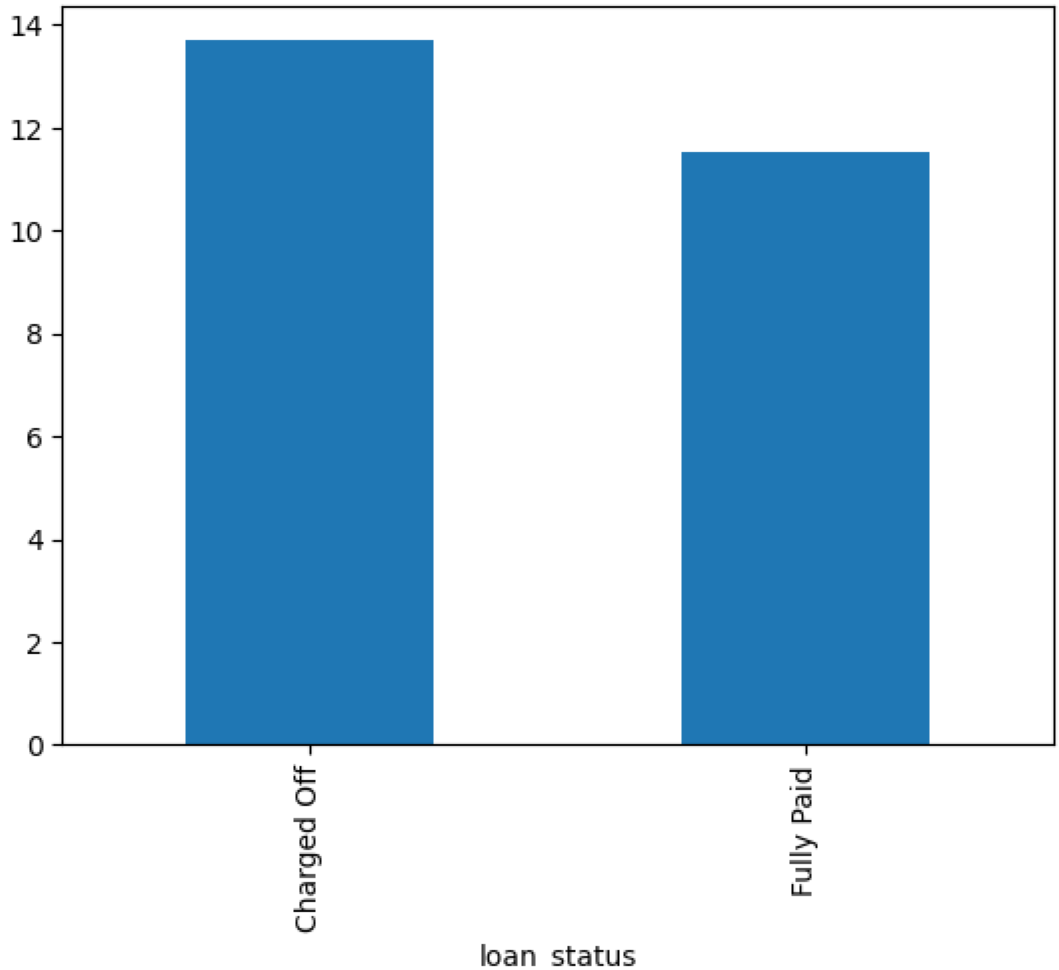


As we can see, loan_amnt is high for purposes of debt_consolidation,house,small_business.

Let us analyze how int_rate impact loan_status.

In [88]:
```
loan_data.groupby(by="loan_status")["int_rate"].mean().plot.bar()
```

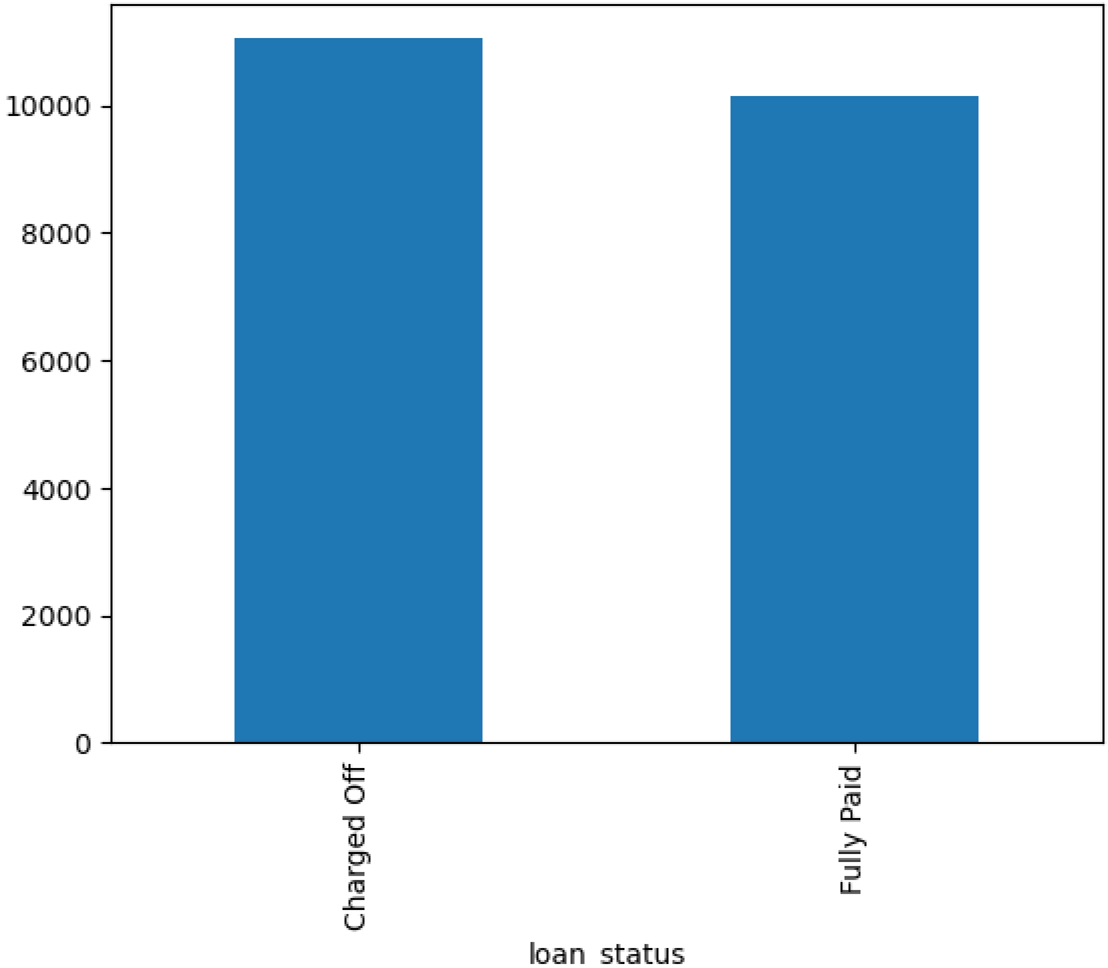
Out[88]:
```
<AxesSubplot:xlabel='loan_status'>
```



We can conclude, the persons who has charged off has high average int_rate than those who are fully paid.

In [89]:
```
loan_data.groupby(by="loan_status")["loan_amnt"].mean().plot.bar()
```

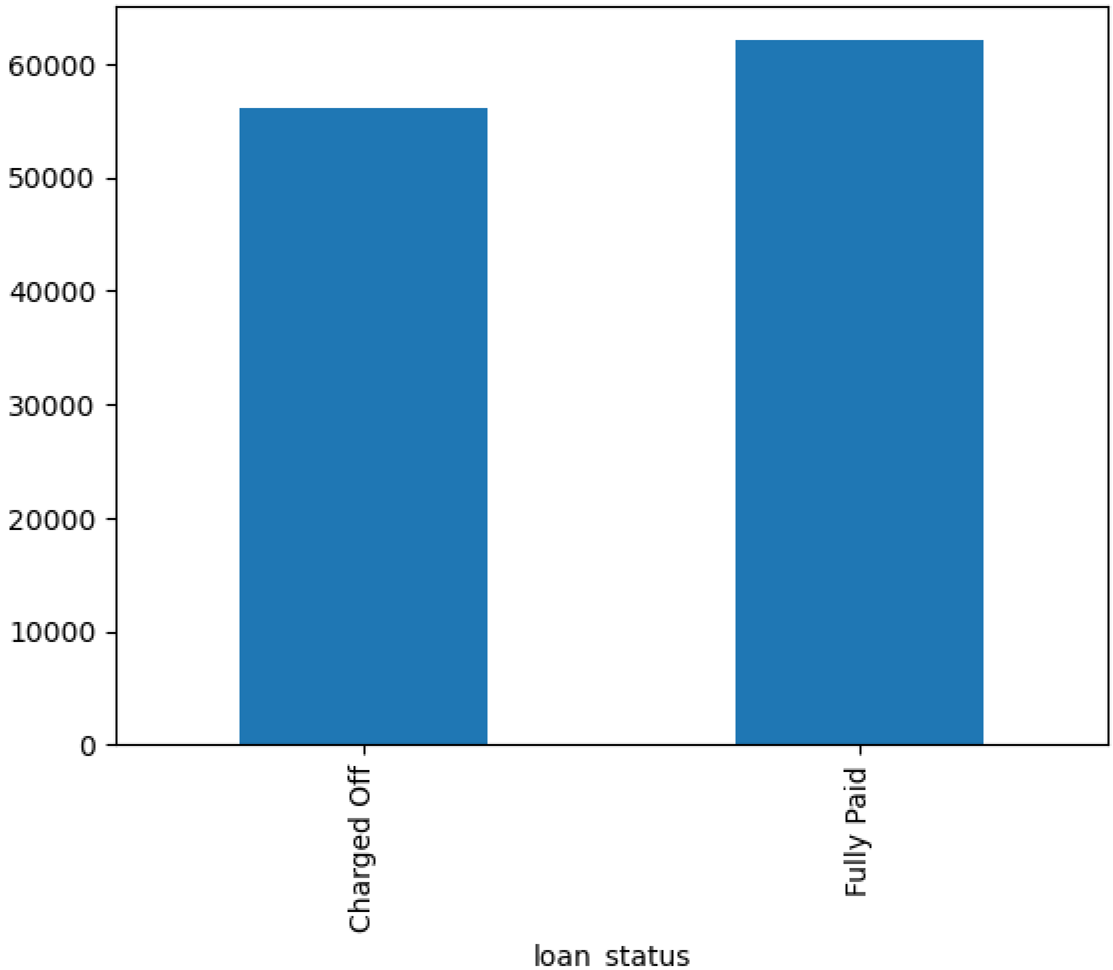
Out[89]:
```
<AxesSubplot:xlabel='loan_status'>
```



We can observe, the persons who has charged off has high average loan_amount than those who are fully paid.

In [90]:
```
loan_data.groupby(by="loan_status")["annual_inc"].mean().plot.bar()
```

Out[90]:
```
<AxesSubplot:xlabel='loan_status'>
```



We can observe, the persons who has charged off has low average annual_inc than those who are fully paid.

4. Bivariate analysis

Bivariate analysis is the process of analysing two variable at a time. This can be done in following steps.

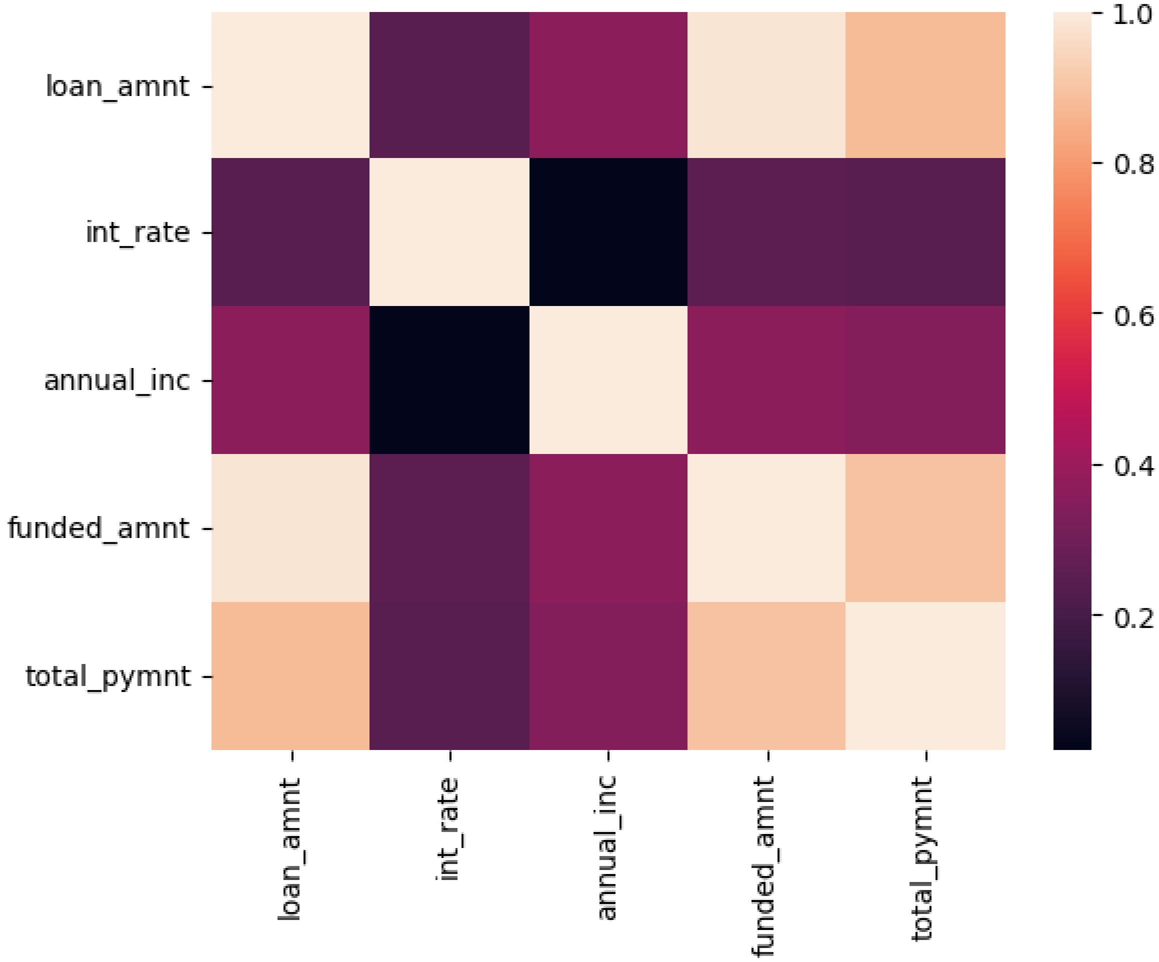
- Bivariate analysis on quantitative variables
- Bivariate analysis on categorical variables

Bivariate analysis on quantitative variables

Let us determine the **correlation** of some of the useful quantitative variables.

In [91]:
```
sns.heatmap(loan_data.loc[:,["loan_amnt","int_rate","annual_inc","funded_amnt","total_py
```

Out[91]:
```
<AxesSubplot:>
```



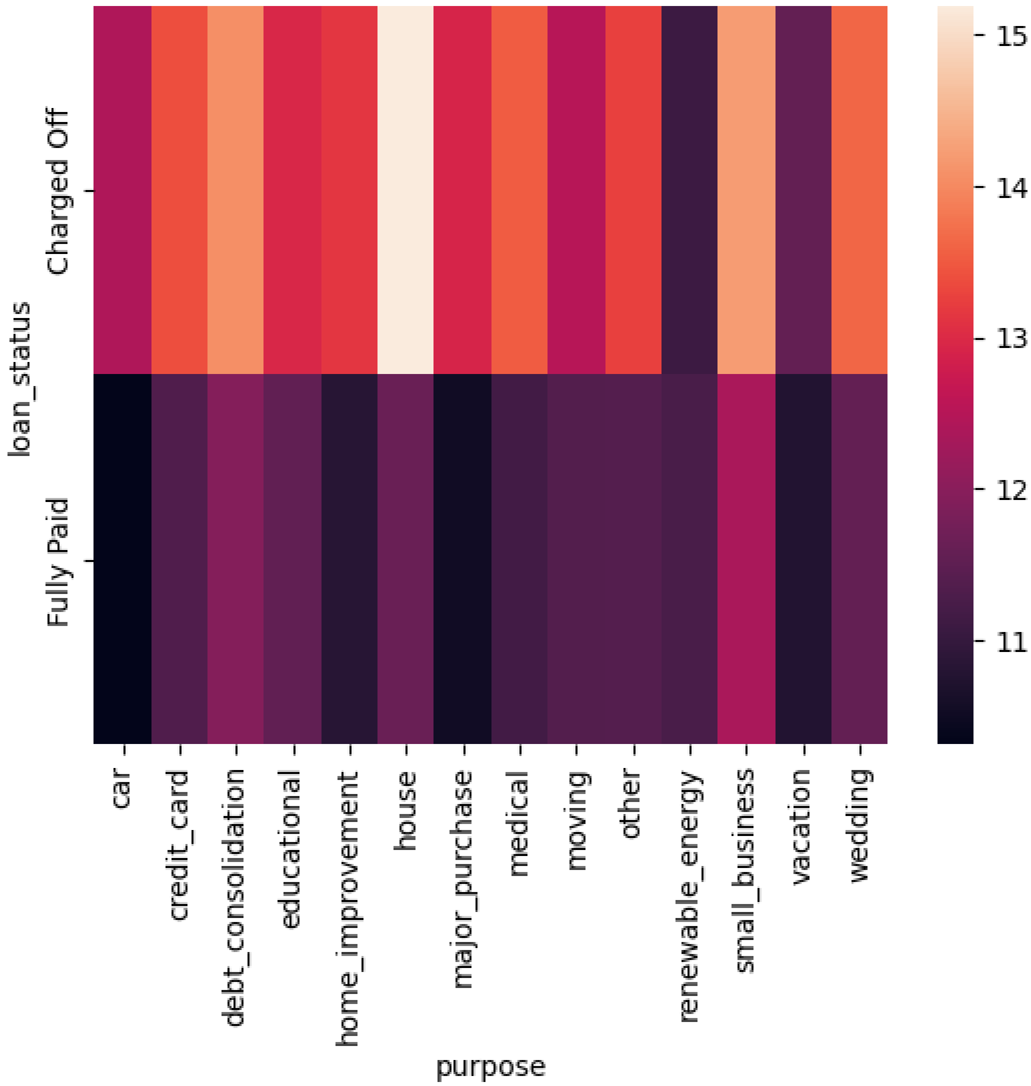
From the above heatmap, we can observe loan_amnt, funded_amnt and total_pymnt are strongly correlated.

Bivariate analysis on categorical variables

Let us analyse how purpose(categorical variable) affect the loan_status(categorical variable)

In [92]:
```
sns.heatmap(loan_data.pivot_table(index="loan_status",columns="purpose",values="int_rate
```

Out[92]:
```
<AxesSubplot:xlabel='purpose', ylabel='loan_status'>
```

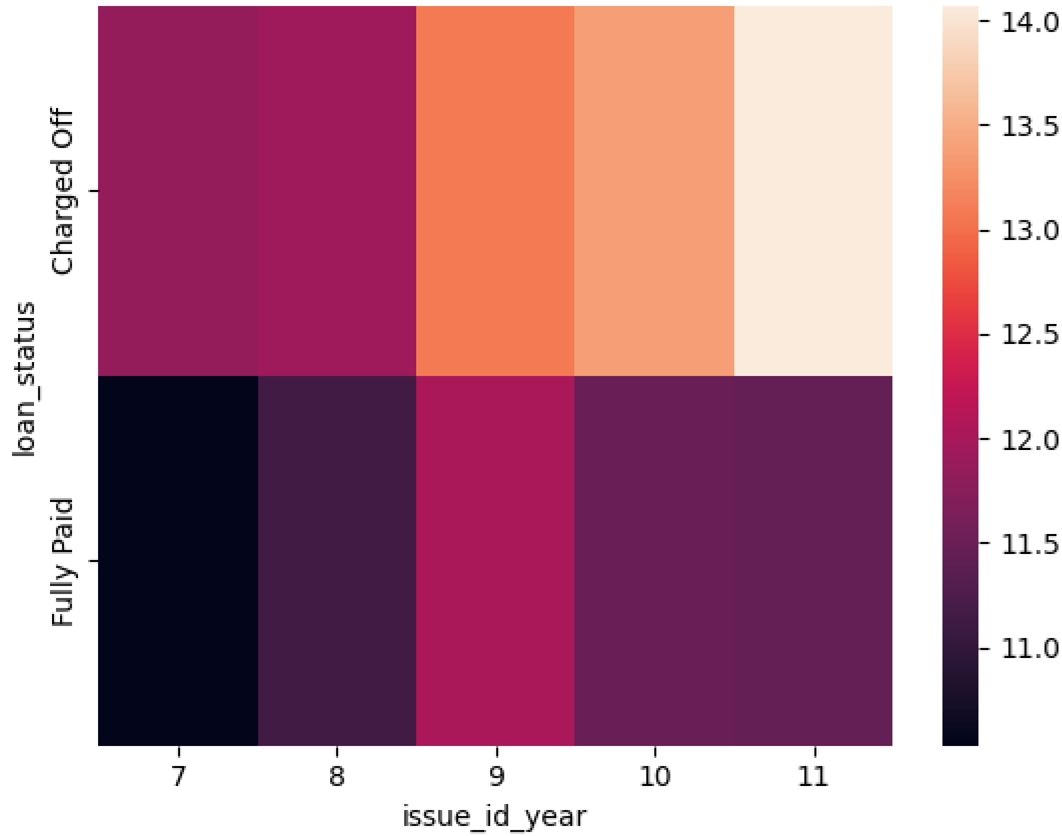


From the above heat_map, we can observe.
For purposes like debt_consolidation,home_improvement,house,medical - If the int_rate is more, they are likely to be charged off.

Let us analyse how issue_year(categorical) affect the loan_status(categorical)

In [93]:
```
sns.heatmap(loan_data.pivot_table(index="loan_status",columns="issue_id_year",values="in
```

Out[93]:
```
<AxesSubplot:xlabel='issue_id_year', ylabel='loan_status'>
```

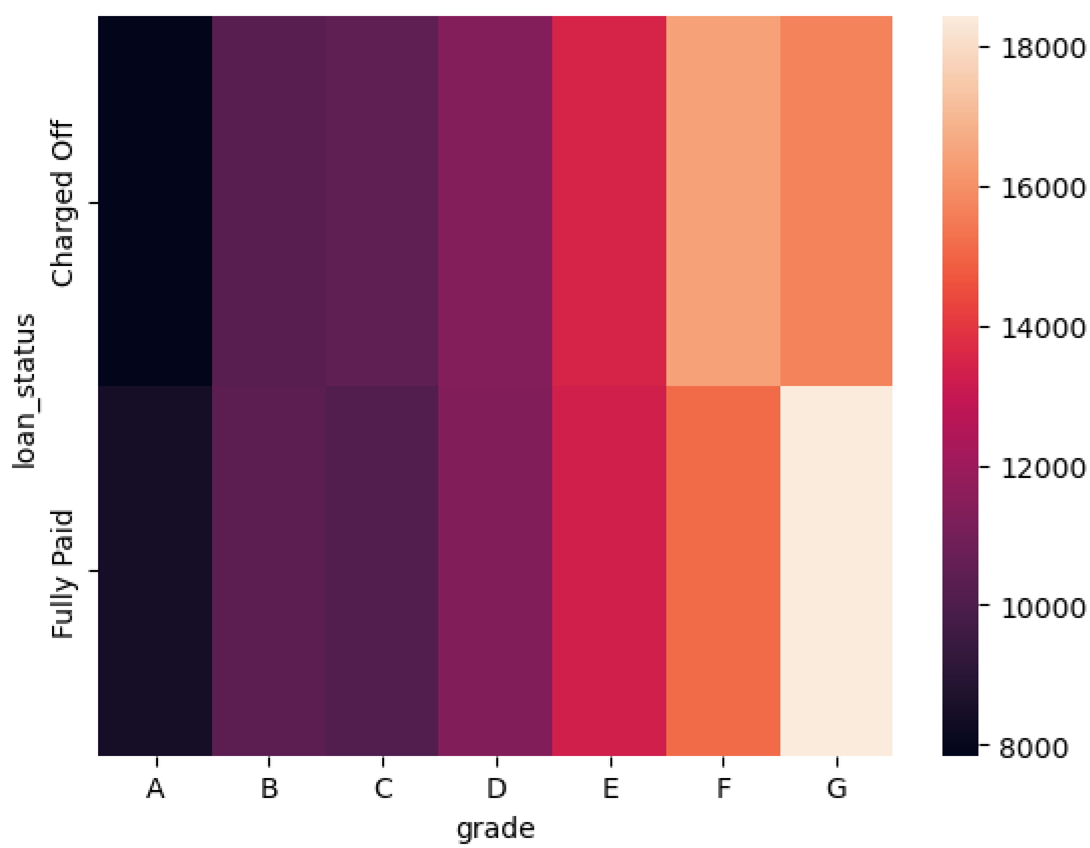


From the above plot, we can observe as the years pass, persons who have high int_rate are likely to be charged off and the rate is increasing.

Let us analyse how grade(categorical) affect the loan_status(categorical)

In [94]:
```
sns.heatmap(loan_data.pivot_table(index="loan_status",columns="grade",values="loan_amnt"
```

Out[94]:
```
<AxesSubplot:xlabel='grade', ylabel='loan_status'>
```



From the above plot, we can clearly observe - for grade G, If the loan amount is high, he is likely to be charged off.

5. Derived metrics

Derived metrics is the process of creating new variables using existing ones and get meaningful information by analysing them. It is of the following types:

- Type driven metrics
- Business driven metrics
- Data driven metrics

Type driven metrics

Let us create a new column for binning the annual income.

In [95]:
```
loan_data["annual_income_range"] = pd.cut(loan_data["annual_inc"],[0,30000,60000,90000,1
```

In [96]:
```
loan_data.loc[:,["annual_inc","annual_income_range"]]
```

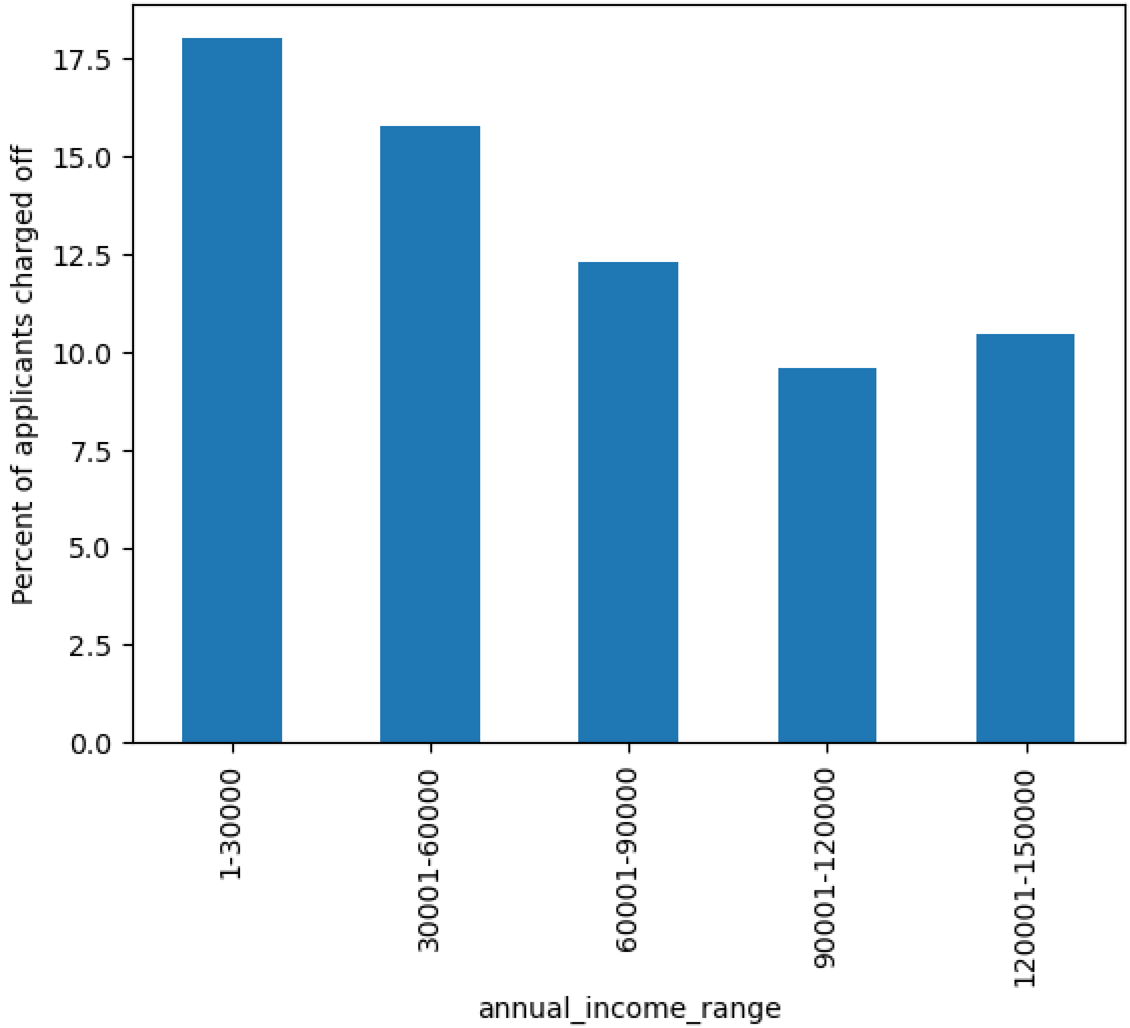
Out[96]:

	annual_inc	annual_income_range
0	24000.0	1-30000
1	30000.0	1-30000
2	12252.0	1-30000
3	49200.0	30001-60000
5	36000.0	30001-60000
...	...	...
39562	35000.0	30001-60000
39573	63500.0	60001-90000
39623	39000.0	30001-60000
39666	40000.0	30001-60000
39680	36153.0	30001-60000

34295 rows × 2 columns

In [97]:
```
dm_annual_inc=loan_data[loan_data["loan_status"] == "Charged Off"].groupby(by=["annual_i
ax=dm_annual_inc.plot.bar()
ax.set_ylabel("Percent of applicants charged off")
ax.plot()
```

Out[97]:
```
[]
```



By the above plot we can observe,More percentage of applicants whose annual income in range 1-30000 are likely to be be charged off.

Let us create a new column for binning the int_rate.

In [98]:
```
loan_data["int_rate_range"]=pd.cut(loan_data.int_rate,[5,8,11,14,17,20,23],labels=["5-8"
```

In [99]:
```
loan_data.loc[:,["int_rate","int_rate_range"]]
```

Out[99]:

	int_rate	int_rate_range
0	10.65	8-11
1	15.27	14-17
2	15.96	14-17
3	13.49	11-14
5	7.90	5-8
...	...	...
39562	10.28	8-11
39573	10.59	8-11
39623	12.49	11-14
39666	11.22	11-14
39680	11.86	11-14

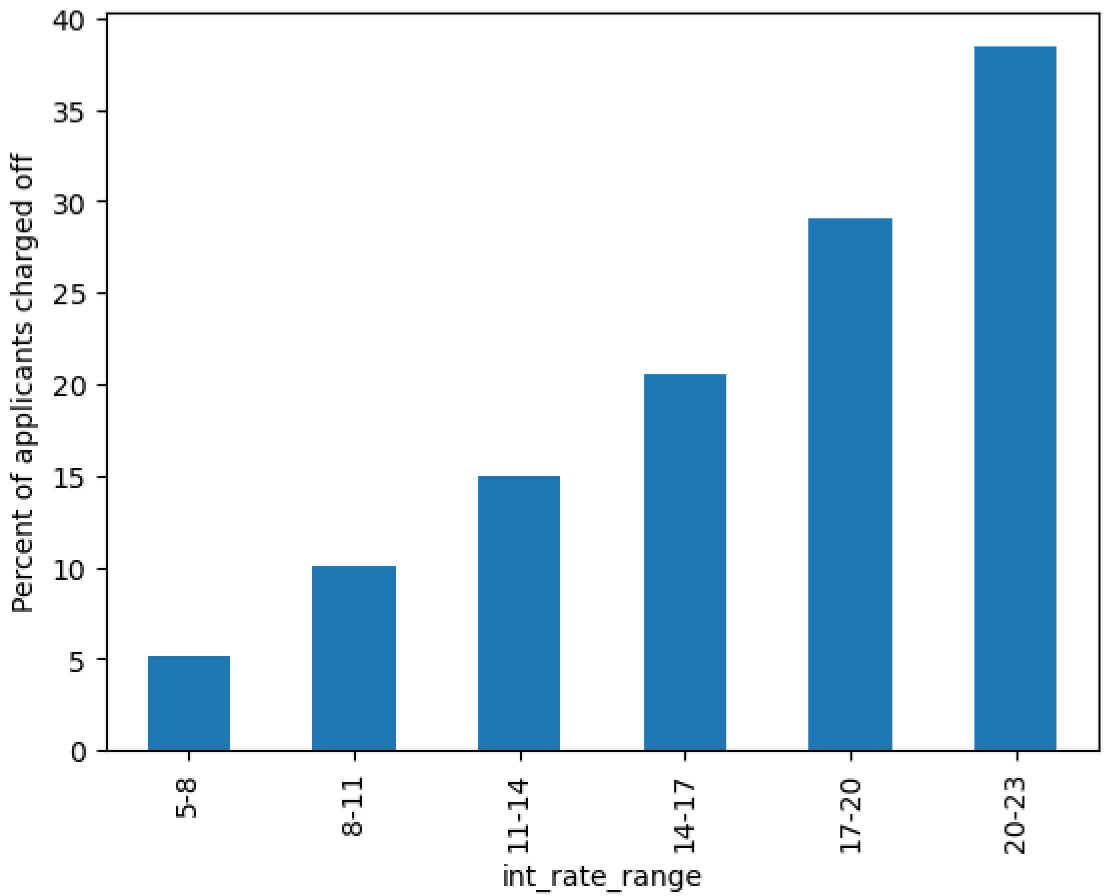
34295 rows × 2 columns

In [100…

```
dm_int_rate=loan_data[loan_data["loan_status"] == "Charged Off"].groupby(by=["int_rate_r
ax=dm_int_rate.plot.bar()
ax.set_ylabel("Percent of applicants charged off")
ax.plot()
```

Out[100]:

```
[]
```



From the above plot, we can observe - The charged off percentage of applicants whose int_rate is above 14 is more compare to those whose int_rate is less than 14 and it is very high for those in range 17-23.
The charged off percentage is increasing with increasing in the int_rate.

Let us create a new column for binning the loan_amnt.

In [101…

```
loan_data["loan_amnt_range"]=pd.cut(loan_data.loan_amnt,[0,5000,10000,15000,20000,25000,
```

In [102…

```
loan_data.loc[:,["loan_amnt","loan_amnt_range"]]
```

Out[102]:

	loan_amnt	loan_amnt_range
0	5000	0-5000
1	2500	0-5000
2	2400	0-5000
3	10000	5001-10000
5	5000	0-5000
...	...	...
39562	4800	0-5000

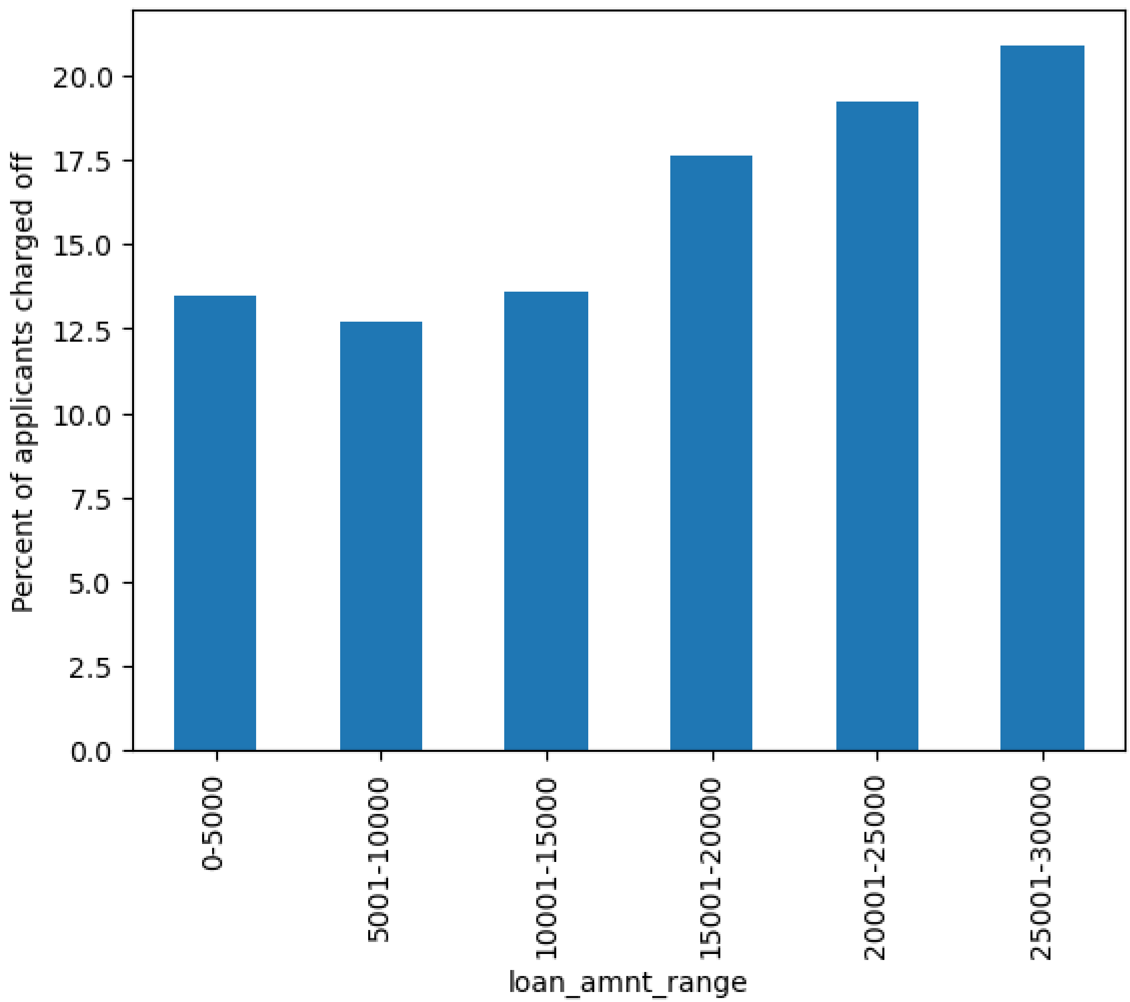
39573	7000	5001-10000
39623	9000	5001-10000
39666	15450	15001-20000
39680	3000	0-5000

34295 rows × 2 columns

In [103…

```
dm_int_rate=loan_data[loan_data["loan_status"] == "Charged Off"].groupby(by=["loan_amnt_
ax=dm_int_rate.plot.bar()
ax.set_ylabel("Percent of applicants charged off")
ax.plot()
```

Out[103]: []



From the above plot, we can observe the charged off percent is high for the loan amount more than 15000 and is even more if loan amount is more than 25000.

Business driven metrics

Let us derive a new variable "monthly_inc" and "percent_of_installment_on_monthly_income" and see how the second one has impact on the loan_status.

In [104…

```
loan_data["monthly_inc"]=loan_data.annual_inc/12
```

In [105…

```
loan_data.loc[:,["annual_inc","monthly_inc"]]
```

Out[105]:

	annual_inc	monthly_inc
0	24000.0	2000.000000
1	30000.0	2500.000000
2	12252.0	1021.000000
3	49200.0	4100.000000
5	36000.0	3000.000000
...	...	...
39562	35000.0	2916.666667
39573	63500.0	5291.666667
39623	39000.0	3250.000000
39666	40000.0	3333.333333
39680	36153.0	3012.750000

34295 rows × 2 columns

In [106…

```
loan_data["percent_of_installment_on_monthly_income"] = 100*loan_data.installment/loan_d
```

In [107…

```
loan_data.loc[:,["installment","monthly_inc","percent_of_installment_on_monthly_income"]
```

Out[107]:

	installment	monthly_inc	percent_of_installment_on_monthly_income
0	162.87	2000.000000	8.143500
1	59.83	2500.000000	2.393200
2	84.33	1021.000000	8.259549
3	339.31	4100.000000	8.275854
5	156.46	3000.000000	5.215333
...	...	...	...
39562	155.52	2916.666667	5.332114
39573	227.82	5291.666667	4.305260
39623	301.04	3250.000000	9.262769
39666	507.46	3333.333333	15.223800
39680	99.44	3012.750000	3.300639

34295 rows × 3 columns

In [108…

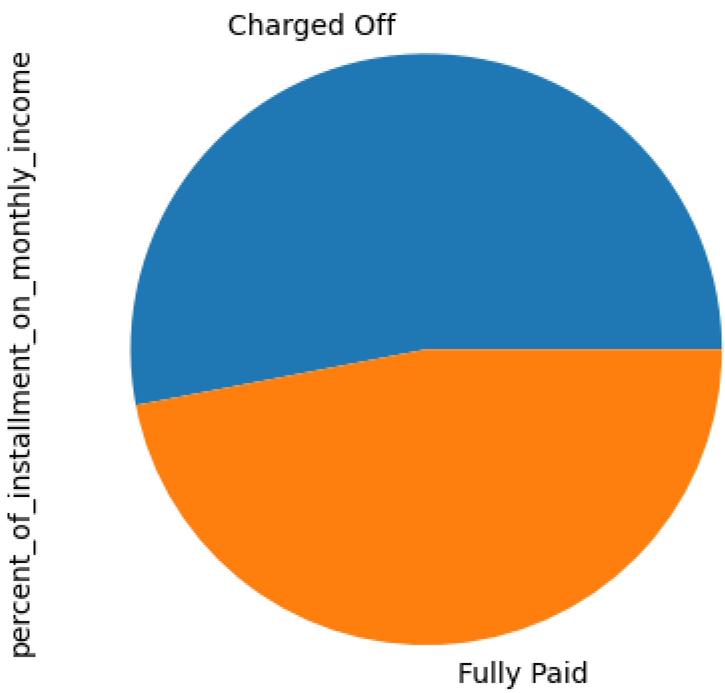
```
loan_data.percent_of_installment_on_monthly_income.describe()
```

Out[108]:

```
count    34295.000000
mean         6.560226
std          3.906670
min          0.216686
25%          3.594379
50%          5.807538
75%          8.760527
max         29.016500
Name: percent_of_installment_on_monthly_income, dtype: float64
```

In [109…
```
loan_data.groupby(by="loan_status")["percent_of_installment_on_monthly_income"].mean().p
```

Out[109]:
```
<AxesSubplot:ylabel='percent_of_installment_on_monthly_income'>
```



We can observe percent_of_installment_on_monthly_income plays a slight role to determine loan status. The applicants whose the percentage of installment on monthly income is more are likely to be Charged Off.