Given historical data on loans given out with information on whether or not the borrower defaulted (charge-off), can we build a model that can predict wether or nor a borrower will pay back their loan? This way in the future when we get a new potential customer we can assess whether or not they are likely to pay back the loan. Keep in mind classification metrics when evaluating the performance of your model! source: https://www.kaggle.com/harlfoxem/housesalesprediction

In [1]:

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

In [2]:

```python
Data=pd.read_csv(r'C:\Users\chumj\Downloads\lending.csv')
```

In [3]:

```python
Data.head(3)
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	open_acc	pu
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	16.0	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	17.0	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	13.0	

3 rows × 27 columns

In [4]:

```python
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
```

```
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

EXPLORATORY DATA ANALYSIS.

Keep in mind our soul aim is to predict if a customer will predict or default a loan. Let's start by exploring and anaylsing some import faetures in our Dataset.The loan status itself is a very important feature,lets start with it.

In [5]:

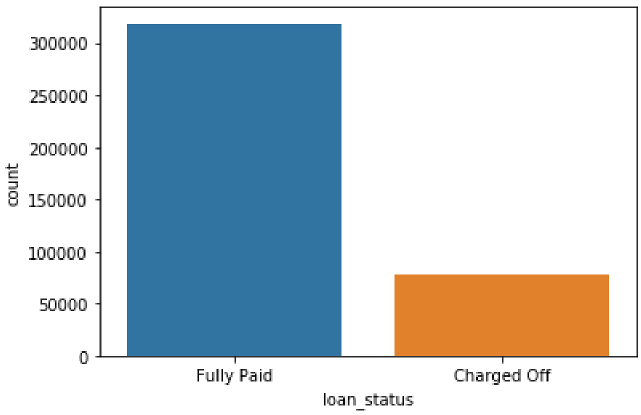
```
Data.columns
```

Out[5]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

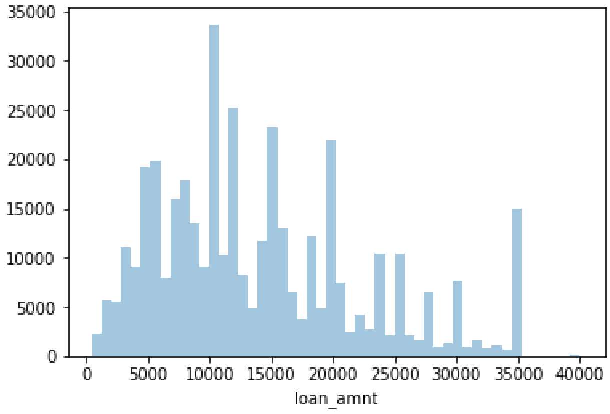
In [6]:

```
sns.countplot(x='loan_status',data=Data);
```



In [7]:

```
#loan amount
sns.distplot(Data['loan_amnt'],bins=50,kde=False);
```



In [8]:

```python
#Launching correlation between the continous variable features.
Data.corr()
```

Out[8]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mo
loan_amnt	1.000000	0.168921	0.953929	0.336887	0.016636	0.198556	-0.077779	0.328320	0.099911	0.223886	0.2
int_rate	0.168921	1.000000	0.162758	-0.056771	0.079038	0.011649	0.060986	-0.011280	0.293659	-0.036404	0.0
installment	0.953929	0.162758	1.000000	0.330381	0.015786	0.188973	-0.067892	0.316455	0.123915	0.202430	0.1
annual_inc	0.336887	-0.056771	0.330381	1.000000	-0.081685	0.136150	-0.013720	0.299773	0.027871	0.193023	0.2
dti	0.016636	0.079038	0.015786	-0.081685	1.000000	0.136181	-0.017639	0.063571	0.088375	0.102128	0.0
open_acc	0.198556	0.011649	0.188973	0.136150	0.136181	1.000000	-0.018392	0.221192	-0.131420	0.680728	0.1
pub_rec	-0.077779	0.060986	-0.067892	-0.013720	-0.017639	-0.018392	1.000000	-0.101664	-0.075910	0.019723	0.0
revol_bal	0.328320	-0.011280	0.316455	0.299773	0.063571	0.221192	-0.101664	1.000000	0.226346	0.191616	0.1
revol_util	0.099911	0.293659	0.123915	0.027871	0.088375	-0.131420	-0.075910	0.226346	1.000000	-0.104273	0.0
total_acc	0.223886	-0.036404	0.202430	0.193023	0.102128	0.680728	0.019723	0.191616	-0.104273	1.000000	0.3
mort_acc	0.222315	-0.082583	0.193694	0.236320	-0.025439	0.109205	0.011552	0.194925	0.007514	0.381072	1.0
pub_rec_bankruptcies	-0.106539	0.057450	-0.098628	-0.050162	-0.014558	-0.027732	0.699408	-0.124532	-0.086751	0.042035	0.0

In [9]:

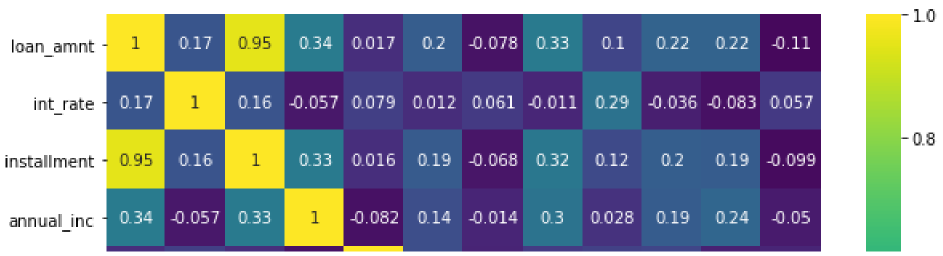
```python
Data.corr()['loan_amnt'].sort_values(ascending=False)
```

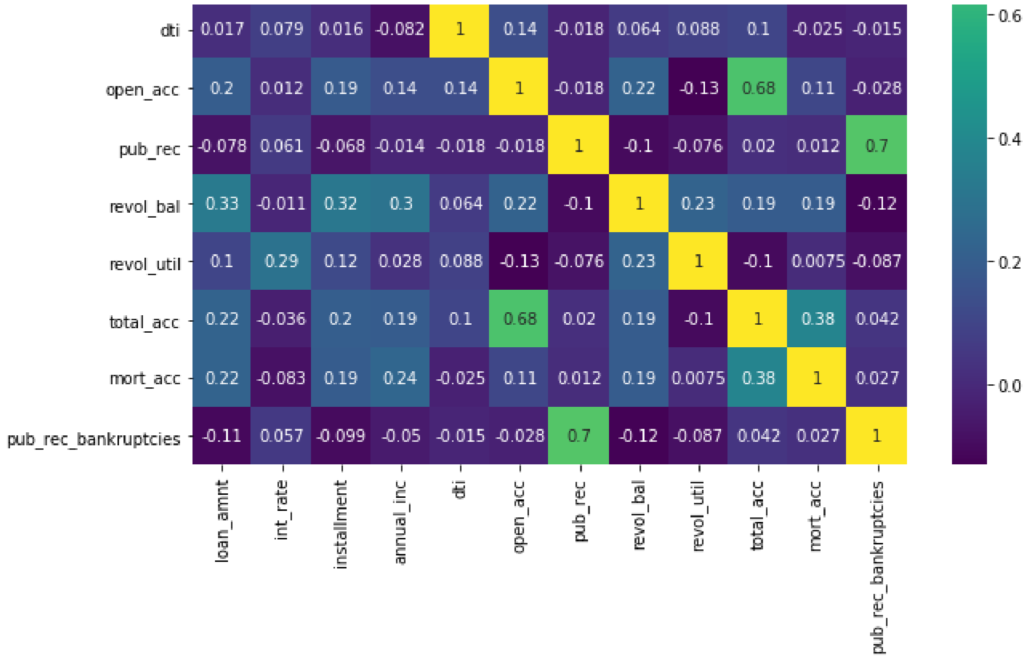
Out[9]:

```
loan_amnt               1.000000
installment             0.953929
annual_inc              0.336887
revol_bal               0.328320
total_acc               0.223886
mort_acc                0.222315
open_acc                0.198556
int_rate                0.168921
revol_util              0.099911
dti                     0.016636
pub_rec                -0.077779
pub_rec_bankruptcies   -0.106539
Name: loan_amnt, dtype: float64
```

In [10]:

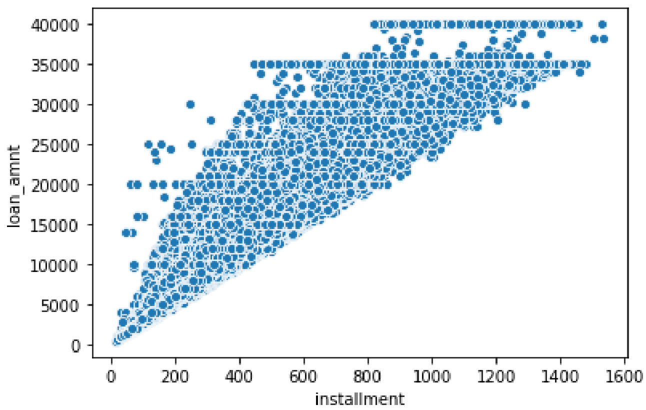
```python
plt.figure(figsize=(10,8))
sns.heatmap(Data.corr(),annot=True,cmap='viridis');
```





In [11]:

```
#installment and loan amout seems to correlation well,lets take a look for more Analysis
sns.scatterplot(x='installment',y='loan_amnt',data=Data);
```



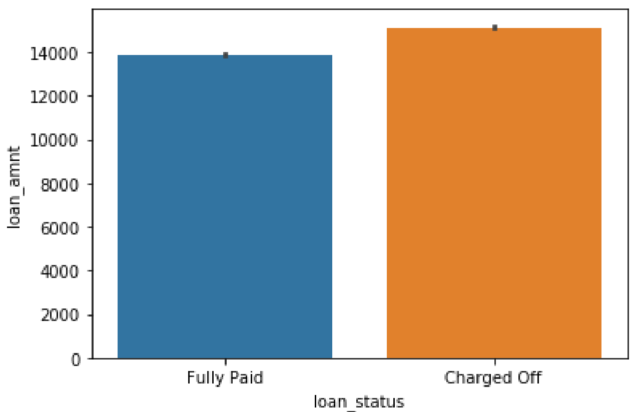
lets take look at loan ststus(categorical feature) and loan amount.

In [12]:

```
sns.barplot(x='loan_status',y='loan_amnt',data=Data)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18ce36ad4c8>
```

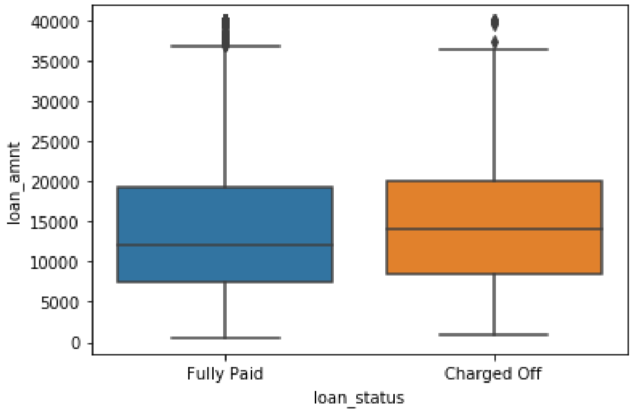


In [13]:

```
sns.boxplot(x='loan_status',y='loan_amnt',data=Data)
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18ce370c9c8>
```



In [14]:

```
Data.groupby('loan_status')['loan_amnt'].describe()
```

Out[14]:

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

In [15]:

```
#sns.kdeplot(Data['installment'],Data['loan_amnt'],
                 #cmap="plasma", shade=True, shade_lowest=False);
```

In [ ]:

```

```

In [16]:

```
#lets take a look of the Grading and subGrading in relation to the loan
```

In [17]:

```
sorted(Data['grade'].unique())
```

Out[17]:

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

In [18]:

```
sorted(Data['sub_grade'].unique())
```

Out[18]:

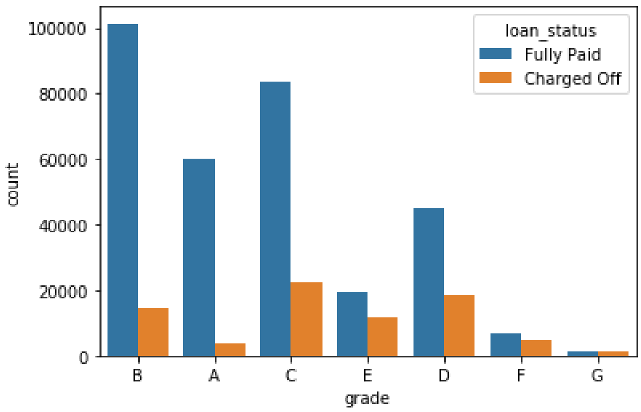
```
['A1',
 'A2',
```

```
'A3',
'A4',
'A5',
'B1',
'B2',
'B3',
'B4',
'B5',
'C1',
'C2',
'C3',
'C4',
'C5',
'D1',
'D2',
'D3',
'D4',
'D5',
'E1',
'E2',
'E3',
'E4',
'E5',
'F1',
'F2',
'F3',
'F4',
'F5',
'G1',
'G2',
'G3',
'G4',
'G5']
```

let's check the grading with relation to loan status graphically

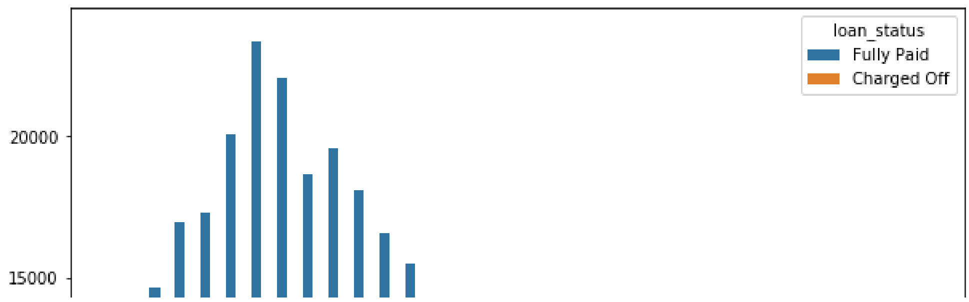
In [19]:

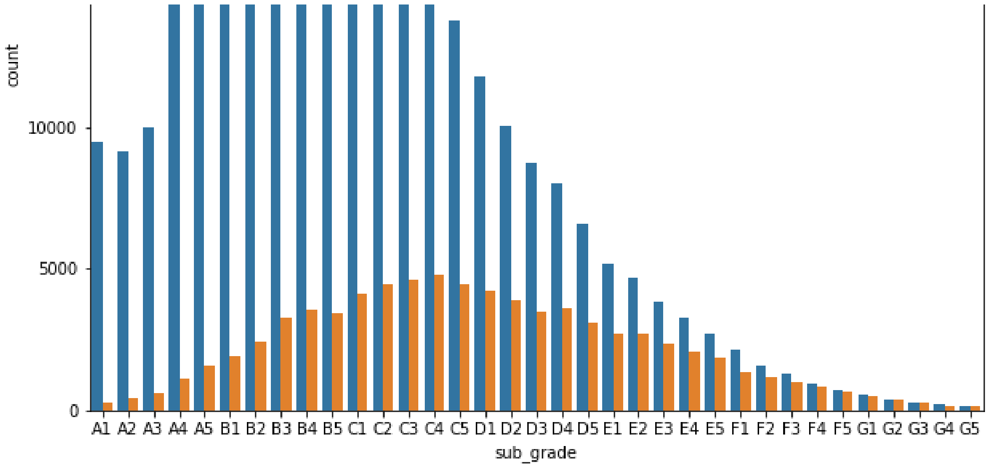
```
sns.countplot(x='grade',hue='loan_status',data=Data);
```



In [20]:

```
plt.figure(figsize=(10,8))
subgrade_order=sorted(Data['sub_grade'].unique())
sns.countplot(x='sub_grade',data=Data,order=subgrade_order,hue='loan_status');
```





something seems to be happening with F and G,It seems they do not paid back that often

In [21]:

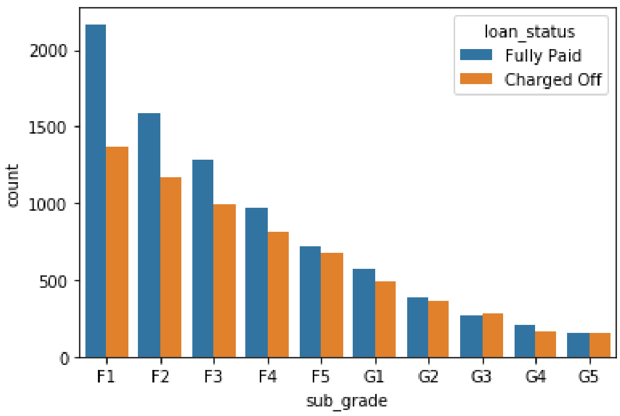
```
F_and_G=Data[(Data['grade']=='G')|(Data['grade']=='F')]
subgrade_order=sorted(F_and_G['sub_grade'].unique())
```

In [22]:

```
sns.countplot(x='sub_grade',order=subgrade_order,hue='loan_status',data=F_and_G)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18ce16e2648>
```



Loan status recall is the main feature we our about to predict,lets change "Fully Paid"=1,"Charged Off"=0

In [23]:

```
Data['loan_status'].unique()
```

Out[23]:

```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [24]:

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

In [25]:

```
Data[['repaid_loans','loan_status']].head(5)
```

Out[25]:

	repaid_loans	loan_status
0	1	Fully Paid
1	1	Fully Paid
2	1	Fully Paid
3	1	Fully Paid
4	0	Charged Off

In [26]:

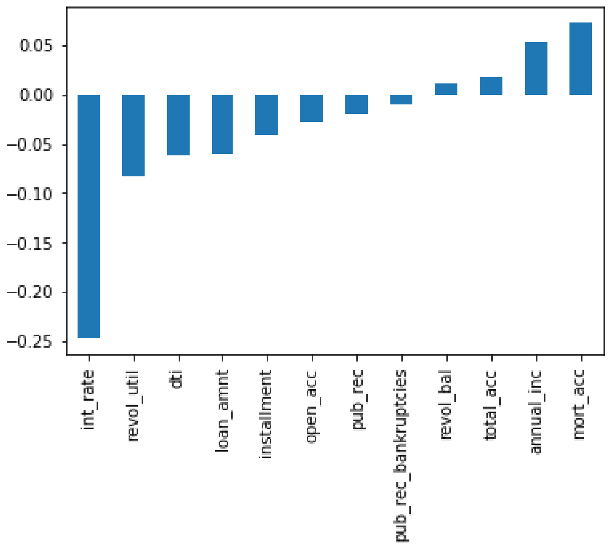
```
Data['repaid_loans']
```

Out[26]:

```
0         1
1         1
2         1
3         1
4         0
         ..
396025    1
396026    1
396027    1
396028    1
396029    1
Name: repaid_loans, Length: 396030, dtype: int64
```

In [27]:

```
Data.corr()['repaid_loans'].sort_values().drop('repaid_loans').plot(kind='bar');
```



In [28]:

```
Data.corr()['repaid_loans'].sort_values().drop('repaid_loans')
```

Out[28]:

```
int_rate               -0.247758
revol_util             -0.082373
dti                    -0.062413
loan_amnt              -0.059836
installment            -0.041082
open_acc               -0.028012
pub_rec                -0.019933
pub_rec_bankruptcies   -0.009383
revol_bal               0.010892
total_acc               0.017893
annual_inc              0.053432
```

```
mort_acc                0.073111
Name: repaid_loans, dtype: float64
```

PREPROCESSING OF OUR DATASET

Converting categorical strings features to dummy variable,removing and filling missing data

In [29]:

```
Data.head(2)
```

Out[29]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	pub_rec	revo
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	0.0	36
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	0.0	20

2 rows × 28 columns

In [30]:

```
len(Data)
```

Out[30]:

```
396030
```

In [31]:

```
Data.isnull().sum()
```

Out[31]:

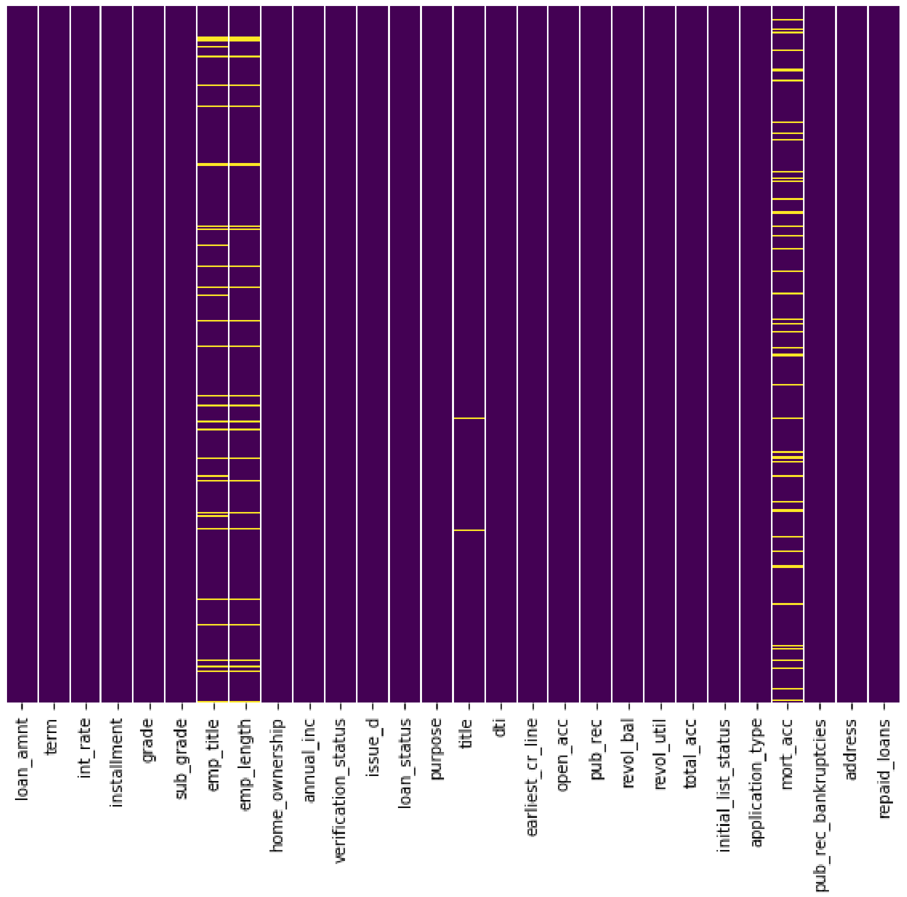
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
repaid_loans                0
dtype: int64
```

In [32]:

```python
plt.figure(figsize=(10,8))
sns.heatmap(Data.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[32]:

<matplotlib.axes._subplots.AxesSubplot at 0x18ce1227f88>



In [33]:

```python
# percentage of the entire dataset
Data.isnull().sum()/len(Data)*100
```

Out[33]:

```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
```

```
repaid_loans             0.000000
dtype: float64
```

In [34]:

```
# let's start with mort_acc of almost 10%,we cannot drop it out
Data['mort_acc'].value_counts()
```

Out[34]:

```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
23.0         2
32.0         2
26.0         2
31.0         2
30.0         1
28.0         1
34.0         1
Name: mort_acc, dtype: int64
```

In [35]:

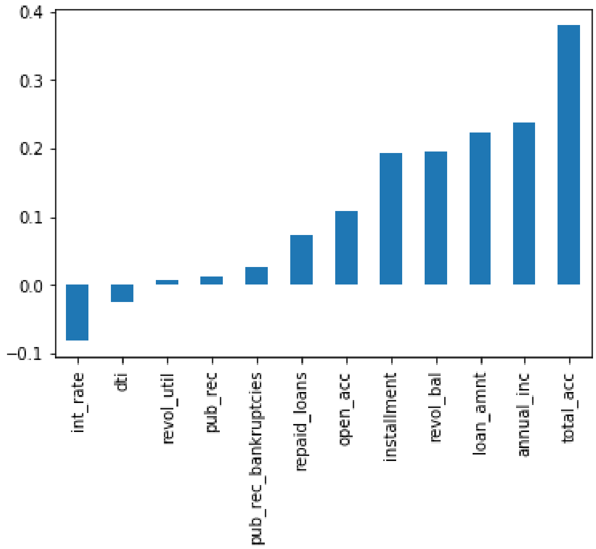
```
#looking which feature correlate most with mort_acc,then select teh faeture and look for it's mean

#and later apply it to the missing values in mort_acc
Data.corr()['mort_acc'].sort_values()
```

Out[35]:

```
int_rate               -0.082583
dti                    -0.025439
revol_util              0.007514
pub_rec                 0.011552
pub_rec_bankruptcies    0.027239
repaid_loans            0.073111
open_acc                0.109205
installment             0.193694
revol_bal               0.194925
loan_amnt               0.222315
annual_inc              0.236320
total_acc               0.381072
mort_acc                1.000000
Name: mort_acc, dtype: float64
```

In [36]:

```
Data.corr()['mort_acc'].sort_values().drop('mort_acc').plot(kind='bar');
```



In [37]:

```
#total_acc featurs correlate most with mort_acc,let use the fillna() by groupby the total_acc
#and calculate mean value for the mort_acc per entry.
print('mean of mort_acc col per total_acc')
Data.groupby('total_acc').mean()['mort_acc']
```

```
mean of mort_acc col per total_acc
```

Out[37]:

```
total_acc
2.0      0.000000
3.0      0.052023
4.0      0.066743
5.0      0.103289
6.0      0.151293
           ...
124.0    1.000000
129.0    1.000000
135.0    3.000000
150.0    2.000000
151.0    0.000000
Name: mort_acc, Length: 118, dtype: float64
```

In [38]:

```
# if mort_acc is missing,we fill in that missing value with the mean value corresponding to it's t
otal_acc
total_acc_avg=Data.groupby('total_acc').mean()['mort_acc']
```

In [39]:

```
total_acc_avg[2]
# we can choose,2=0,3=o.052023 etc
```

Out[39]:

```
0.0
```

In [40]:

```
def fill_mort_acc(total_acc,mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc]
    else:
        return mort_acc
```

In [41]:

```
Data['mort_acc']=Data.apply(lambda x:fill_mort_acc(x['total_acc'],x['mort_acc']),axis=1)
```

In [42]:

```
Data.isnull().sum()
```

Out[42]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                    0
pub_rec_bankruptcies      535
address                     0
repaid_loans                0
dtype: int64
```

In [43]:

```
#let's take a look at emp_title
Data['emp_title'].nunique()
```

Out[43]:

```
173105
```

In [44]:

```
Data['emp_title'].value_counts()
#from below it shows many emp_title and is not realistic to convert them to a dummy variable,it ma
kes sense if we drop it.
```

Out[44]:

```
Teacher                          4389
Manager                          4250
Registered Nurse                 1856
RN                               1846
Supervisor                       1830
                                 ...
Dore Academy                        1
Loan servicing                      1
Director of Quality And Admin       1
admitting registrar                 1
Rockford Police Department          1
Name: emp_title, Length: 173105, dtype: int64
```

In [45]:

```
Data=Data.drop('emp_title',axis=1)
```

In [46]:

```
Data['emp_length'].value_counts()
```

Out[46]:

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

In [47]:

```
sorted(Data['emp_length'].dropna().unique())
```

Out[47]:

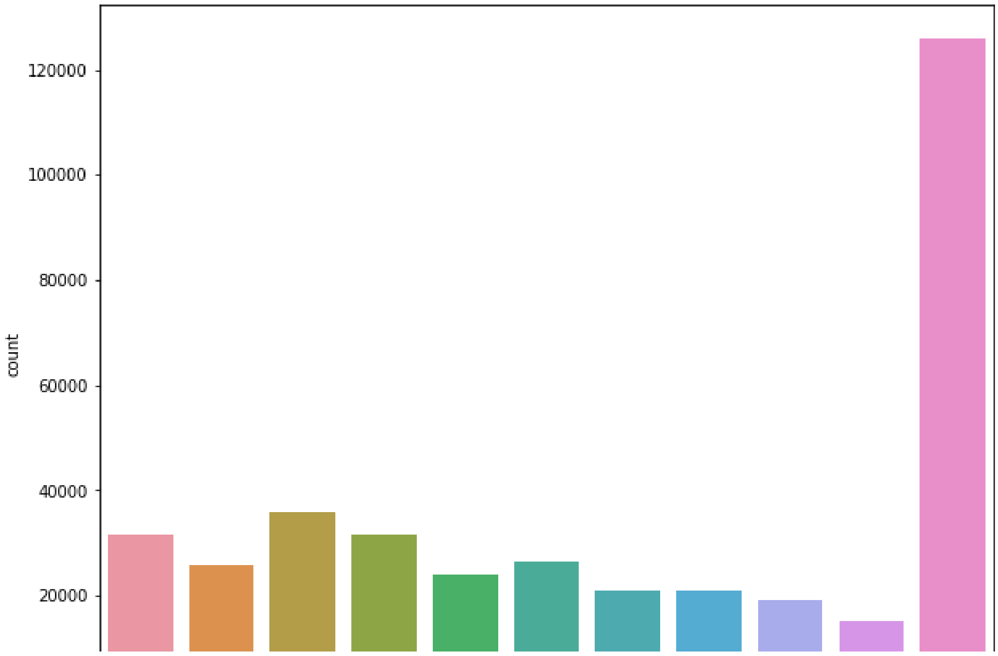
```
['1 year',
 '10+ years',
 '2 years',
 '3 years',
 '4 years',
 '5 years',
 '6 years',
 '7 years',
 '8 years',
 '9 years',
 '< 1 year']
```

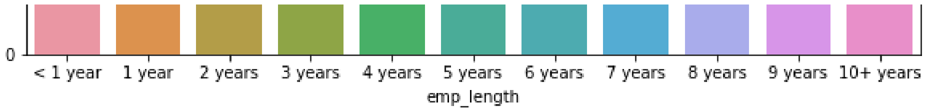
In [48]:

```
emp_length_order= ['< 1 year','1 year','2 years', '3 years', '4 years','5 years','6 years','7 years','8 years','9 years','10+ years',]
```

In [49]:

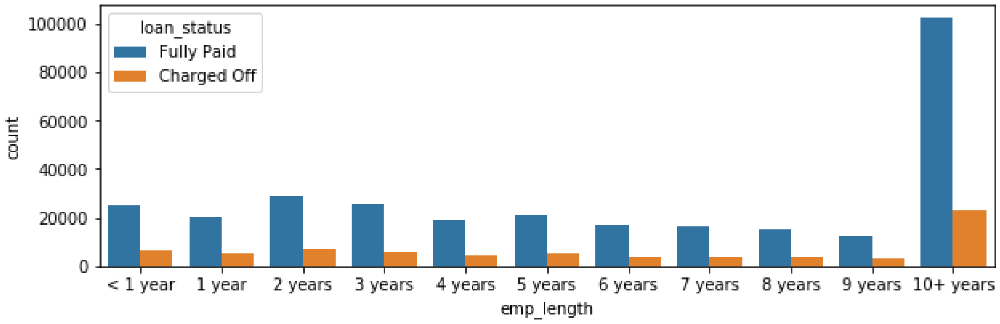
```
plt.figure(figsize=(10,8))
sns.countplot(x='emp_length',order=emp_length_order,data=Data);
```





In [50]:

```
plt.figure(figsize=(10,3))
sns.countplot(x='emp_length',order=emp_length_order,data=Data,hue='loan_status');
# looking at the plot below,charged off seems to be even from <1year....10+years,no real pattern w
e can exactract from emp_length,
#so it makes sense if we drop it.
```



In [51]:

```
Data=Data.drop('emp_length',axis=1)
```

In [52]:

```
# checking title
Data['title'].value_counts()
```

Out[52]:

```
Debt consolidation          152472
Credit card refinancing      51487
Home improvement             15264
Other                        12930
Debt Consolidation           11608
                             ...
Moving Cost                      1
To North Dakota                  1
PAY LESS                         1
2013 Life Change                 1
Save my sanity                   1
Name: title, Length: 48817, dtype: int64
```

In [53]:

```
Data['title'].unique()
```

Out[53]:

```
array(['Vacation', 'Debt consolidation', 'Credit card refinancing', ...,
       'Credit buster ', 'Loanforpayoff', 'Toxic Debt Payoff'],
      dtype=object)
```

In [54]:

```
Data['purpose']
```

Out[54]:

```
0                    vacation
1          debt_consolidation
2                 credit_card
3                 credit_card
```

```
4                 credit_card
                  ...
396025    debt_consolidation
396026    debt_consolidation
396027    debt_consolidation
396028    debt_consolidation
396029    debt_consolidation
Name: purpose, Length: 396030, dtype: object
```

Title and purpose seems to be the same information,droping title makes sense.

In [55]:

```
Data=Data.drop('title',axis=1)
```

In [56]:

```
Data.isnull().sum()
```

Out[56]:

```
loan_amnt                 0
term                      0
int_rate                  0
installment               0
grade                     0
sub_grade                 0
home_ownership            0
annual_inc                0
verification_status       0
issue_d                   0
loan_status               0
purpose                   0
dti                       0
earliest_cr_line          0
open_acc                  0
pub_rec                   0
revol_bal                 0
revol_util              276
total_acc                 0
initial_list_status       0
application_type          0
mort_acc                  0
pub_rec_bankruptcies    535
address                   0
repaid_loans              0
dtype: int64
```

revol_util and pub_rec_bankruptcies just record less than 0.5% so it insignificant,so we can drop them.

In [57]:

```
Data=Data.dropna()
```

In [58]:

```
#Data.isnull().sum()
```

In [59]:

```
Data.select_dtypes(['object']).columns
```

Out[59]:

```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'issue_d', 'loan_status', 'purpose', 'earliest_cr_line',
       'initial_list_status', 'application_type', 'address'],
      dtype='object')
```

let's take a look of all the string features

In [60]:

```
Data['term'].nunique()
```

Out[60]:

```
2
```

In [61]:

```
Data['term'].value_counts()
```

Out[61]:

```
 36 months    301247
 60 months     93972
Name: term, dtype: int64
```

In [62]:

```
Data['term']=Data['term'].apply(lambda term:int(term[:3]))
```

In [63]:

```
#grade is a sub_grade,so we can drop the grade feature.
Data=Data.drop('grade',axis=1)
```

In [64]:

```
Data['home_ownership'].nunique()
```

Out[64]:

```
6
```

In [65]:

```
Data['home_ownership'].value_counts()
```

Out[65]:

```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

In [66]:

```
# we can convert home_ownership to dummy variable,lets try to combine 'none 'and 'any' to 'Other'
```

In [67]:

```
Data['home_ownership']=Data['home_ownership'].replace(['NONE','ANY'],'OTHER')
Dummies=pd.get_dummies(Data['home_ownership'],drop_first=True)
Data=Data.drop('home_ownership',axis=1)
Data=pd.concat([Data,Dummies],axis=1)
```

In [68]:

```
#for the address col we can extract the zip code from the address,we can make a new col call zip
Data['zip']=Data['address'].apply(lambda address:address[-5:])
```

In [69]:

```
Data['address'].head(2)
```

Out[69]:

```
0       0174 Michelle Gateway\r\nMendozaberg, OK 22690
1    1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113
Name: address, dtype: object
```

In [70]:

```
Dummies=pd.get_dummies(Data['zip'],drop_first=True)
Data=Data.drop(['zip','address'],axis=1)
Data=pd.concat([Data,Dummies],axis=1)
```

In [71]:

```
Data['issue_d'].head(3)
#this can be track or leakage,telling us ahead wheather loan was issue or not,so we immediately dr
op it out
```

Out[71]:

```
0    Jan-2015
1    Jan-2015
2    Jan-2015
Name: issue_d, dtype: object
```

In [72]:

```
Data=Data.drop('issue_d',axis=1)
```

In [73]:

```
#This like a historic time stamp which we can apply lambda function for the years
Data['earliest_year']=Data['earliest_cr_line'].apply(lambda date:int(date[-4]))
```

In [74]:

```
Data['earliest_cr_line'].head(2)
```

Out[74]:

```
0    Jun-1990
1    Jul-2004
Name: earliest_cr_line, dtype: object
```

In [75]:

```
Data=Data.drop('earliest_cr_line',axis=1)
```

In [76]:

```
Data.head(2)
```

Out[76]:

	loan_amnt	term	int_rate	installment	sub_grade	annual_inc	verification_status	loan_status	purpose	dti	...	05113
0	10000.0	36	11.44	329.48	B4	117000.0	Not Verified	Fully Paid	vacation	26.24	...	0
1	8000.0	36	11.99	265.68	B5	65000.0	Not Verified	Fully Paid	debt_consolidation	22.05	...	1

2 rows × 33 columns

In [77]:

```
Data.select_dtypes(['object']).columns
```

Out[77]:

```
Index(['sub_grade', 'verification_status', 'loan_status', 'purpose',
       'initial_list_status', 'application_type'],
      dtype='object')
```

In [78]:

```
Dummies=pd.get_dummies(Data[['verification_status', 'purpose', 'initial_list_status',
'application_type','sub_grade']],drop_first=True)
Data=Data.drop(['verification_status', 'purpose',
'initial_list_status','application_type','sub_grade'],axis=1)
Data=pd.concat([Data,Dummies],axis=1)
```

In [79]:

```
Data=Data.drop('loan_status',axis=1)
```

In [80]:

```
Data.head(2)
```

Out[80]:

	loan_amnt	term	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	...	sub_grade_F1	sub_grade_F2
0	10000.0	36	11.44	329.48	117000.0	26.24	16.0	0.0	36369.0	41.8	...	0	0
1	8000.0	36	11.99	265.68	65000.0	22.05	17.0	0.0	20131.0	53.3	...	0	0

2 rows × 79 columns

In [81]:

```
Data.to_csv(r'C:\Users\chumj\Downloads\Data.csv',index=False,header=True)
```

WE ARE GOOD TO START IMPLEMENTING OUR MEACHINE LEARNING MODELS AND ANN.

In [82]:

```
#Setting X and y variables for our different models,that is the features(X),and label(y)
X=Data.drop('repaid_loans',axis=1)
y=Data['repaid_loans']
```

In [83]:

```
from sklearn.model_selection import train_test_split
```

In [84]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

In [85]:

```
#lets begins with logistics regression model.
from sklearn.linear_model import LogisticRegression
```

In [86]:

```
model1 = LogisticRegression()
```

```
model1.fit(X_train,y_train)
```

Out[86]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [87]:

```
predictions = model1.predict(X_test)
```

In [88]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [89]:

```
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
print(accuracy_score(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.26      0.00      0.00     23363
           1       0.80      1.00      0.89     95203

    accuracy                           0.80    118566
   macro avg       0.53      0.50      0.45    118566
weighted avg       0.70      0.80      0.72    118566

[[   37 23326]
 [  107 95096]]
0.8023632407266839
```

In [90]:

```
#using Decision tress and random forest.
from sklearn.tree import DecisionTreeClassifier
```

In [91]:

```
dtree = DecisionTreeClassifier()
```

In [92]:

```
dtree.fit(X_train,y_train)
```

Out[92]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [93]:

```
predictions1 = dtree.predict(X_test)
```

In [94]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [95]:

```
print(classification_report(y_test,predictions1))
print(confusion_matrix(y_test,predictions1))
print(accuracy_score(y_test,predictions1))
```

```
              precision    recall  f1-score   support

           0       0.56      0.59      0.58     23363
           1       0.90      0.89      0.89     95203

    accuracy                           0.83    118566
   macro avg       0.73      0.74      0.73    118566
weighted avg       0.83      0.83      0.83    118566

[[13773  9590]
 [10643 84560]]
0.829352428183459
```

In [97]:

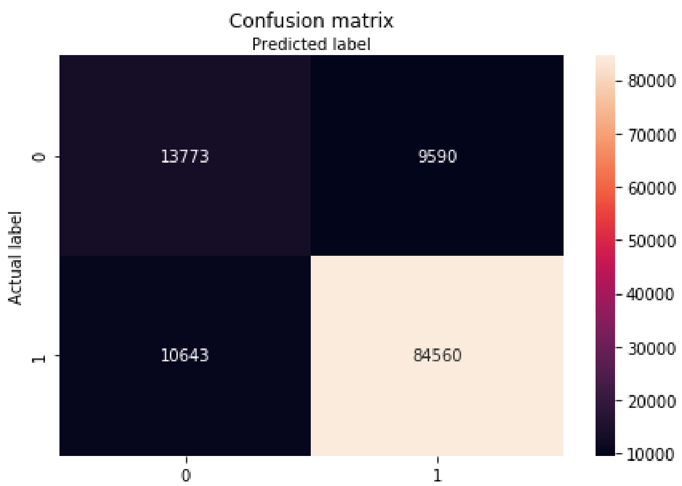
```
cm3=confusion_matrix(y_test,predictions1)
```

In [98]:

```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cm3), annot=True ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[98]:

```
Text(0.5, 257.44, 'Predicted label')
```



In [99]:

```
#Given a customer below we you offer loan to the person or not
import random
random_ind=random.randint(0,len(Data))
```

In [100]:

```
new_person4=Data.drop('repaid_loans',axis=1).iloc[random_ind]
```

In [101]:

```
new_person4
```

Out[101]:

```
loan_amnt        20000.00
term                36.00
int_rate            11.48
installment        659.37
annual_inc      105000.00
                  ...
sub_grade_G1         0.00
sub_grade_G2         0.00
sub_grade_G3         0.00
sub_grade_G4         0.00
sub_grade_G5         0.00
Name: 146375, Length: 78, dtype: float64
```

In [102]:

```
dtree.predict(new_person4.values.reshape(1,78))
```

Out[102]:

```
array([1], dtype=int64)
```

In [103]:

```
#check if this person end up paying the loan
Data.iloc[random_ind]['repaid_loans']
```

Out[103]:

```
1.0
```

In [ ]:

```

```