Problem Statement : To Build a logistic regression model for loan status for LoanTap

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge,Lasso,ElasticNet
import sklearn.metrics as metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn import decomposition
from  scipy import stats
from sklearn import decomposition
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
from imblearn.over_sampling import SMOTE
from sklearn.impute import KNNImputer
from category_encoders import TargetEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import precision_score,recall_score,f1_score,fbeta_score
from imblearn.metrics import geometric_mean_score
from scipy.stats import pearsonr,spearmanr,kendalltau
```

In [2]:

```
df=pd.read_csv('logistic_regression.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	M
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	M

5 rows × 27 columns

In [4]:

```
df.shape
```

Out[4]:

```
(396030, 27)
```

In [5]:

```
df.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
```

In [6]:

```
df.isnull().sum()
# There are missing values in emp_title, emp_length, title,revol_util,mort_acc,pub_rec_bank
```

Out[6]:

```
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
dtype: int64
```

In [7]:

```
nulls=df.isnull().sum().sum()
missing_values=nulls/len(df)
print('Missing values are : ',missing_values*100,'%')
# There are 20.61% missing values
```

```
Missing values are :  20.601722091760724 %
```

In [8]:

```
df.drop_duplicates(keep='first', inplace=True, ignore_index=True)
```

In [9]:

```
df.columns
```

Out[9]:

```
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 [10]:

```
df.describe()
# There are lot of outliners in annual_inc, loan_amnt
```

Out[10]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_a
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.0000
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.3111
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.1376
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.0000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.0000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.0000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.0000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.0000

In [11]:

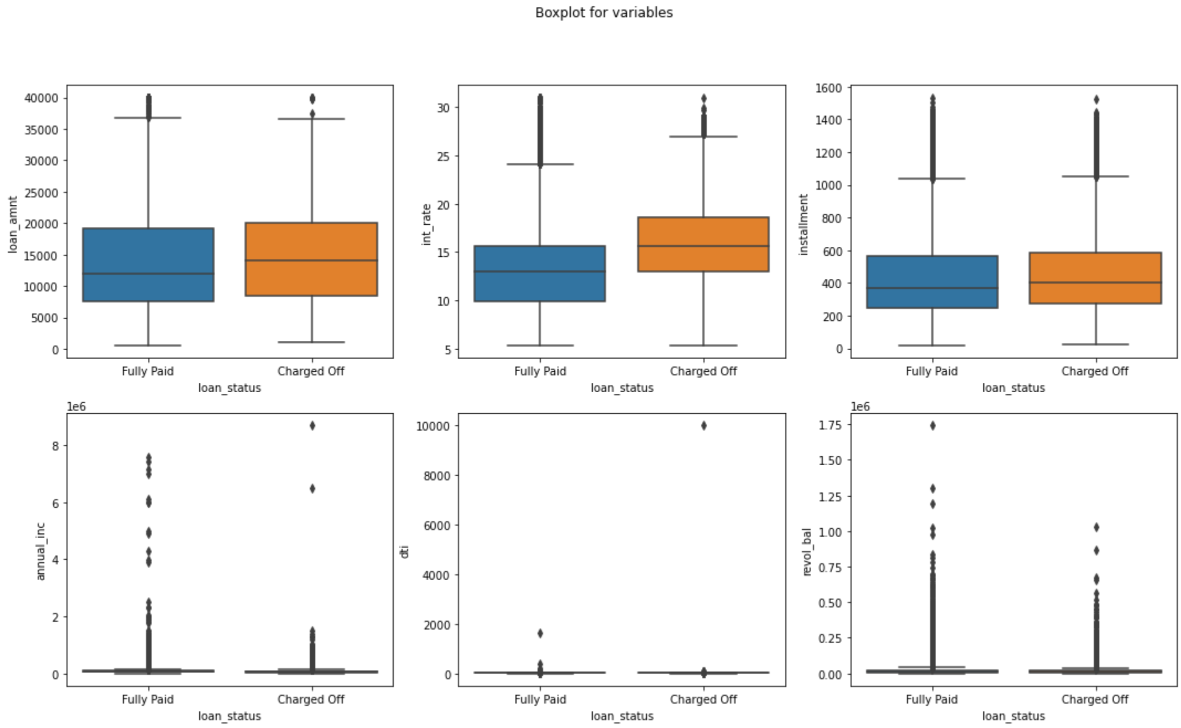
```
df.describe(include=object)
# term of 36 months is mostly taken by people
# Most purpose of the loan purpose is 'dept_consolidation'
```

Out[11]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status
count	396030	396030	396030	373103	377729	396030	39603
unique	2	7	35	173105	11	6	3
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verified
freq	302005	116018	26655	4389	126041	198348	13956

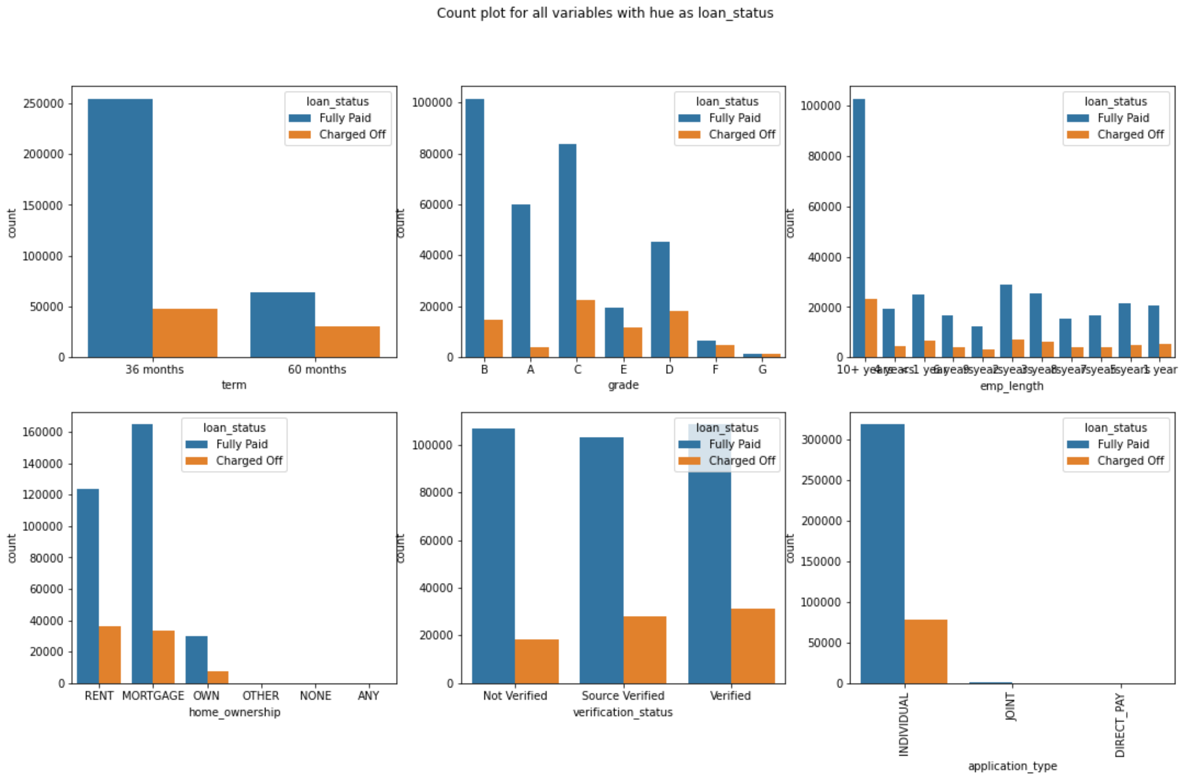
In [12]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

fig.suptitle('Boxplot for variables')
sns.boxplot(ax=axes[0, 0], data=df, y='loan_amnt',x='loan_status')
sns.boxplot(ax=axes[0, 1], data=df, y='int_rate',x='loan_status')
sns.boxplot(ax=axes[0, 2], data=df, y='installment',x='loan_status')
sns.boxplot(ax=axes[1, 0], data=df, y='annual_inc',x='loan_status')
sns.boxplot(ax=axes[1, 1], data=df, y='dti',x='loan_status')
sns.boxplot(ax=axes[1, 2], data=df, y='revol_bal',x='loan_status')

plt.show()
# Below is the boxplot for all the variables
# When the median interest rates are low people are able to pay interest fully.
# There are outliners in annual_inc. That needs to removed
```



In [13]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
plt.xticks(rotation = 90)
fig.suptitle('Count plot for all variables with hue as loan_status')
sns.countplot(ax=axes[0, 0], data=df, x='term',hue=df['loan_status'])
sns.countplot(ax=axes[0, 1], data=df, x='grade',hue=df['loan_status'])
sns.countplot(ax=axes[0, 2], data=df, x='emp_length',hue=df['loan_status'] )
sns.countplot(ax=axes[1, 0], data=df, x='home_ownership',hue=df['loan_status'] )
sns.countplot(ax=axes[1, 1], data=df, x='verification_status',hue=df['loan_status'])
sns.countplot(ax=axes[1, 2], data=df, x='application_type',hue=df['loan_status'])
plt.show()
# There are more number for count for term 36 months
# Grade 2 and 3 has the highest count of loan appliers
# Higest number of people who take loan have their homes in Mortgage
# More number of application_type are from 'INDIVIDUAL' than joint and direct_pay
```

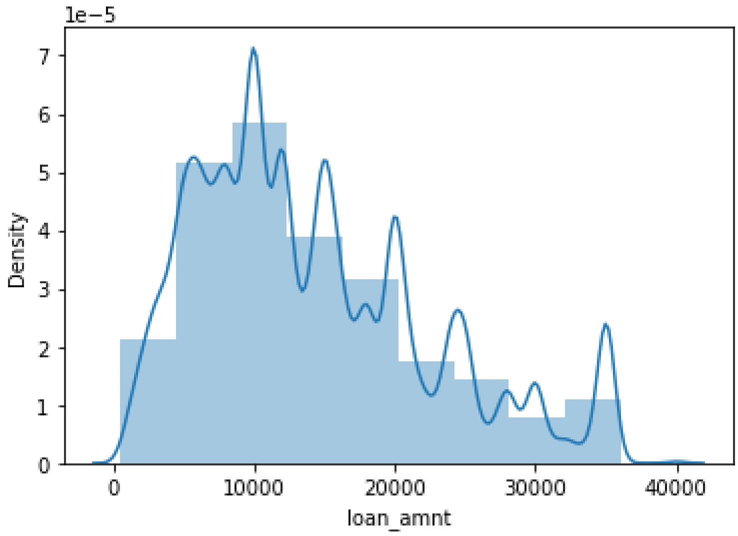


In [14]:

```
sns.distplot(df['loan_amnt'],bins=10)
# Distribution plot of loan_amount
```

Out[14]:

```
<AxesSubplot:xlabel='loan_amnt', ylabel='Density'>
```

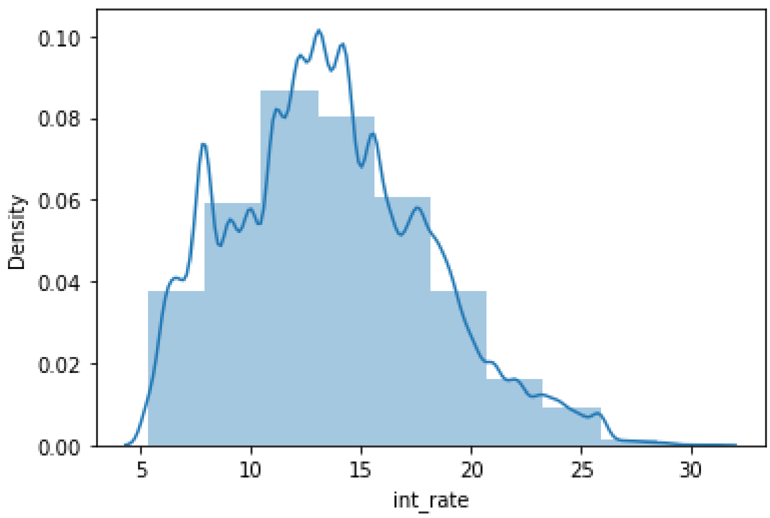


In [15]:

```
sns.distplot(df['int_rate'],bins=10)
# Distribution plot of int_rate
# There are more application with interest rate between 10%-20%
```

Out[15]:

```
<AxesSubplot:xlabel='int_rate', ylabel='Density'>
```

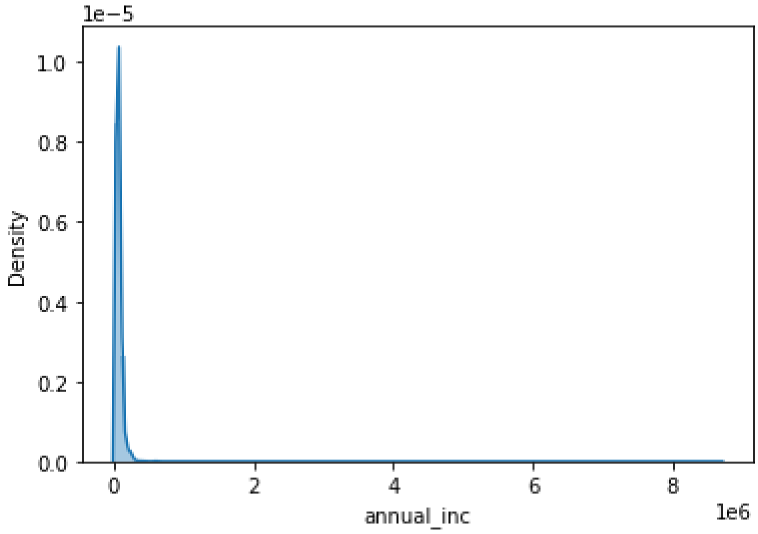


In [16]:

```
sns.distplot(df['annual_inc'],bins=100)
# There are more number of applications from annual income 0-1
```

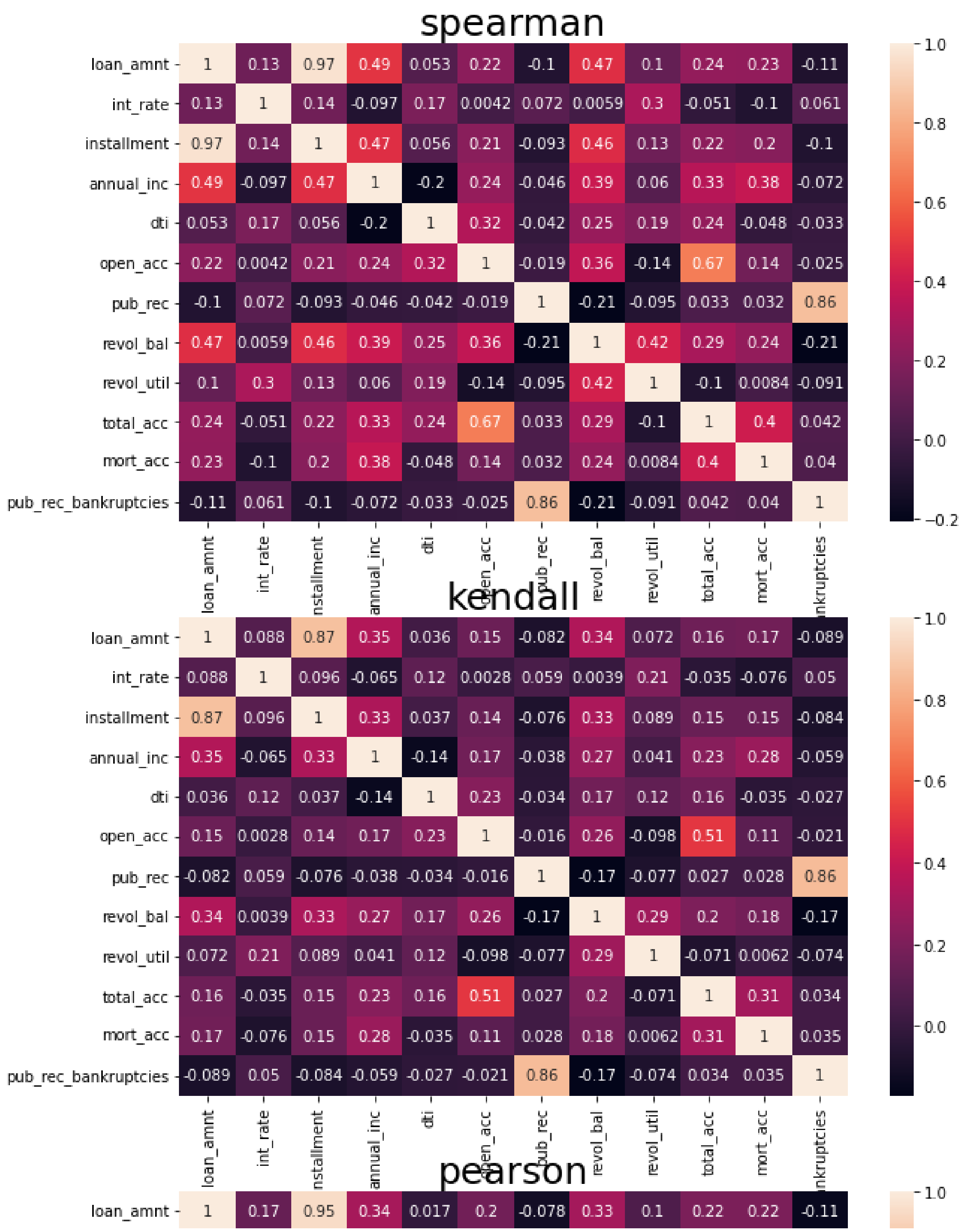
Out[16]:

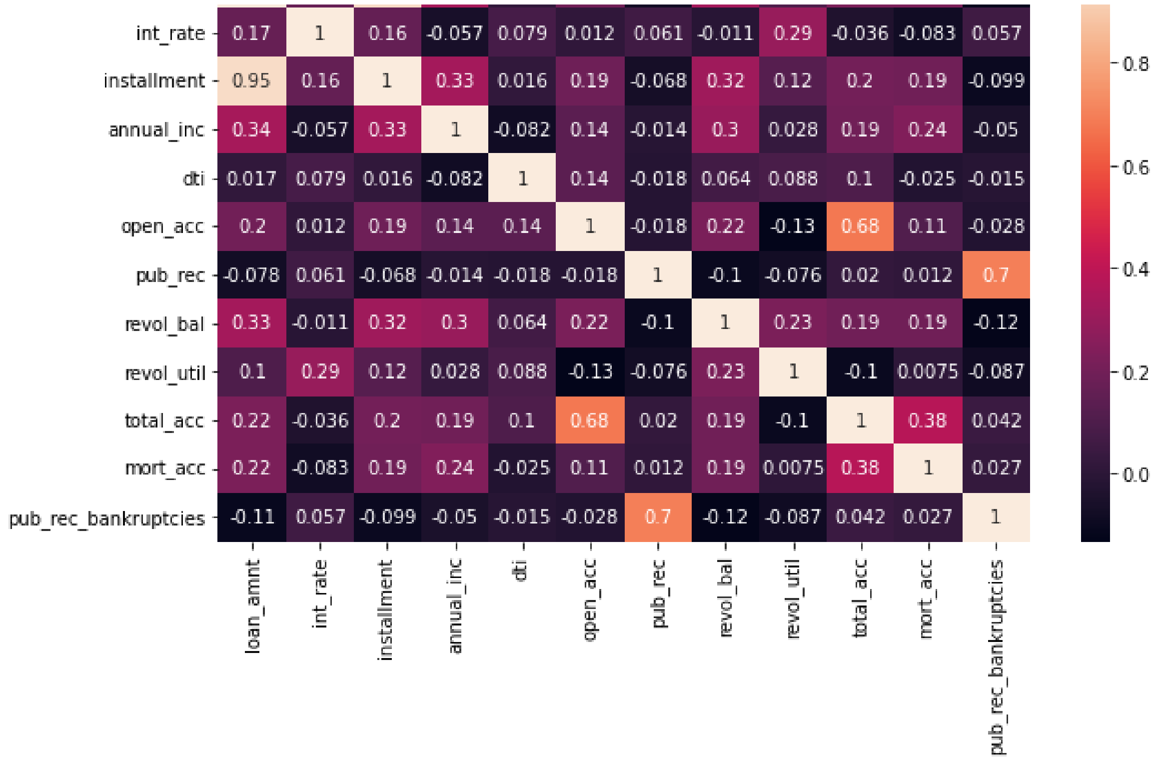
```
<AxesSubplot:xlabel='annual_inc', ylabel='Density'>
```



In [17]:

```
fig, axes = plt.subplots(3, 1, figsize=(10, 20))


sns.heatmap(df.corr(method ='spearman'),ax=axes[0],annot=True)
axes[0].set_title('spearman',fontsize=25)
sns.heatmap(df.corr(method ='kendall'),ax=axes[1],annot=True)
axes[1].set_title('kendall',fontsize=25)
sns.heatmap(df.corr(method ='pearson'),ax=axes[2],annot=True)
axes[2].set_title('pearson',fontsize=25)

plt.show()
# Below is the spearman, kendall and pearson correlations
# Loan amount and installment has high correlation of 0.97
# Total_acc and open_acc has correleation of 0.86
```





In [18]:

```
df['loan_status'].value_counts(normalize=True)
# 80.7 % of applicants pay their loan fully
# 19.29 % of applicants do not pay their loan fully
```

Out[18]:

```
Fully Paid     0.803871
Charged Off    0.196129
Name: loan_status, dtype: float64
```

In [19]:

```
pd.crosstab(index=df['grade'], columns=df['loan_status'], margins=True,normalize='index')
# Below is the conditional probability for all the grades vs loan_status
# In A grade there are 93.7 % who pay the loan comlpetely
# A grade has the highest 'Fully Paid' count
```

Out[19]:

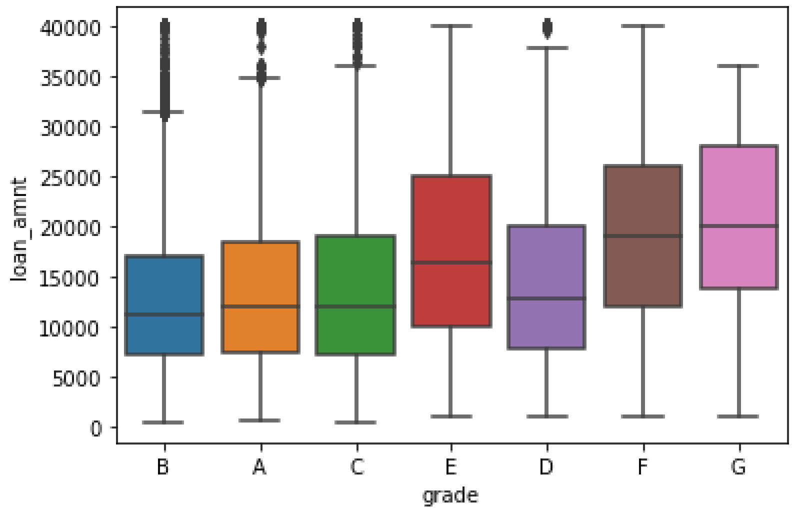
loan_status	Charged Off	Fully Paid
grade		
A	0.062879	0.937121
B	0.125730	0.874270
C	0.211809	0.788191
D	0.288678	0.711322
E	0.373634	0.626366
F	0.427880	0.572120
G	0.478389	0.521611
All	0.196129	0.803871

In [20]:

```
sns.boxplot(data=df, y='loan_amnt',x='grade')
# Grade G has higher median loan amount than other grades
```

Out[20]:

```
<AxesSubplot:xlabel='grade', ylabel='loan_amnt'>
```

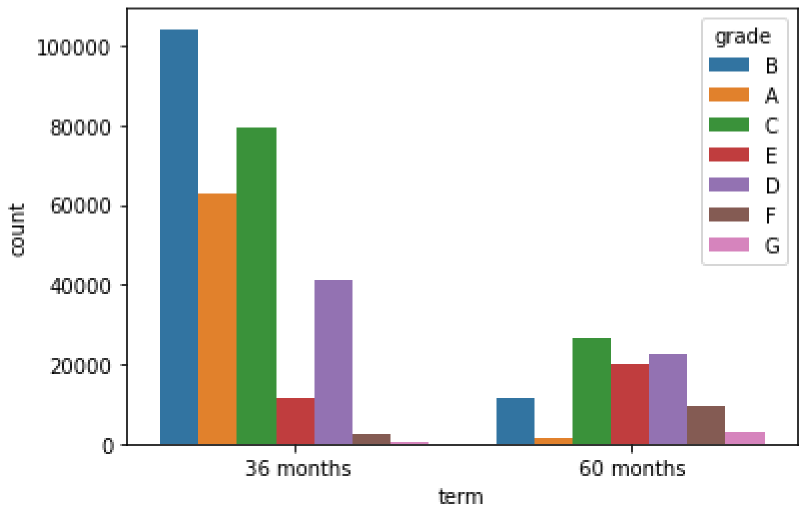


In [21]:

```
sns.countplot(data=df, x='term',hue='grade')
# Grade B has higher count of loan whose term is 36 months
# Grade C has higher count of loan whose term is 60 months
```

Out[21]:

```
<AxesSubplot:xlabel='term', ylabel='count'>
```

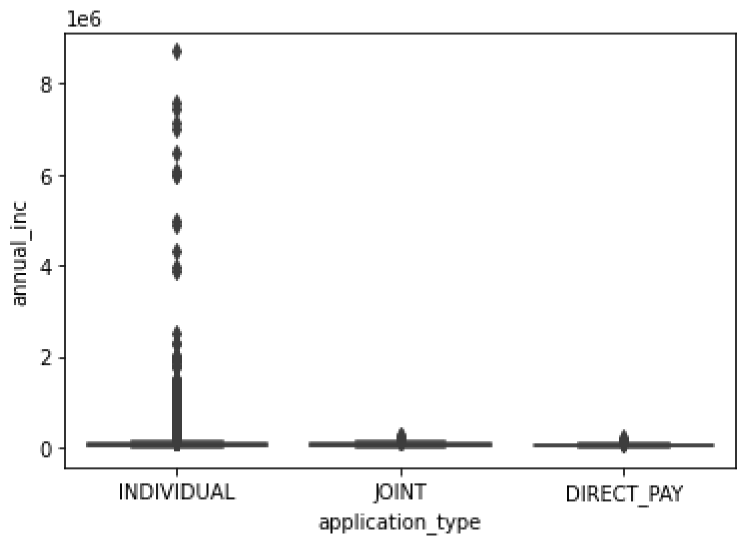


In [22]:

```
sns.boxplot(data=df, y='annual_inc',x='application_type')
```

Out[22]:

```
<AxesSubplot:xlabel='application_type', ylabel='annual_inc'>
```



In [23]:

```
def outliners(x,col):
    Q1 = np.percentile(x[col], 25)
    Q3 = np.percentile(x[col], 75)
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR
    lower = Q1 - 1.5*IQR
    #print(upper,lower)
    ls=list(x.iloc[((x[col]<lower) | (x[col]>upper)).values].index)
    return ls
```

In [24]:

```
len(outliners(df,'loan_amnt'))/len(df)
# There are 0.04% outliners of loan_amount
```

Out[24]:

```
0.0004822866954523647
```

In [25]:

```
df.drop(outliners(df,'int_rate'),axis=0,inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
# Outliners are removed
```

In [26]:

```
df['term'].value_counts(normalize=True)
```

Out[26]:

```
 36 months    0.768494
 60 months    0.231506
Name: term, dtype: float64
```

In [27]:

```
df['term']=df['term'].apply(lambda x : 36 if x==' 36 months' else 60)
# Changing months to integer
```

In [28]:

```
df['term'].value_counts(normalize=True)
```

Out[28]:

```
36    0.768494
60    0.231506
Name: term, dtype: float64
```

In [29]:

```
df['int_rate'].value_counts(normalize=True)
```

Out[29]:

```
10.99    0.031640
12.99    0.024556
15.61    0.023837
11.99    0.021879
8.90     0.020443
           ...
14.28    0.000003
22.94    0.000003
18.72    0.000003
18.36    0.000003
24.59    0.000003
Name: int_rate, Length: 524, dtype: float64
```

In [30]:

```
len(outliners(df,'int_rate'))/len(df)*100
# There are 5 % Outliners in 'int_rate'
```

Out[30]:

```
0.05124243791634481
```

In [31]:

```
df.drop(outliners(df,'int_rate'),axis=0,inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
# Outliners are removed
```

In [32]:

```
df['installment'].value_counts(normalize=True)
```

Out[32]:

```
327.34    0.002469
332.10    0.002018
491.01    0.001877
336.90    0.001750
392.81    0.001742
            ...
37.43     0.000003
116.00    0.000003
826.69    0.000003
480.14    0.000003
572.44    0.000003
Name: installment, Length: 54495, dtype: float64
```

In [33]:

```
len(outliners(df,'installment'))/len(df)*100
```

Out[33]:

```
2.8088110760817444
```

In [34]:

```
df.drop(outliners(df,'installment'),axis=0,inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
# Outliners are removed
```

In [35]:

```
df['grade'].value_counts(normalize=True)
```

Out[35]:

```
B    0.298260
C    0.268310
A    0.166612
D    0.159264
E    0.079215
F    0.026782
G    0.001556
Name: grade, dtype: float64
```

In [36]:

```
temp_dict = {'A': 1, 'B': 2, 'C': 3,'D': 4, 'E': 5, 'F': 6,'G': 7}
# mapping values in column from dictionary
# Converting 'object' to 'integer'
```

In [37]:

```
for i in temp_dict.keys():
    d=df[df['grade']==i]
    ans=d['loan_status'].value_counts()
    print(ans[1]/ans[0],i)
```

```
0.06729653850680026 A
0.1445937235628248 B
0.2713514723438122 C
0.4090740224760843 D
0.6042519266542652 E
0.7558499655884378 F
0.6846590909090909 G
```

In [38]:

```
df['grade'] = df['grade'].map(temp_dict)
```

In [39]:

```
df['grade'].value_counts(normalize=True)
```

Out[39]:

```
2    0.298260
3    0.268310
1    0.166612
4    0.159264
5    0.079215
6    0.026782
7    0.001556
Name: grade, dtype: float64
```

In [40]:

```
df['sub_grade'].value_counts()
```

Out[40]:

```
B3    26081
B4    25128
C1    23041
B2    21973
C2    21787
B5    21626
C3    20375
C4    19487
B1    18841
A5    18124
C5    17547
A4    15558
D1    15374
D2    13335
D3    11634
D4    11119
A3    10545
A1     9709
A2     9550
D5     9224
E1     7556
E2     7106
E3     5968
E4     5182
E5     4372
F1     3410
F2     2647
F3     2197
F4     1619
F5      332
G1      242
G2      147
G3       85
G4       70
G5       49
Name: sub_grade, dtype: int64
```

In [41]:

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

Out[41]:

```
10+ years    120359
2 years       34577
< 1 year      30639
3 years       30544
5 years       25618
1 year        24975
4 years       23119
6 years       20127
7 years       20074
8 years       18416
9 years       14744
Name: emp_length, dtype: int64
```

In [42]:

```
temp_dict = {'< 1 year':0.5 ,'1 year': 1,'2 years': 2,'3 years': 3,'4 years': 4,'5 years':
            '8 years': 8,'9 years': 9,'10+ years': 10}
df['emp_length'] = df['emp_length'].map(temp_dict)
df['emp_length'].value_counts(normalize=True)
# mapping values in column from dictionary
# Converting 'object' to 'integer'
```

Out[42]:

```
10.0    0.331392
2.0     0.095203
0.5     0.084360
3.0     0.084099
5.0     0.070536
1.0     0.068765
4.0     0.063655
6.0     0.055417
7.0     0.055271
8.0     0.050706
9.0     0.040596
Name: emp_length, dtype: float64
```

In [43]:

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

Out[43]:

```
MORTGAGE    189404
RENT        155274
OWN          36220
OTHER          111
NONE            29
ANY              2
Name: home_ownership, dtype: int64
```

In [44]:

```
df['annual_inc'].value_counts()
```

Out[44]:

```
60000.00     15110
50000.00     13162
65000.00     11140
40000.00     10533
70000.00     10384
             ...
41349.00         1
62910.00         1
36489.00         1
105654.00        1
31789.88         1
Name: annual_inc, Length: 26298, dtype: int64
```

In [45]:

```
df['verification_status'].value_counts()
```

Out[45]:

```
Verified           129611
Source Verified    126770
Not Verified       124659
Name: verification_status, dtype: int64
```

In [46]:

```
temp_dict = {'Verified':1 ,'Source Verified': 1,'Not Verified': 0}
df['verification_status'] = df['verification_status'].map(temp_dict)
df['verification_status'].value_counts(normalize=True)
# mapping values in column from dictionary
# Converting 'object' to 'integer'
```

Out[46]:

```
1    0.672845
0    0.327155
Name: verification_status, dtype: float64
```

In [47]:

```
df['issue_d'].value_counts()
```

Out[47]:

```
Oct-2014    14291
Jul-2014    12129
Jan-2015    11204
Dec-2013    10154
Nov-2013    10093
            ...
Jul-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: issue_d, Length: 115, dtype: int64
```

In [48]:

```
df['month']=df["issue_d"].str.split("-",n = 1, expand = True)[0]
df['year']=df["issue_d"].str.split("-",n = 1, expand = True)[1]
df.drop('issue_d',axis=1,inplace=True)
# Splitting 'issue_d' to month and year
```

In [49]:

```
df['month'].value_counts()
```

Out[49]:

```
Oct    40674
Jul    38313
Jan    33332
Nov    32831
Apr    32043
Aug    31379
May    30764
Mar    30598
Jun    29070
Dec    27895
Feb    27549
Sep    26592
Name: month, dtype: int64
```

In [50]:

```
df['year'].value_counts()
# Mapping values in column from dictionary
# Converting 'object' to 'integer'
```

Out[50]:

```
2014    98595
2013    94544
2015    90078
2012    40146
2016    25892
2011    17266
2010     9258
2009     3826
2008     1240
2007      195
Name: year, dtype: int64
```

In [51]:

```
df['loan_status']=df['loan_status'].apply(lambda x : 0 if x=='Fully Paid' else 1)
# mapping values
# Converting 'object' to 'integer'
```

In [52]:

```
df['purpose'].value_counts()
```

Out[52]:

```
debt_consolidation    225231
credit_card            80484
home_improvement       22914
other                  20461
major_purchase          8533
small_business          5147
car                     4652
medical                 4086
moving                  2776
vacation                2424
house                   1963
wedding                 1789
renewable_energy         323
educational              257
Name: purpose, dtype: int64
```

In [53]:

```
df['title'].value_counts()
```

Out[53]:

```
Debt consolidation          145439
Credit card refinancing      49671
Home improvement             14398
Other                        12350
Debt Consolidation           11179
                             ...
Paydown Credit Cards             1
Debtconsolidation2013            1
Station Park Honda               1
2nd debt consoidation            1
Toxic Debt Payoff                1
Name: title, Length: 47924, dtype: int64
```

In [54]:

```
df['dti'].value_counts()
```

Out[54]:

```
14.40    302
0.00     301
19.20    295
16.80    294
18.00    292
        ...
47.98      1
59.18      1
48.37      1
45.71      1
55.53      1
Name: dti, Length: 4234, dtype: int64
```

In [55]:

```
len(outliners(df,'dti'))/len(df)
```

Out[55]:

```
0.0006298551333193366
```

In [56]:

```
df.drop(outliners(df,'dti'),axis=0,inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
# Outliners are removed
```

In [57]:

```
df['earliest_cr_line'].value_counts()
```

Out[57]:

```
Oct-2000    2935
Aug-2000    2836
Oct-2001    2803
Aug-2001    2780
Nov-2000    2657
            ... 
Jan-1953       1
Jul-1955       1
Oct-1950       1
Aug-1951       1
Aug-1959       1
Name: earliest_cr_line, Length: 683, dtype: int64
```

In [58]:

```
df['ear_month']=df["earliest_cr_line"].str.split("-",n = 1, expand = True)[0]
df['ear_year']=df["earliest_cr_line"].str.split("-",n = 1, expand = True)[1]
df.drop('earliest_cr_line',axis=1,inplace=True)
# Splitting 'earliest_cr_line' to month and year
```

In [59]:

```
df['open_acc'].value_counts()
```

Out[59]:

```
9.0     35593
10.0    34193
8.0     34105
11.0    31450
7.0     30492
        ... 
55.0        2
76.0        2
57.0        1
58.0        1
90.0        1
Name: open_acc, Length: 61, dtype: int64
```

In [60]:

```
len(outliners(df,'open_acc'))/len(df)
# Outliners are moved
```

Out[60]:

```
0.0249343487394958
```

In [61]:

```
df.drop(outliners(df,'open_acc'),axis=0,inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
```

In [62]:

```
df['pub_rec'].value_counts()
```

Out[62]:

```
0.0     316447
1.0      47382
2.0       5118
3.0       1426
4.0        480
5.0        216
6.0        112
7.0         50
8.0         31
9.0         11
10.0        11
11.0         6
13.0         4
12.0         4
19.0         2
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: pub_rec, dtype: int64
```

In [63]:

```
df['revol_bal'].value_counts()
```

Out[63]:

```
0.0        2054
5655.0       39
7792.0       38
6095.0       37
3953.0       36
           ...
43378.0       1
46191.0       1
41586.0       1
24062.0       1
28053.0       1
Name: revol_bal, Length: 52312, dtype: int64
```

In [64]:

```
len(outliners(df,'revol_bal'))/len(df)
```

Out[64]:

```
0.05280025854755525
```

In [65]:

```
df['revol_util'].value_counts()
```

Out[65]:

```
0.00      2108
53.00      712
60.00      700
55.00      690
54.00      689
          ...
109.30       1
146.10       1
0.75         1
111.10       1
128.10       1
Name: revol_util, Length: 1215, dtype: int64
```

In [66]:

```
len(outliners(df,'revol_util'))/len(df)
```

Out[66]:

```
0.0
```

In [67]:

```
df['total_acc'].value_counts()
```

Out[67]:

```
21.0     13777
20.0     13773
22.0     13737
19.0     13432
23.0     13424
         ...
117.0        1
100.0        1
96.0         1
116.0        1
101.0        1
Name: total_acc, Length: 107, dtype: int64
```

In [68]:

```
len(outliners(df,'total_acc'))/len(df)
```

Out[68]:

```
0.01562058146267893
```

In [69]:

```
df.drop(outliners(df,'total_acc'),axis=0,inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
# Outliners are removed
```

In [70]:

```
df['initial_list_status'].value_counts()
```

Out[70]:

```
f    221315
w    144190
Name: initial_list_status, dtype: int64
```

In [71]:

```
df['initial_list_status']=df['initial_list_status'].apply(lambda x : 0 if x=='f' else 1)
# Encoding 'initial_list_status' column
```

In [72]:

```
df['application_type'].value_counts()
```

Out[72]:

```
INDIVIDUAL    365149
JOINT            315
DIRECT_PAY        41
Name: application_type, dtype: int64
```

In [73]:

```
df['mort_acc'].value_counts()
```

Out[73]:

```
0.0     131597
1.0      55833
2.0      45627
3.0      34268
4.0      24883
5.0      16063
6.0       9592
7.0       5183
8.0       2648
9.0       1363
10.0       714
11.0       389
12.0       207
13.0       103
14.0        81
15.0        45
16.0        28
17.0        12
19.0        11
18.0        11
20.0         6
22.0         4
24.0         3
26.0         2
25.0         2
21.0         2
28.0         1
31.0         1
23.0         1
32.0         1
27.0         1
Name: mort_acc, dtype: int64
```

In [74]:

```
outliners(df,'mort_acc')
```

Out[74]:

```
[]
```

In [75]:

```
df['pub_rec_bankruptcies'].value_counts()
```

Out[75]:

```
0.0    322746
1.0     40102
2.0      1692
3.0       326
4.0        73
5.0        30
6.0         6
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

In [76]:

```
outliners(df,'pub_rec_bankruptcies')
```

Out[76]:

```
[]
```

In [77]:

```
df['address'].value_counts()
```

Out[77]:

```
USCGC Smith\r\nFPO AE 70466                                    8
USNS Johnson\r\nFPO AE 05113                                   8
USS Smith\r\nFPO AP 70466                                      8
USCGC Jones\r\nFPO AE 22690                                    6
USCGC Miller\r\nFPO AA 22690                                   6
                                                              ..
6721 Traci Forest\r\nNorth Triciaberg, KS 00813                1
747 Patricia Springs Suite 292\r\nWest Lynnfort, DC 22690      1
7670 Adams Lights\r\nBoydbury, WY 22690                        1
777 Jeremy Ramp Suite 840\r\nWest Stacey, MT 29597             1
787 Michelle Causeway\r\nBriannaton, AR 48052                  1
Name: address, Length: 363502, dtype: int64
```

In [78]:

```
df['pincode']=df['address'].apply(lambda x: x[-5:])
df['pincode']=df['pincode'].astype('int64')
# Extracting pincode
```

In [79]:

```
df.drop('address',axis=1,inplace=True)
```

In [80]:

```
df['year']=df['year'].astype('int64')
df['ear_year']=df['ear_year'].astype('int64')
# Extracting year and month from 'ear_year'
```

In [81]:

```
temp_dict = {'Jan':1 ,'Feb': 2,'Mar': 3,'Apr': 4,'May': 5,'Jun': 6,'Jul': 7,'Aug': 8,
            'Sep': 9,'Oct': 10,'Nov': 11,'Dec':12}
df['ear_month'] = df['ear_month'].map(temp_dict)
df['month'] = df['month'].map(temp_dict)
# Mapping month with month number
```

In [82]:

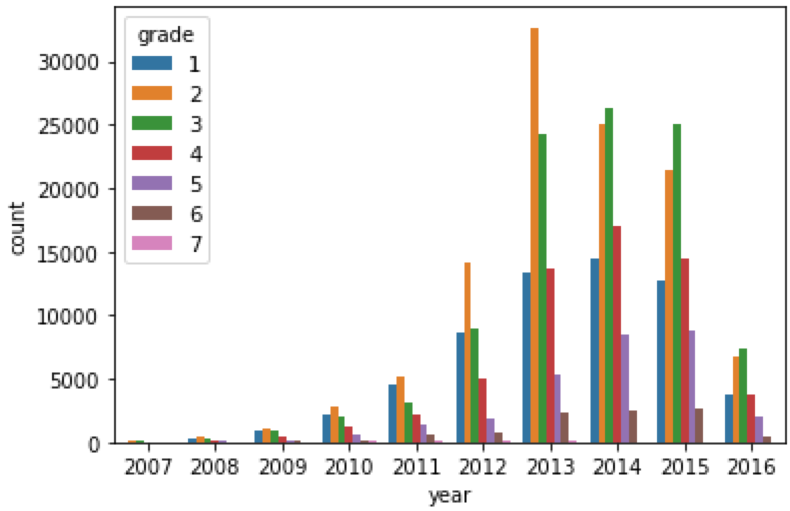
```
df['pub_rec']=df['pub_rec'].apply(lambda x : 1 if x>1 else 0)
df['mort_acc']=df['mort_acc'].apply(lambda x : 1 if x>1 else 0)
df['pub_rec_bankruptcies']=df['pub_rec_bankruptcies'].apply(lambda x : 1 if x>1 else 0)
# Encoding
```

In [83]:

```
sns.countplot(data=df, x='year',hue='grade')
# There are more number of application from the year 2013 to 2015
```

Out[83]:

<AxesSubplot:xlabel='year', ylabel='count'>

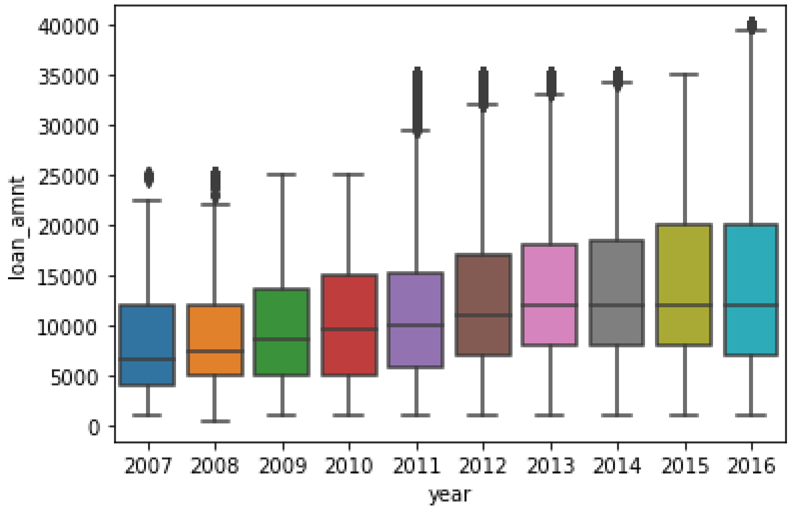


In [84]:

```
sns.boxplot(data=df, y='loan_amnt',x='year')
# There is an increase in median loan amount from 2007 to 2013
```

Out[84]:

```
<AxesSubplot:xlabel='year', ylabel='loan_amnt'>
```

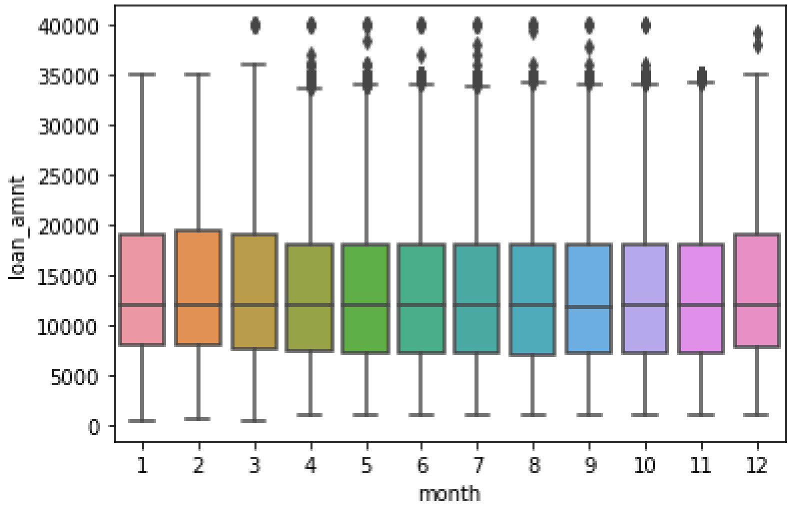


In [85]:

```
sns.boxplot(data=df, y='loan_amnt',x='month')
```

Out[85]:

```
<AxesSubplot:xlabel='month', ylabel='loan_amnt'>
```



In [86]:

```
pearsonr(df['loan_amnt'],df['installment']),spearmanr(df['loan_amnt'],df['installment']),ke
# Correlations between loan amount and installment:
# 1. Pearson corr of 0.95
```

Out[86]:

```
((0.9503389567461586, 0.0),
 SpearmanrResult(correlation=0.9658405979190868, pvalue=0.0),
 KendalltauResult(correlation=0.8624859664100726, pvalue=0.0))
```

In [87]:

```
pearsonr(df['pincode'],df['loan_status']),spearmanr(df['pincode'],df['loan_status']),kendal
# Correlations between pincode and loan_status:
# 1. Pearson corr of 0.346
```

Out[87]:

```
((0.34614298172409663, 0.0),
 SpearmanrResult(correlation=0.2964290525458191, pvalue=0.0),
 KendalltauResult(correlation=0.2564410872717571, pvalue=0.0))
```

In [88]:

```
X= df.drop('loan_status',axis=1)
y=df['loan_status']
X_train, X_test,y_train, y_test = train_test_split(X,y ,
                                    random_state=9,
                                    test_size=0.2,
                                    shuffle=True)
# Test and Train split for the data
```

In [89]:

```
X_train.shape
```

Out[89]:

```
(292404, 28)
```

In [90]:

```
ind_list=X_train[X_train.isnull().any(axis=1)].index
X_t=X_train.drop(ind_list)
y_t=y_train.drop(ind_list)
ls=[]
for i in X_t.columns:
    if X_t.dtypes[i]=='object':
        ls.append(i)
```

In [91]:

```
encoder = TargetEncoder(cols=ls,return_df=True,min_samples_leaf=1, smoothing=1.2,handle_mis
X_train = encoder.fit_transform(X=X_train,y=y_train)
X_test = encoder.transform(X_test)
# Target encoding based on target for all the objects
```

In [92]:

```
imputer = KNNImputer(n_neighbors=11,weights='distance')
X_train=imputer.fit_transform(X_train)
X_test=imputer.transform(X_test)
# KNN inputer n_neighbors as 11 and weights as distance
# This is done to fill the NA values with the nerest neighbours
```

In [93]:

```
sm=SMOTE(k_neighbors=17)
X_train,y_train=sm.fit_resample(pd.DataFrame(X_train),pd.DataFrame(y_train))
# As the data is imbalanced adding more data using SMOTE
```

In [94]:

```
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
# Scaling the data
```

In [95]:

```
l=np.arange(0.1,1,0.1)
for i in l:
    classifier = LogisticRegression(penalty='elasticnet',solver='saga',C=0.01,l1_ratio=i,cl
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_train)
    print(precision_score(y_train, y_pred)),print(recall_score(y_train, y_pred))
    print('------')
    y_pred = classifier.predict(X_test)
    print(precision_score(y_test, y_pred)),print(recall_score(y_test, y_pred))
    print('------')
    print('------')
# There are no improvements in precision and recall by changing l1_ratio from 0.1 to 1
```

```
0.7660003468909895
0.7489506931784458
------
0.378978751994625
0.6450114351057747
------
------
0.7659717464639719
0.7489422139314029
------
0.37902514799109954
0.6452258433390509
------
------
0.7659600300056802
0.7489210158137958
------
0.3791727902582406
0.6453687821612349
------
------
0.765898259154074
0.7488913384491457
------
0.37918871252204583
0.6453687821612349
------
------
0.7658691617960769
0.74887437995506
------
0.3791873058517337
0.6455831903945112
------
------
0.7658444368325434
0.7488404629668886
------
0.3791134989926125
0.6455117209834191
------
------
0.7658880713180043
0.7488701403315385
------
```

```
0.3791280265200789
0.6457261292166953
------
------
0.7658727577819113
0.7488616610844957
------
0.379139559286463 8
0.6455831903945112
------
------
0.7658852145560812
0.7488998176961886
------
0.3792075883488626
0.6457261292166953
------
------
```

In [96]:

```
classifier = LogisticRegression(penalty='l1',solver='liblinear',C=100,class_weight='balance
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_train)
print('Precision : ',precision_score(y_train, y_pred))
print('Recall : ',recall_score(y_train, y_pred))
print('F1_Score : ',f1_score(y_train, y_pred))
# Precision and Recall score are 0.766 and 0.750 for train data
```

```
Precision :  0.7660928013876843
Recall :  0.7489761309195744
F1_Score :  0.7574377773490257
```

In [97]:

```
y_pred = classifier.predict(X_train)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred)

print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[181931  53939]
 [ 59209 176661]]
```

In [98]:

```
tn,fp,fn,tp=cm[0][0],cm[0][1],cm[1][0],cm[1][1]
```

In [99]:

```
tn,fp,fn,tp
```

Out[99]:

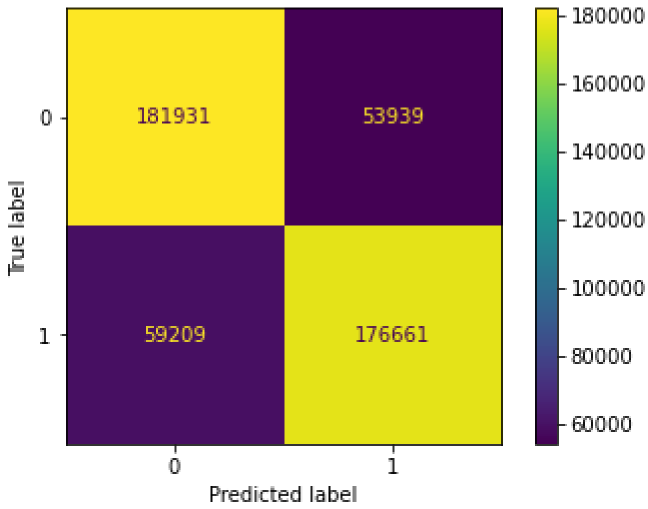
```
(181931, 53939, 59209, 176661)
```

In [100]:

```
ConfusionMatrixDisplay(cm).plot()
```

Out[100]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1d20feea
430>
```



In [101]:

```
print ("Accuracy : ", accuracy_score(y_train, y_pred))
print ("Precision Score : ", precision_score(y_train, y_pred))
print ("Recall Score : ", recall_score(y_train, y_pred))
print ("F1 Score : ",f1_score(y_train, y_pred))
print ("G-mean Score : ",geometric_mean_score(y_train, y_pred))
print ("F0.5 Score : ",fbeta_score(y_train, y_pred,beta=0.5))
print ("F2 Score : ",fbeta_score(y_train, y_pred,beta=2))
```

```
Accuracy :  0.7601475388985458
Precision Score :  0.7660928013876843
Recall Score :  0.7489761309195744
F1 Score :  0.7574377773490257
G-mean Score :  0.7600654449041107
F0.5 Score :  0.7626071641327152
F2 Score :  0.7523380008176616
```

In [102]:

```
y_pred = classifier.predict(X_test)
print ("Accuracy : ", accuracy_score(y_test, y_pred))
print ("Precision Score : ", precision_score(y_test, y_pred))
print ("Recall Score : ", recall_score(y_test, y_pred))
print ("F1 Score : ",f1_score(y_test, y_pred))
print ("G-mean Score : ",geometric_mean_score(y_test, y_pred))
print ("F0.5 Score : ",fbeta_score(y_test, y_pred,beta=0.5))
print ("F2 Score : ",fbeta_score(y_test, y_pred,beta=2))
# Precision and recall are lower for test data when compared with train data
```

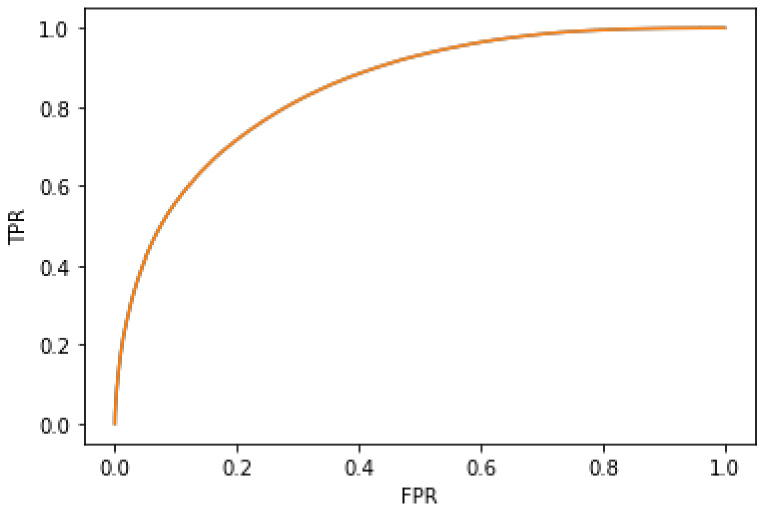
```
Accuracy :  0.7297027400445958
Precision Score :  0.37889033558738294
Recall Score :  0.6447255574614065
F1 Score :  0.47728896060950765
G-mean Score :  0.6952890863064117
F0.5 Score :  0.41294356758340356
F2 Score :  0.5653885205008963
```

In [103]:

```
fpr,tpr,thres=roc_curve(y_train,classifier.predict_proba(X_train)[:,1])
```

In [104]:

```
plt.plot(fpr,tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.plot(fpr,tpr)
plt.show()
```



In [105]:

```
roc_auc_score(y_train,classifier.predict_proba(X_train)[:,1])
# Area under the curve is 0.8477
```

Out[105]:

```
0.847180682280796
```

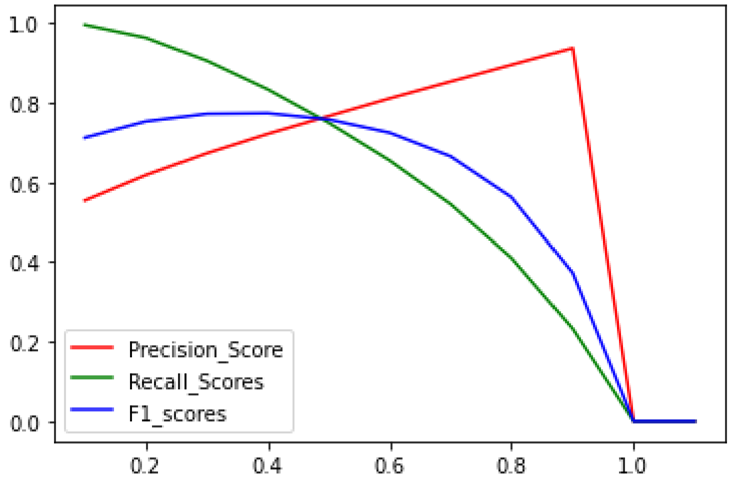
Tuning hyperparameter

In [106]:

```
p=[]
r=[]
f=[]
classifier = LogisticRegression(penalty='l1',solver='liblinear',C=100,class_weight='balance
classifier.fit(X_train, y_train)
y_prediction = classifier.predict_proba(X_train)[:,1]
l=np.arange(0.1,1.2,0.1)
for i in l:
    y_pred=[]
    for j in y_prediction:
        if j<=i:
            y_pred.append(0)
        else:
            y_pred.append(1)

    #print('Threshold is ',i)
    #print ("Precision Score : ", precision_score(y_train, y_pred))
    #print ("Recall Score : ", recall_score(y_train, y_pred))
    #print ("F1 Score : ",f1_score(y_train, y_pred))
    p.append(precision_score(y_train, y_pred))
    r.append(recall_score(y_train, y_pred))
    f.append(f1_score(y_train, y_pred))

plt.plot(l,p,'r')
plt.plot(l,r,'g')
plt.plot(l,f,'b')
plt.legend(['Precision_Score','Recall_Scores','F1_scores'])
# We can conclude that precision score increases while recall score decreases
# For us both recall and precision both are inportant
```

Out[106]:

```
<matplotlib.legend.Legend at 0x1d21104b970>
```

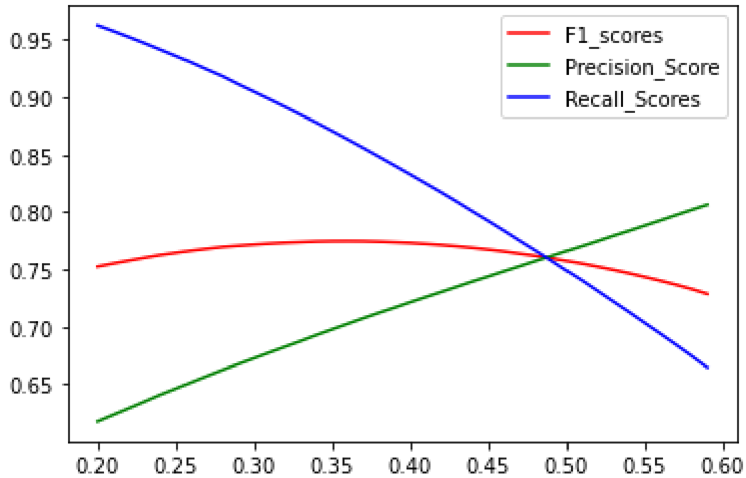


In [107]:

```
l=np.arange(0.2,0.6,0.01)
fscores=[]
pre=[]
rec=[]
for i in l:
    y_pred=[]
    for j in y_prediction:
        if j<=i:
            y_pred.append(0)
        else:
            y_pred.append(1)
    pre.append(precision_score(y_train, y_pred))
    rec.append(recall_score(y_train, y_pred))
    fscores.append(f1_score(y_train, y_pred))
plt.plot(l,fscores,'r')
plt.plot(l,pre,'g')
plt.plot(l,rec,'b')
plt.legend(['F1_scores','Precision_Score','Recall_Scores'])
# Max F1 score reached somewhere between 0.35 to 0.45
```

Out[107]:

```
<matplotlib.legend.Legend at 0x1d20a87c670>
```

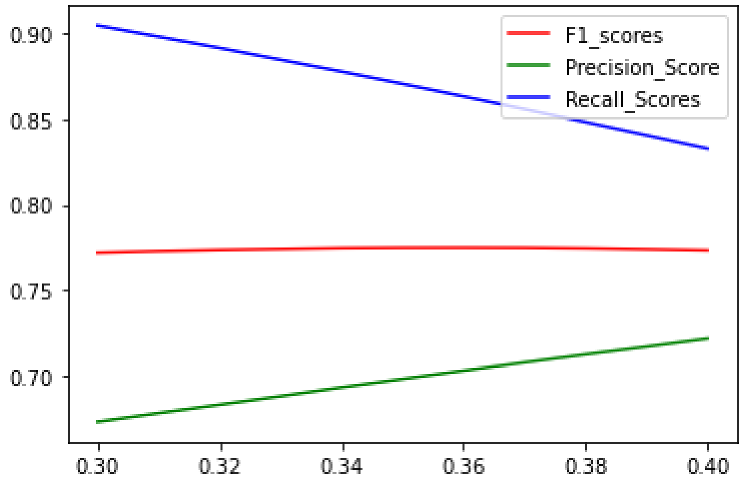


In [108]:

```
l=np.arange(0.3,0.4,0.01)
fscores=[]
pre=[]
rec=[]
for i in l:
    y_pred=[]
    for j in y_prediction:
        if j<=i:
            y_pred.append(0)
        else:
            y_pred.append(1)
    pre.append(precision_score(y_train, y_pred))
    rec.append(recall_score(y_train, y_pred))
    fscores.append(f1_score(y_train, y_pred))
plt.plot(l,fscores,'r')
plt.plot(l,pre,'g')
plt.plot(l,rec,'b')
plt.legend(['F1_scores','Precision_Score','Recall_Scores'])
```

Out[108]:

```
<matplotlib.legend.Legend at 0x1d2111b9250>
```

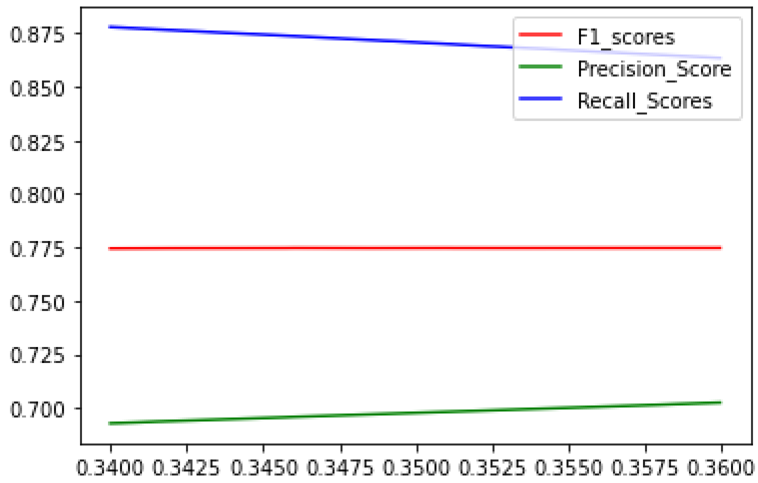


In [109]:

```
l=np.arange(0.34,0.36,0.00005)
fscores=[]
pre=[]
rec=[]
for i in l:
    y_pred=[]
    for j in y_prediction:
        if j<=i:
            y_pred.append(0)
        else:
            y_pred.append(1)
    pre.append(precision_score(y_train, y_pred))
    rec.append(recall_score(y_train, y_pred))
    fscores.append(f1_score(y_train, y_pred))
plt.plot(l,fscores,'r')
plt.plot(l,pre,'g')
plt.plot(l,rec,'b')
plt.legend(['F1_scores','Precision_Score','Recall_Scores'])
```

Out[109]:

```
<matplotlib.legend.Legend at 0x1d2107f0b80>
```

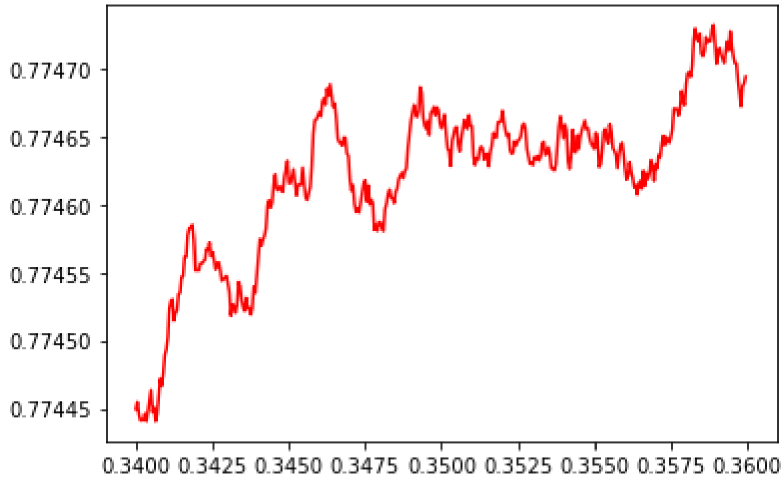


In [110]:

```
plt.plot(l,fscores,'r')
```

Out[110]:

```
[<matplotlib.lines.Line2D at 0x1d2113299d0>]
```



In [111]:

```
np.argmax(fscores)
```

Out[111]:

```
378
```

In [112]:

```
thr=l[np.argmax(fscores)]
print('The threshold for train data is ', thr)
```

```
The threshold for train data is  0.35889999999999794
```

In [113]:

```
classifier = LogisticRegression(penalty='l1',solver='liblinear',C=100,class_weight='balance
classifier.fit(X_train, y_train)
y_pred1 = classifier.predict_proba(X_train)[:,1]
y_predt=[]
for j in y_pred1:
    if j<=thr:
        y_predt.append(0)
    else:
        y_predt.append(1)
```

In [114]:

```
print ("Accuracy : ", accuracy_score(y_train, y_predt))
print ("Precision Score : ", precision_score(y_train, y_predt))
print ("Recall Score : ", recall_score(y_train, y_predt))
print ("F1 Score : ",f1_score(y_train, y_predt))
print ("G-mean Score : ",geometric_mean_score(y_train, y_predt))
print ("F0.5 Score : ",fbeta_score(y_train, y_predt,beta=0.5))
print ("F2 Score : ",fbeta_score(y_train, y_predt,beta=2))
# Metrics for Train data
```

```
Accuracy :  0.7487047950142027
Precision Score :  0.7020542283210713
Recall Score :  0.8641455038792555
F1 Score :  0.7747121805860152
G-mean Score :  0.7397515209947144
F0.5 Score :  0.7294182146635155
F2 Score :  0.8260037218108489
```

In [115]:

```
y_pred1 = classifier.predict_proba(X_test)[:,1]
y_predte=[]
for j in y_pred1:
    if j<=thr:
        y_predte.append(0)
    else:
        y_predte.append(1)
```

In [116]:

```
print ("Accuracy : ", accuracy_score(y_test, y_predte))
print ("Precision Score : ", precision_score(y_test, y_predte))
print ("Recall Score : ", recall_score(y_test, y_predte))
print ("F1 Score : ",f1_score(y_test, y_predte))
print ("G-mean Score : ",geometric_mean_score(y_test, y_predte))
print ("F0.5 Score : ",fbeta_score(y_test, y_predte,beta=0.5))
print ("F2 Score : ",fbeta_score(y_test, y_predte,beta=2))
# Metrics for Test data
```

```
Accuracy :  0.6388968687158862
Precision Score :  0.3186456536358586
Recall Score :  0.7788736420811893
F1 Score :  0.45226485174195424
G-mean Score :  0.6868858994671433
F0.5 Score :  0.36134910740337944
F2 Score :  0.6043096851467801
```

In [117]:

```
cm = confusion_matrix(y_train, y_predt)
print ("Confusion Matrix : \n", cm)
```

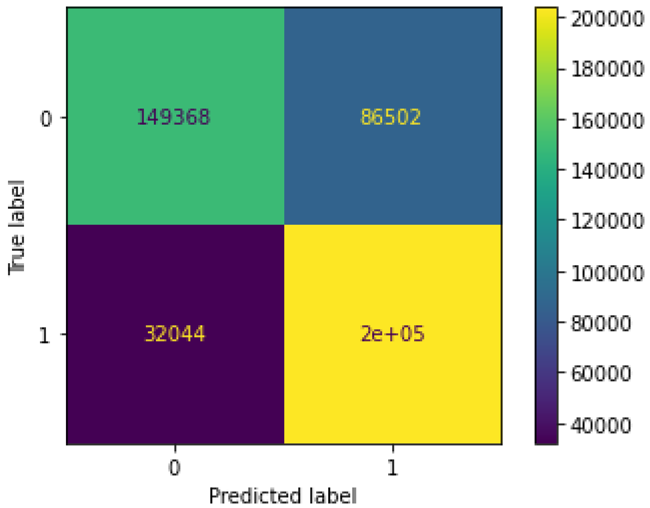
```
Confusion Matrix :
 [[149368  86502]
 [ 32044 203826]]
```

In [118]:

```
ConfusionMatrixDisplay(cm).plot()
# Confusion matrix for Train Data
```

Out[118]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1d20fbc8
ee0>
```



In [119]:

```
arr=classifier.coef_.reshape(-1)
```

In [120]:

```
col=df.columns
```

In [121]:

```
dic={}
for i in range(len(col)-1):
    dic[col[i]]=arr[i]
```

In [122]:

```
dict(sorted(dic.items(), key=lambda item: abs(item[1])))
```

Out[122]:

```
{'mort_acc': -0.0018622833243111085,
 'open_acc': 0.014881911427646369,
 'year': -0.017422705341610487,
 'grade': 0.018624810025700524,
 'total_acc': -0.0304574601184983,
 'initial_list_status': 0.0324330879011071,
 'application_type': -0.03717693831261523,
 'ear_month': -0.04122891723530323,
 'pub_rec_bankruptcies': -0.0483916944438291,
 'int_rate': -0.052692980252915536,
 'pub_rec': -0.061474971412049925,
 'verification_status': 0.06214626984004846,
 'emp_length': -0.06299493535800307,
 'annual_inc': -0.0650865530291524,
 'loan_status': -0.08227787297103605,
 'month': -0.095625908036303,
 'revol_util': -0.1170674246905151,
 'home_ownership': 0.12972289677834542,
 'revol_bal': 0.13755842114475852,
 'dti': 0.14963294495434115,
 'loan_amnt': -0.18299968166606184,
 'title': 0.1941999028009056,
 'installment': 0.2215059341702857,
 'term': 0.27831302612605774,
 'purpose': 0.47030665759889995,
 'sub_grade': 0.5028416569854783,
 'ear_year': 0.8928003224205188,
 'emp_title': 0.9712957020107084}
```

In [123]:

```
classifier.intercept_
# Intercept in logistic regression
```

Out[123]:

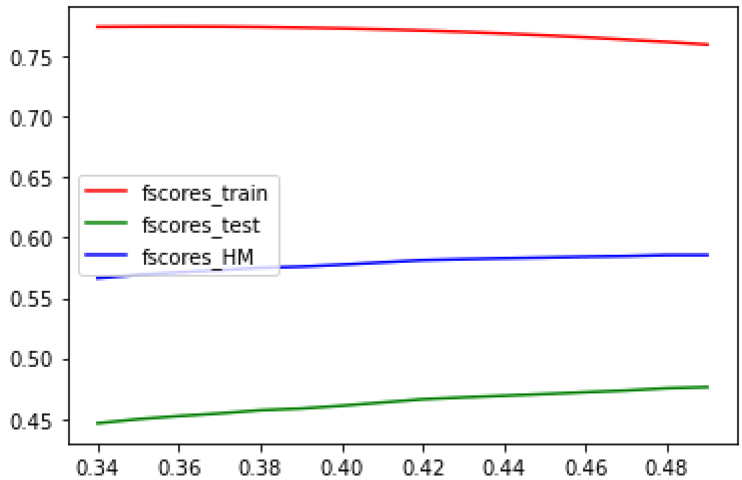
```
array([0.05915696])
```

In [124]:

```
y_pred1 = classifier.predict_proba(X_train)[:,1]
y_pred2 = classifier.predict_proba(X_test)[:,1]
l=np.arange(0.34,0.5,0.01)
fscores_train=[]
fscores_test=[]
fscores_HM=[]
for i in l:
    y_pred=[]
    for j in y_pred1:
        if j<=i:
            y_pred.append(0)
        else:
            y_pred.append(1)
    fscores_train.append(f1_score(y_train, y_pred))
    y_pred=[]
    for j in y_pred2:
        if j<=i:
            y_pred.append(0)
        else:
            y_pred.append(1)
    fscores_test.append(f1_score(y_test, y_pred))
    A=fscores_train[-1]
    B=fscores_test[-1]
    fscores_HM.append(2*A*B/(A+B))


plt.plot(l,fscores_train,'r')
plt.plot(l,fscores_test,'g')
plt.plot(l,fscores_HM,'b')

plt.legend(['fscores_train','fscores_test','fscores_HM'])
# Tradeoff between F1 scores for train and test
```

Out[124]:

```
<matplotlib.legend.Legend at 0x1d20f8bfc70>
```



In [125]:

```
np.argmax(fscores_test)
```

Out[125]:

```
15
```

In [126]:

```
np.argmax(fscores_train)
```

Out[126]:

```
2
```

In [127]:

```
np.argmax(fscores_HM)
```

Out[127]:

```
15
```

In [128]:

```
thre=l[np.argmax(fscores_HM)]
```

In [129]:

```
classifier = LogisticRegression(penalty='l1',solver='liblinear',C=100,class_weight='balance
classifier.fit(X_train, y_train)
y_pred1 = classifier.predict_proba(X_train)[:,1]
y_predt=[]
for j in y_pred1:
    if j<=thre:
        y_predt.append(0)
    else:
        y_predt.append(1)
print ("Accuracy : ", accuracy_score(y_test, y_predte))
print ("Precision Score : ", precision_score(y_test, y_predte))
print ("Recall Score : ", recall_score(y_test, y_predte))
print ("F1 Score : ",f1_score(y_test, y_predte))
print ("G-mean Score : ",geometric_mean_score(y_test, y_predte))
print ("F0.5 Score : ",fbeta_score(y_test, y_predte,beta=0.5))
print ("F2 Score : ",fbeta_score(y_test, y_predte,beta=2))
#Metrics for train data with optimum threshold
```

```
Accuracy :  0.6388968687158862
Precision Score :  0.3186456536358586
Recall Score :  0.7788736420811893
F1 Score :  0.45226485174195424
G-mean Score :  0.6868858994671433
F0.5 Score :  0.36134910740337944
F2 Score :  0.6043096851467801
```

In [130]:

```
cm = confusion_matrix(y_train, y_predt)
print ("Confusion Matrix : \n", cm)
```

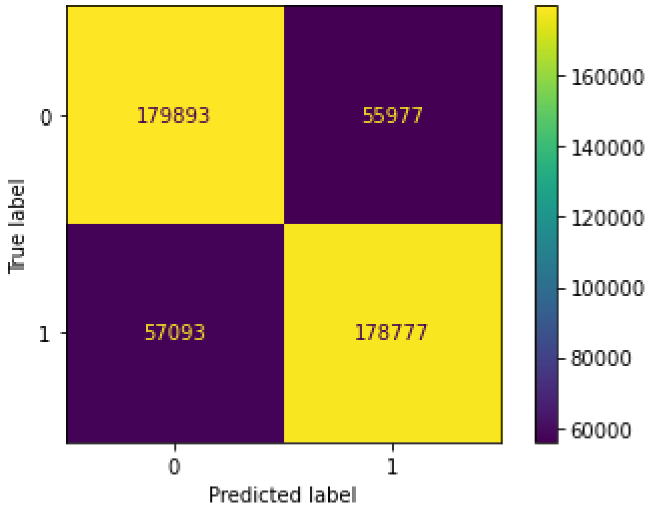
```
Confusion Matrix :
 [[179893  55977]
 [ 57093 178777]]
```

In [131]:

```
ConfusionMatrixDisplay(cm).plot()
```

Out[131]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1d2057c9
fd0>
```



In [132]:

```
y_pred1 = classifier.predict_proba(X_test)[:,1]
y_predte=[]
for j in y_pred1:
    if j<=thre:
        y_predte.append(0)
    else:
        y_predte.append(1)
#Metrics for train data with optimum threshold
```

In [133]:

```
print ("Accuracy : ", accuracy_score(y_test, y_predte))
print ("Precision Score : ", precision_score(y_test, y_predte))
print ("Recall Score : ", recall_score(y_test, y_predte))
print ("F1 Score : ",f1_score(y_test, y_predte))
print ("G-mean Score : ",geometric_mean_score(y_test, y_predte))
print ("F0.5 Score : ",fbeta_score(y_test, y_predte,beta=0.5))
print ("F2 Score : ",fbeta_score(y_test, y_predte,beta=2))
```

```
Accuracy :  0.7242308586749839
Precision Score :  0.3741479939594302
Recall Score :  0.6551600914808462
F1 Score :  0.47629439118800815
G-mean Score :  0.6965623374074288
F0.5 Score :  0.4092556877031322
F2 Score :  0.5695982303744299
```

In [ ]: