In [1]:

```
#!pip install category-encoders
```

In [2]:

```
import seaborn as sns
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import random
random.seed = 42
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.metrics import classification_report, plot_confusion_matrix, confusion_matrix,\
roc_auc_score, roc_curve, precision_recall_curve, fbeta_score, recall_score,\
precision_recall_fscore_support,accuracy_score, make_scorer, log_loss
from sklearn.calibration import CalibratedClassifierCV
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler,MinMaxScaler
```

Problem Statement:

Problem Statement:

Given a set of attributes for an Individual, determine if a credit line should be extended to them. If so, what should the repayment terms be in business recommendations?

- **If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company**
- **If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company**

Data dictionary :

Data dictionary :

- **loan_amnt : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.**
- **term : The number of payments on the loan. Values are in months and can be either 36 or 60.**
- **int_rate : Interest Rate on the loan**
- **installment : The monthly payment owed by the borrower if the loan originates.**
- **grade : LoanTap assigned loan grade**
- **sub_grade : LoanTap assigned loan subgrade**
- **emp_title :The job title supplied by the Borrower when applying for the loan.***
- **emp_length : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.**
- **home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.**
- **annual_inc : The self-reported annual income provided by the borrower during registration. verification_status : Indicates if income was verified by LoanTap, not verified, or if the income source was verified**
- **issue_d : The month which the loan was funded loan_status : Current status of the loan - Target Variable**
- **purpose : A category provided by the borrower for the loan request.**
- **title : The loan title provided by the borrower**
- **dti : A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower's self-reported monthly income.**
- **earliest_cr_line :The month the borrower's earliest reported credit line was opened**
- **open_acc : The number of open credit lines in the borrower's credit file.**
- **pub_rec : Number of derogatory public records**
- **revol_bal : Total credit revolving balance**
- **revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.**
- **total_acc : The total number of credit lines currently in the borrower's credit file**
- **initial_list_status : The initial listing status of the loan. Possible values are – W, F**
- **application_type : Indicates whether the loan is an individual application or a joint application with two co- borrowers**
- **mort_acc : Number of mortgage accounts. pub_rec_bankruptcies : Number of public record bankruptcies**
- **Address: Address of the individual**

In [3]:

```
loan_tap=pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921")
```

In [4]:

```
loan_tap.head().T
```

Out[4]:

	0	1	2	3	4
loan_amnt	10000.0	8000.0	15600.0	7200.0	24375.0
term	36 months	36 months	36 months	36 months	60 months
int_rate	11.44	11.99	10.49	6.49	17.27

installment	329.48	265.68	506.97	220.65	609.33
grade	B	B	B	A	C
sub_grade	B4	B5	B3	A2	C5
emp_title	Marketing	Credit analyst	Statistician	Client Advocate	Destiny Management Inc.
emp_length	10+ years	4 years	< 1 year	6 years	9 years
home_ownership	RENT	MORTGAGE	RENT	RENT	MORTGAGE
annual_inc	117000.0	65000.0	43057.0	54000.0	55000.0
verification_status	Not Verified	Not Verified	Source Verified	Not Verified	Verified
issue_d	Jan-2015	Jan-2015	Jan-2015	Nov-2014	Apr-2013
loan_status	Fully Paid	Fully Paid	Fully Paid	Fully Paid	Charged Off
purpose	vacation	debt_consolidation	credit_card	credit_card	credit_card
title	Vacation	Debt consolidation	Credit card refinancing	Credit card refinancing	Credit Card Refinance
dti	26.24	22.05	12.79	2.6	33.95
earliest_cr_line	Jun-1990	Jul-2004	Aug-2007	Sep-2006	Mar-1999
open_acc	16.0	17.0	13.0	6.0	13.0
pub_rec	0.0	0.0	0.0	0.0	0.0
revol_bal	36369.0	20131.0	11987.0	5472.0	24584.0
revol_util	41.8	53.3	92.2	21.5	69.8
total_acc	25.0	27.0	26.0	13.0	43.0
initial_list_status	w	f	f	f	f
application_type	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL
mort_acc	0.0	3.0	0.0	0.0	1.0
pub_rec_bankruptcies	0.0	0.0	0.0	0.0	0.0
address	0174 Michelle Gateway\r\nMendozaberg, OK 22690	1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113	87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113	823 Reid Ford\r\nDelacruzside, MA 00813	679 Luna Roads\r\nGreggshire, VA 11650

In [5]:

```
loan tap.describe()
```

Out[5]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000	396030.000000	3.960300e+05	395754.000000	396030.000000	358235.000000	395495.000000
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153	0.178191	1.584454e+04	53.791749	25.414744	1.813991	0.121648
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649	0.530671	2.059184e+04	24.452193	11.886991	2.147930	0.356174
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000	0.000000	0.000000e+00	0.000000	2.000000	0.000000	0.000000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000	0.000000	6.025000e+03	35.800000	17.000000	0.000000	0.000000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000	0.000000	1.118100e+04	54.800000	24.000000	1.000000	0.000000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000	0.000000	1.962000e+04	72.900000	32.000000	3.000000	0.000000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000	86.000000	1.743266e+06	892.300000	151.000000	34.000000	8.000000

In [6]:

```
loan tap.dtypes
```

Out[6]:

```
loan_amnt               float64
term                     object
int_rate                float64
installment             float64
grade                    object
sub_grade                object
emp_title                object
emp_length               object
home_ownership           object
annual_inc              float64
verification_status      object
issue_d                  object
loan_status              object
purpose                  object
title                    object
dti                     float64
earliest_cr_line         object
open_acc                float64
pub_rec                 float64
revol_bal               float64
revol_util              float64
total_acc               float64
initial_list_status      object
application_type         object
mort_acc                float64
pub_rec_bankruptcies    float64
address                  object
dtype: object
```

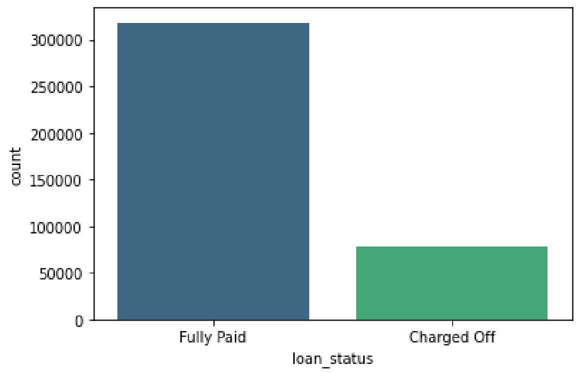
Loan Status Distribution

In [7]:

```
# lets check if loan_status now has only fully paid and charged off
sns.countplot(x=loan_tap['loan_status'], data=loan_tap, palette='viridis')
```

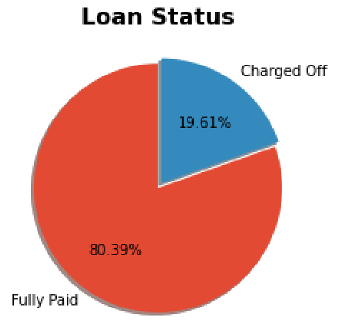
Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bfec07370>
```



In [8]:

```
plt.style.use('ggplot')
data = loan_tap["loan_status"].value_counts(normalize=True)
plt.pie(data, labels=data.index, startangle = 90, shadow = True, radius=1, explode= [0,0.05],autopct='%0.2f%%')
plt.title("Loan Status", fontsize=16, fontweight='bold')
plt.show()
```



In [9]:

```
#plt.figure(figsize=(6,3),dpi=120)
#loan_tap.corr()['loan_status'].sort_values().drop('loan_status').plot(kind='bar', cmap='viridis') # correlation with loan_status for continuous features with loan_status feature dropped
#plt.xticks(rotation=90);
```

In [10]:

```
loan_tap.columns
```

Out[10]:

```
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')
```

EDA

loan_amnt & installment

In [11]:

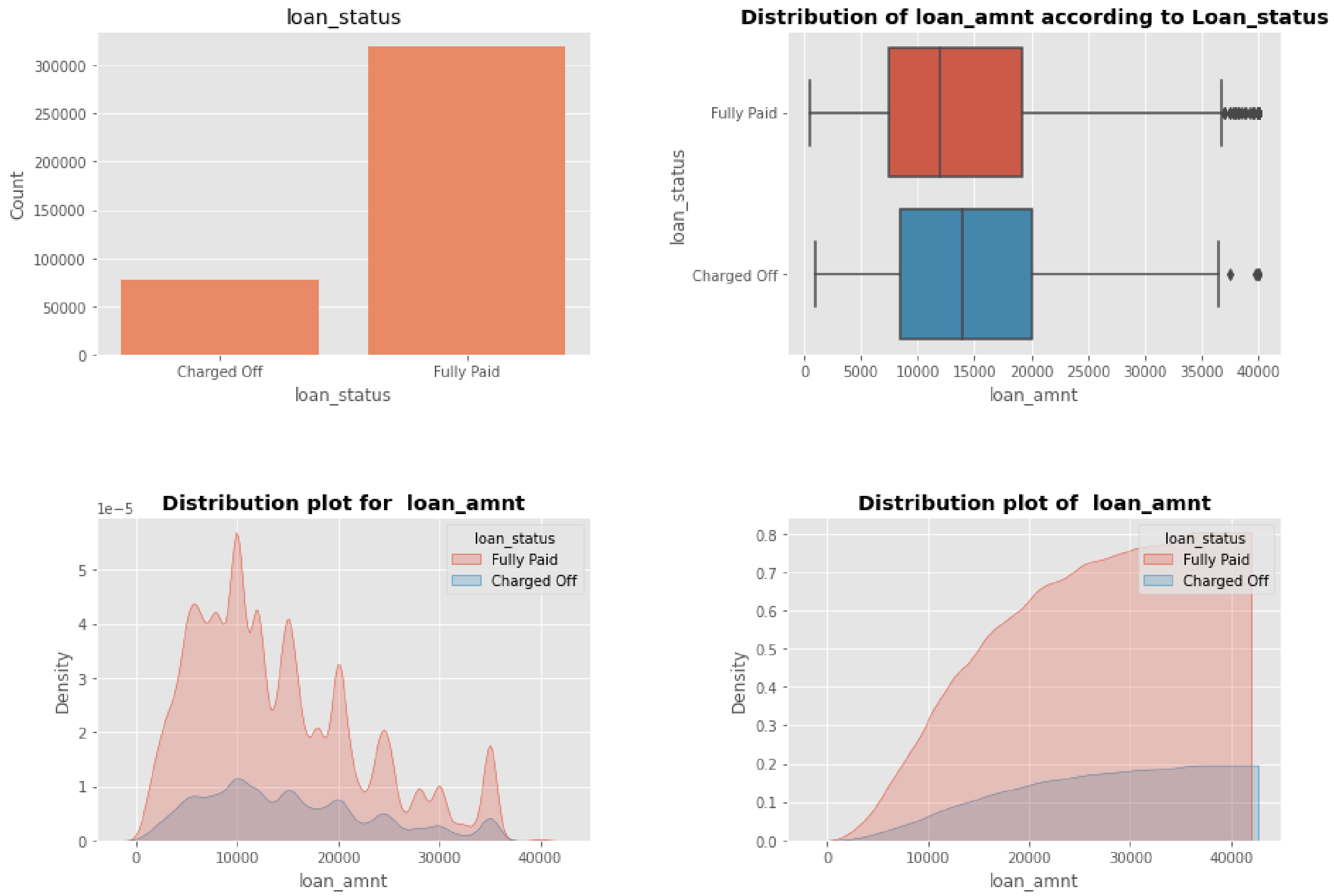
```
#sns.pairplot(loan_tap,hue='loan_status',y_vars=['loan_amnt'])
#sns.pairplot(loan_tap,hue='loan_status',y_vars=['installment'])
#plt.show()
```

In [12]:

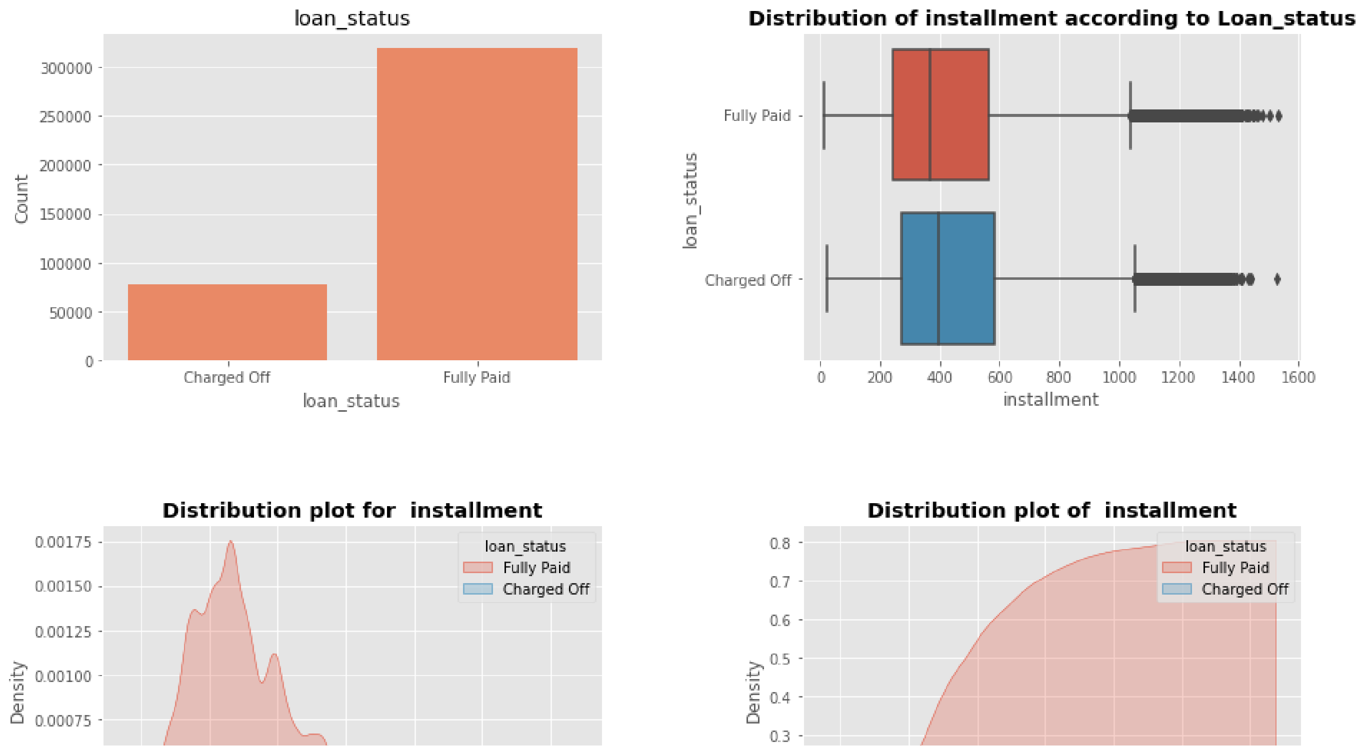
```
reccol=np.unique(loan_tap['loan_status'], return_counts=True)
for col in ['loan_amnt','installment','annual_inc','int_rate']:
  fig = plt.figure(figsize=(15,10))
  #plt.subplots_adjust(wspace = 0.5,hspace=1)
  fig.subplots_adjust(top=0.92);
  fig.subplots_adjust(hspace=0.5, wspace=0.4);
  ax1 = plt.subplot(221)
  sns.barplot(x=list(reccol[0]), y=list(reccol[1]), ax=ax1, color='coral');
  ax1.set_title('loan_status');
  ax1.set_xlabel('loan_status');
```

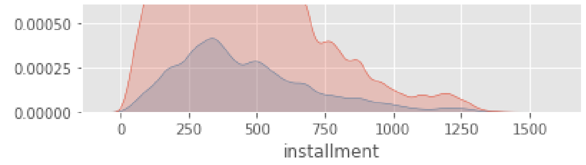
```
ax1.set_ylabel('Count');


ax2 = plt.subplot(222)
sns.boxplot(data=loan_tap,x=col,y='loan_status',orient='h')
plt.title(f"Distribution of {col} according to {'Loan_status'}",fontweight='bold')

ax1 = plt.subplot(223)
sns.kdeplot(loan_tap[col], shade=True, ax=ax1,hue=loan_tap['loan_status'])
plt.xlabel(col)
plt.title(f"Distribution plot for  {col}",fontweight='bold')

ax2 = plt.subplot(224)
sns.kdeplot(loan_tap[col], shade=True, cumulative=True,ax=ax2,hue=loan_tap['loan_status'])
plt.xlabel(col)
plt.title(f'Distribution plot of  {col}',fontweight='bold')


print('-'*50+col+'-'*50)
plt.show()
```

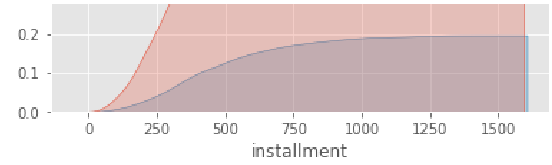
```
--------------------------------------------------loan_amnt--------------------------------------------------
```



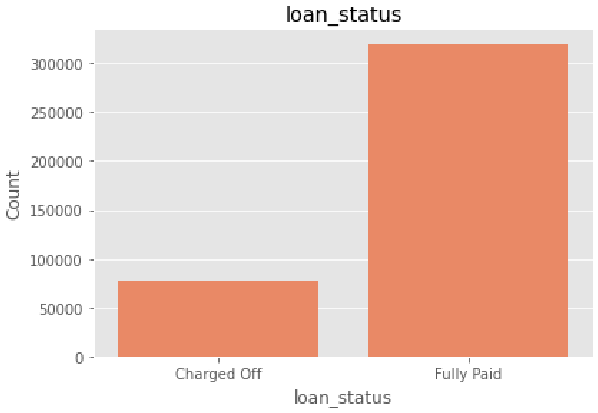

```
--------------------------------------------------installment--------------------------------------------------
```

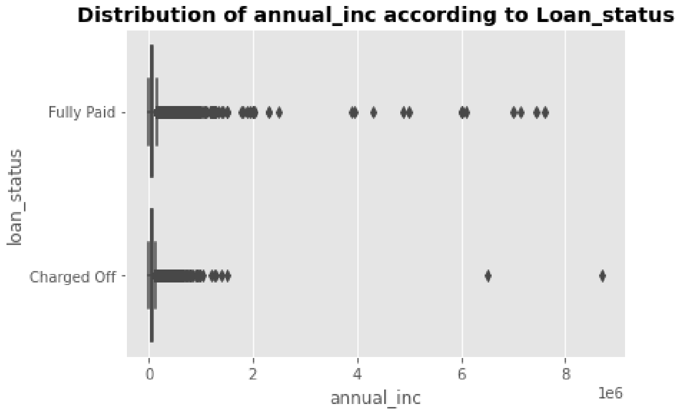


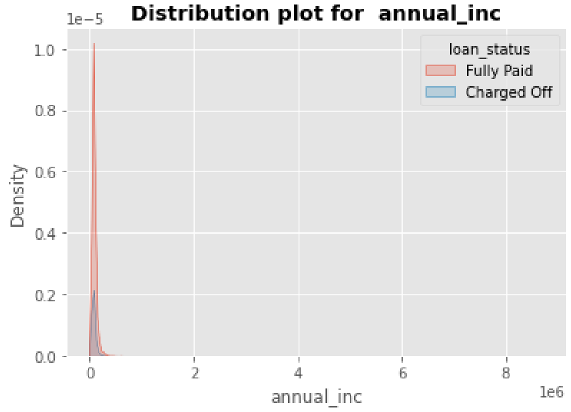


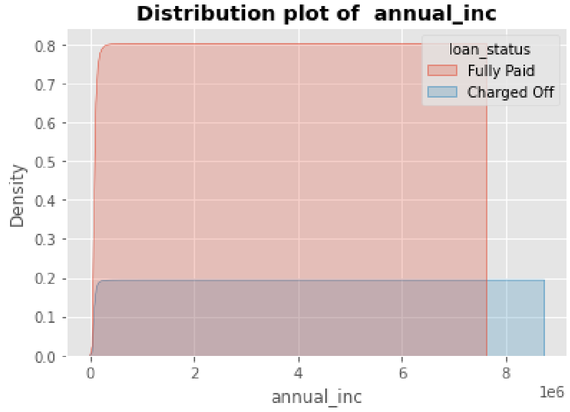


--annual_inc--

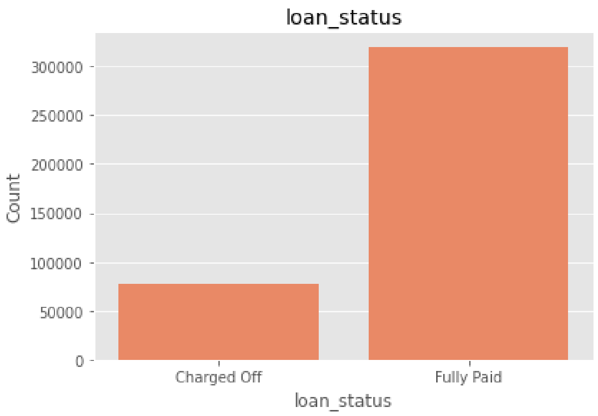


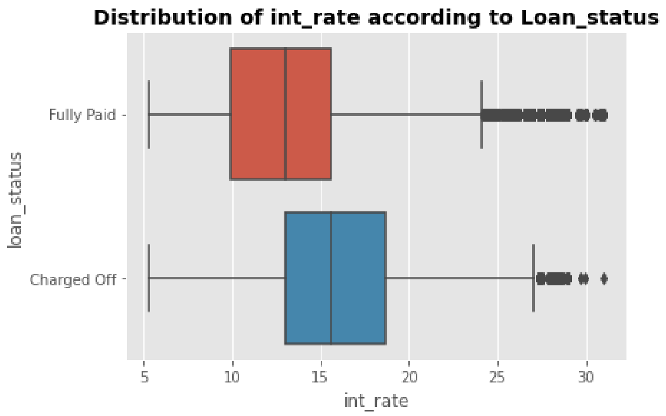


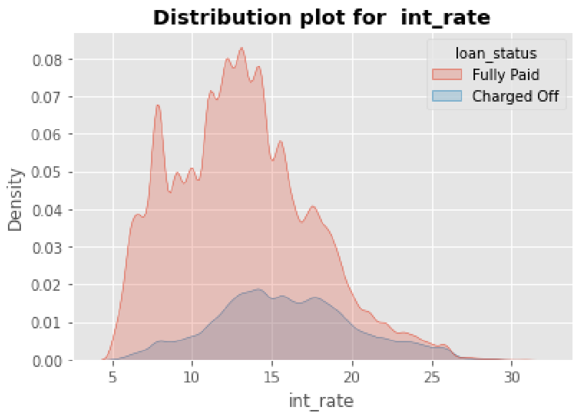


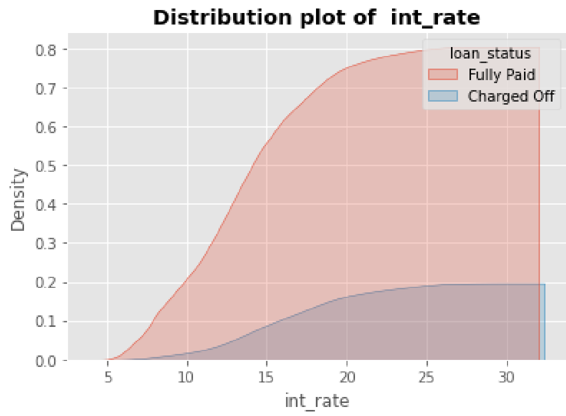


--int_rate--









Outlier treatment

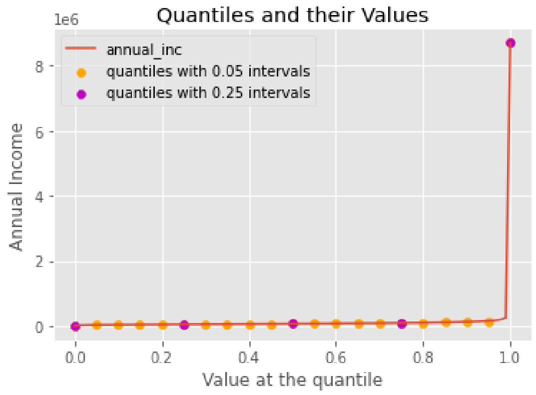
annual_inc

In [13]:

```
quantiles = loan_tap['annual_inc'].quantile(np.arange(0,1.01,0.01), interpolation='higher')

plt.title("Quantiles and their Values")
quantiles.plot()
# quantiles with 0.05 difference
plt.scatter(x=quantiles.index[::5], y=quantiles.values[::5], c='orange', label="quantiles with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=quantiles.index[::25], y=quantiles.values[::25], c='m', label = "quantiles with 0.25 intervals")
plt.ylabel('Annual Income')
plt.xlabel('Value at the quantile')
plt.legend(loc='best')
```

Out[13]:

```
<matplotlib.legend.Legend at 0x7f6bf930cb50>
```



In [14]:

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(loan_tap['annual_inc'],90+i))
print('-'*100)
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(loan_tap['annual_inc'],99+(i/100)))
```

```
90 percentile value is 120000.0
91 percentile value is 125000.0
92 percentile value is 130000.0
93 percentile value is 136000.0
94 percentile value is 144000.0
95 percentile value is 150000.0
96 percentile value is 160000.0
97 percentile value is 175000.0
98 percentile value is 200000.0
99 percentile value is 250000.0
100 percentile value is 8706582.0
----------------------------------------------------------------------------------------------------
99.1 percentile value is 250000.0
99.2 percentile value is 260000.0
99.3 percentile value is 275000.0
99.4 percentile value is 290000.0
99.5 percentile value is 300000.0
99.6 percentile value is 325000.0
99.7 percentile value is 350000.0
99.8 percentile value is 400000.0
99.9 percentile value is 515000.0
100.0 percentile value is 8706582.0
```

In [15]:

```
loan_tap[loan_tap['annual_inc']>300000].loan_status.value_counts()
```

Out[15]:

```
Fully Paid     1581
Charged Off     237
Name: loan_status, dtype: int64
```

In [16]:

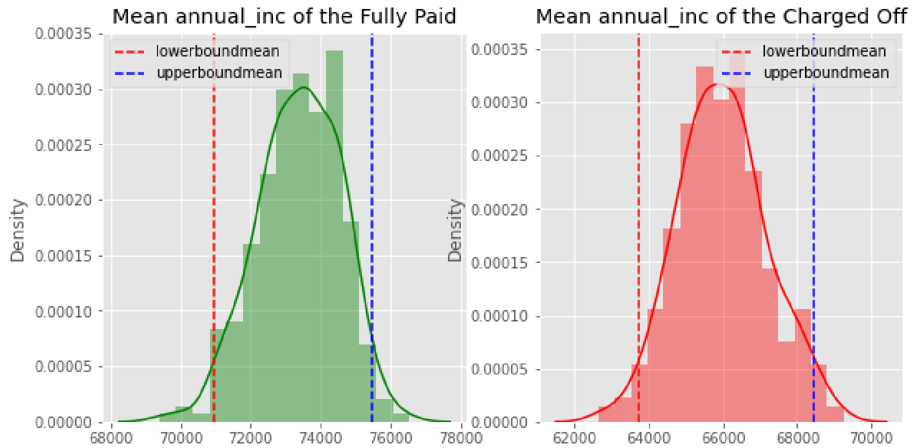
```
loan_tap=loan_tap[loan_tap['annual_inc']<300000]
```

In [17]:

```
bs_Fully_Paid_means=[]
bs_Charged_Off_means=[]
for i in range(1,301):
    bs_Fully_Paid_means.append(np.random.choice(loan_tap[loan_tap.loan_status=="Fully Paid"]["annual_inc"],size=1000).mean())
    bs_Charged_Off_means.append(np.random.choice(loan_tap[loan_tap.loan_status=='Charged Off']["annual_inc"],size=1000).mean())
print(f"mean annual_inc of Fully Paid 95 CI   :   {np.percentile(bs_Fully_Paid_means,[2.5,97.5])}")
print(f"mean annual_inc of Charged Off 95 CI        :   {np.percentile(bs_Charged_Off_means,[2.5,97.5])}")


fig, axs = plt.subplots(1, 2,figsize=(10, 5))
sns.distplot(bs_Fully_Paid_means,color="g",ax=axs[0])
```

```
axs[0].axvline(np.percentile(bs_Fully_Paid_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[0].axvline(np.percentile(bs_Fully_Paid_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[0].legend()
axs[0].set_title("Mean annual_inc of the Fully Paid")
sns.distplot(bs_Charged_Off_means,color="r",ax=axs[1])
axs[1].axvline(np.percentile(bs_Charged_Off_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[1].axvline(np.percentile(bs_Charged_Off_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[1].legend()
axs[1].set_title("Mean annual_inc of the Charged Off")
plt.show()
```

```
mean annual_inc of Fully Paid 95 CI   :   [70972.17733325 75461.47790675]
mean annual_inc of Charged Off 95 CI        :   [63752.99707725 68459.7582295 ]
```



- **It seems that loans with large annual income are more likely to be pay on time Full payment.**
- **Lets remove outliers in the annual_inc some of the users having extreamly large anual income ,lets conside maximum income 300000.0 we can cover 99.5% of data**
- **mean annual_inc of Fully Paid 95 CI :**

```
[71097.35145525 , 75717.72822375]
```

- **mean annual_inc of Charged Off 95 CI :**

```
[63865.159879 ,  68177.4551325]
```

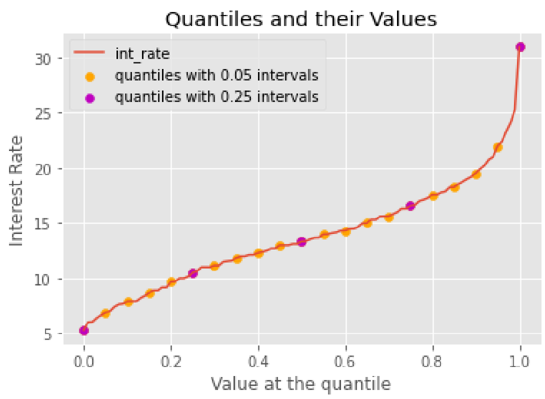
int_rate

In [18]:

```
quantiles = loan_tap['int_rate'].quantile(np.arange(0,1.01,0.01), interpolation='higher')

plt.title("Quantiles and their Values")
quantiles.plot()
# quantiles with 0.05 difference
plt.scatter(x=quantiles.index[::5], y=quantiles.values[::5], c='orange', label="quantiles with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=quantiles.index[::25], y=quantiles.values[::25], c='m', label = "quantiles with 0.25 intervals")
plt.ylabel('Interest Rate')
plt.xlabel('Value at the quantile')
plt.legend(loc='best')
```

Out[18]:

```
<matplotlib.legend.Legend at 0x7f6bf9ce72e0>
```



In [19]:

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(loan_tap['int_rate'],90+i))
print('-'*100)
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(loan tap['int rate'],99+(i/100)))
```

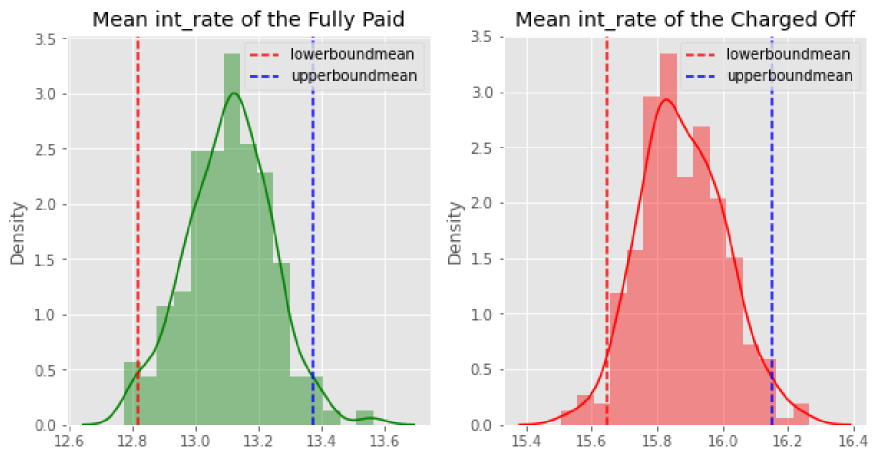
```
90 percentile value is 19.52
91 percentile value is 19.91
```

```
92 percentile value is 20.2
93 percentile value is 20.75
94 percentile value is 21.0
95 percentile value is 21.97
96 percentile value is 22.4
97 percentile value is 23.28
98 percentile value is 24.08
99 percentile value is 25.28
100 percentile value is 30.99
----------------------------------------------------------------------------------------------------
99.1 percentile value is 25.57
99.2 percentile value is 25.57
99.3 percentile value is 25.8
99.4 percentile value is 25.8
99.5 percentile value is 25.83
99.6 percentile value is 25.89
99.7 percentile value is 26.06
99.8 percentile value is 26.99
99.9 percentile value is 27.99
100.0 percentile value is 30.99
```

In [20]:

```
bs_Fully_Paid_means=[]
bs_Charged_Off_means=[]
for i in range(1,301):
    bs_Fully_Paid_means.append(np.random.choice(loan_tap[loan_tap.loan_status=="Fully Paid"]["int_rate"],size=1000).mean())
    bs_Charged_Off_means.append(np.random.choice(loan_tap[loan_tap.loan_status=='Charged Off']["int_rate"],size=1000).mean())
print(f"mean int_rate of Fully Paid 95 CI   :   {np.percentile(bs_Fully_Paid_means,[2.5,97.5])}")
print(f"mean int_rate of Charged Off 95 CI        :   {np.percentile(bs_Charged_Off_means,[2.5,97.5])}")


fig, axs = plt.subplots(1, 2,figsize=(10, 5))
sns.distplot(bs_Fully_Paid_means,color="g",ax=axs[0])
axs[0].axvline(np.percentile(bs_Fully_Paid_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[0].axvline(np.percentile(bs_Fully_Paid_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[0].legend()
axs[0].set_title("Mean int_rate of the Fully Paid")
sns.distplot(bs_Charged_Off_means,color="r",ax=axs[1])
axs[1].axvline(np.percentile(bs_Charged_Off_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[1].axvline(np.percentile(bs_Charged_Off_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[1].legend()
axs[1].set_title("Mean int_rate of the Charged Off")
plt.show()
```

```
mean int_rate of Fully Paid 95 CI   :   [12.8188455  13.37229775]
mean int_rate of Charged Off 95 CI        :   [15.64517675 16.15186525]
```



- **It seems that loans with high intersest rate are more likely to be unpaid.**
- **mean int_rate of Fully Paid 95 CI :**

```
[12.816506  13.4235715]
```

- **mean int_rate of Charged Off 95 CI :**

```
[15.64039175 16.14115875]
```

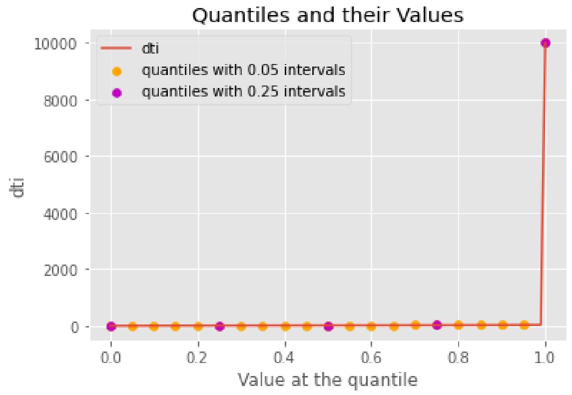
dti

In [21]:

```
quantiles = loan_tap['dti'].quantile(np.arange(0,1.01,0.01), interpolation='higher')

plt.title("Quantiles and their Values")
quantiles.plot()
# quantiles with 0.05 difference
plt.scatter(x=quantiles.index[::5], y=quantiles.values[::5], c='orange', label="quantiles with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=quantiles.index[::25], y=quantiles.values[::25], c='m', label = "quantiles with 0.25 intervals")
plt.ylabel('dti')
plt.xlabel('Value at the quantile')
plt.legend(loc='best')
```

Out[21]:

```
<matplotlib.legend.Legend at 0x7f6bfe62c5e0>
```



In [22]:

```python
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(loan_tap['dti'],90+i))
print('-'*100)
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(loan_tap['dti'],99+(i/100)))
```

```
90 percentile value is 28.52
91 percentile value is 29.02
92 percentile value is 29.55
93 percentile value is 30.13
94 percentile value is 30.84
95 percentile value is 31.6
96 percentile value is 32.43
97 percentile value is 33.36
98 percentile value is 34.39
99 percentile value is 36.46
100 percentile value is 9999.0
----------------------------------------------------------------------------------------------------
99.1 percentile value is 36.78
99.2 percentile value is 37.1
99.3 percentile value is 37.43
99.4 percentile value is 37.81
99.5 percentile value is 38.16
99.6 percentile value is 38.54
99.7 percentile value is 38.95
99.8 percentile value is 39.42
99.9 percentile value is 39.86
100.0 percentile value is 9999.0
```

In [23]:

```python
loan_tap=loan_tap[loan_tap['dti']<50]
```

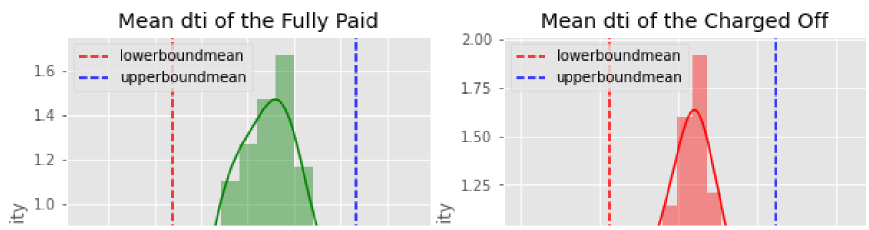
In [23]:

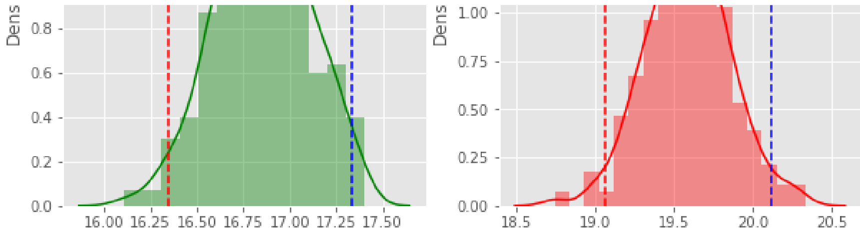
```python

```

In [24]:

```python
bs_Fully_Paid_means=[]
bs_Charged_Off_means=[]
for i in range(1,301):
    bs_Fully_Paid_means.append(np.random.choice(loan_tap[loan_tap.loan_status=="Fully Paid"]["dti"],size=1000).mean())
    bs_Charged_Off_means.append(np.random.choice(loan_tap[loan_tap.loan_status=='Charged Off']["dti"],size=1000).mean())
print(f"mean dti of Fully Paid 95 CI   :   {np.percentile(bs_Fully_Paid_means,[2.5,97.5])}")
print(f"mean dti of Charged Off 95 CI        :   {np.percentile(bs_Charged_Off_means,[2.5,97.5])}")


fig, axs = plt.subplots(1, 2,figsize=(10, 5))
sns.distplot(bs_Fully_Paid_means,color="g",ax=axs[0])
axs[0].axvline(np.percentile(bs_Fully_Paid_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[0].axvline(np.percentile(bs_Fully_Paid_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[0].legend()
axs[0].set_title("Mean dti of the Fully Paid")
sns.distplot(bs_Charged_Off_means,color="r",ax=axs[1])
axs[1].axvline(np.percentile(bs_Charged_Off_means,2.5),linestyle="--", color='r', label="lowerboundmean")
axs[1].axvline(np.percentile(bs_Charged_Off_means,97.5),linestyle="--", color='b', label="upperboundmean")
axs[1].legend()
axs[1].set_title("Mean dti of the Charged Off")
plt.show()
```

```
mean dti of Fully Paid 95 CI   :   [16.3426515 17.331998 ]
mean dti of Charged Off 95 CI        :   [19.069045   20.11527125]
```





- **Lets remove outliers in the dti some of the users having extreamly dti ,lets conside maximum dti 50 we can cover 99.9% of data**
- **mean dti of Fully Paid 95 CI :**

```
[16.419986   17.33308275]
```

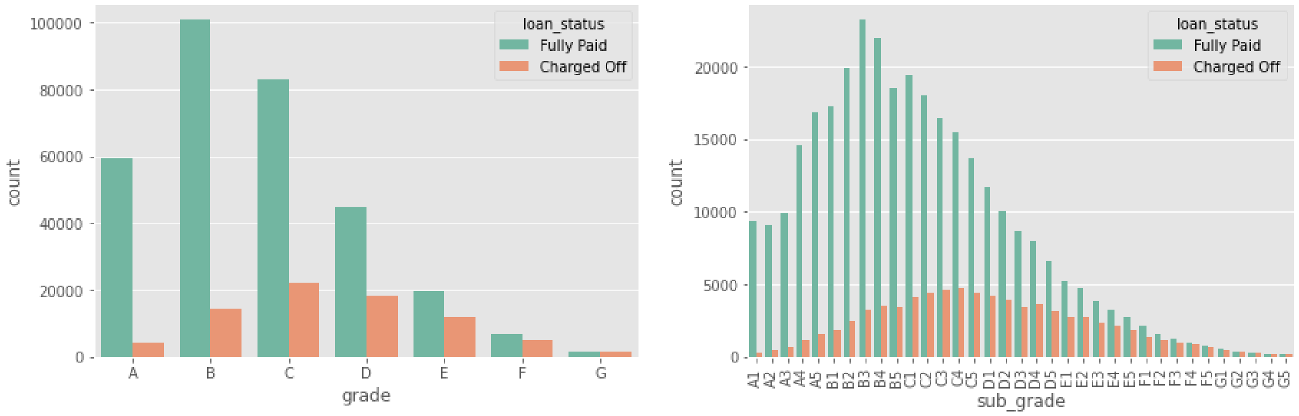
- **mean dti of Charged Off 95 CI :**

```
[19.00360025 19.99442475]
```

grade and sub_grade

In [25]:

```
plt.figure(figsize=(15, 10))
# List of color palette to use
rgb_values = sns.color_palette("Set2", 6)

plt.subplot(2, 2, 1)
grade = sorted(loan_tap.grade.unique().tolist())
sns.countplot(x='grade', data=loan_tap, hue='loan_status', order=grade,palette=rgb_values)

plt.subplot(2, 2, 2)
sub_grade = sorted(loan_tap.sub_grade.unique().tolist())
g = sns.countplot(x='sub_grade', data=loan_tap, hue='loan_status', order=sub_grade,palette=rgb_values)
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```



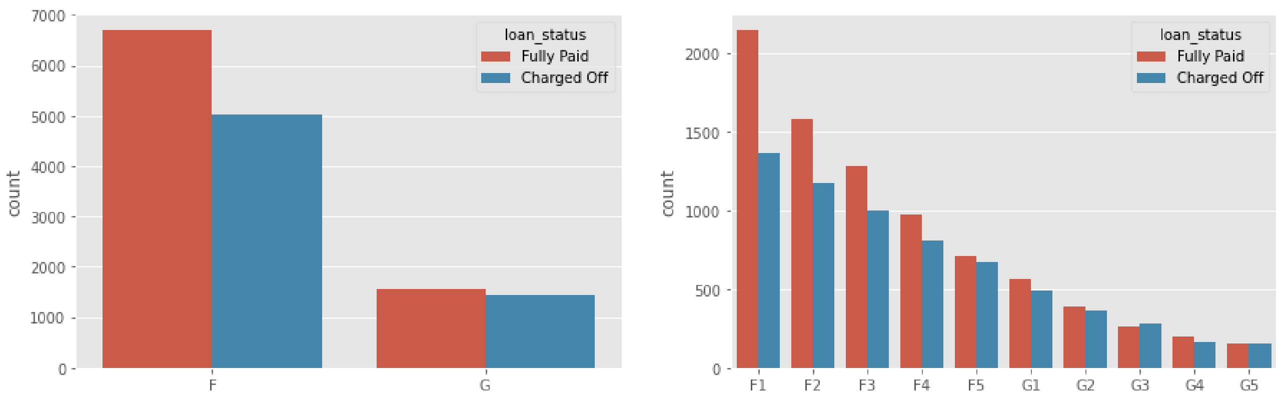
It looks like F and G subgrades don't get paid back that often. Isloate those and recreate the countplot just for those subgrades.

In [26]:

```
df = loan_tap[(loan_tap.grade == 'F') | (loan_tap.grade == 'G')]

plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
grade = sorted(df.grade.unique().tolist())
sns.countplot(x='grade', data=df, hue='loan_status', order=grade)

plt.subplot(2, 2, 2)
sub_grade = sorted(df.sub_grade.unique().tolist())
sns.countplot(x='sub_grade', data=df, hue='loan_status', order=sub_grade)
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bf9c82460>
```



grade sub_grade

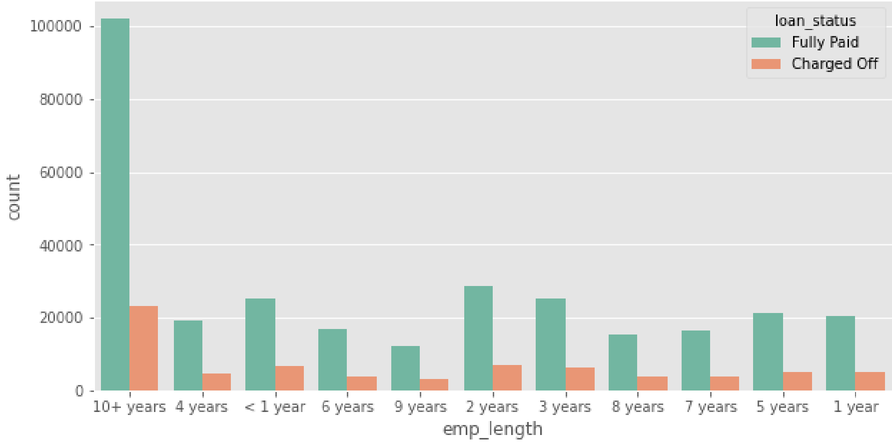
emp_length

In [27]:

```
plt.figure(figsize=(10, 5))
sns.countplot(data=loan_tap,x="emp_length",color="blue",hue="loan_status",palette=rgb_values)
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bfe5a9e50>
```



In [28]:

```
loan_tap.columns
```

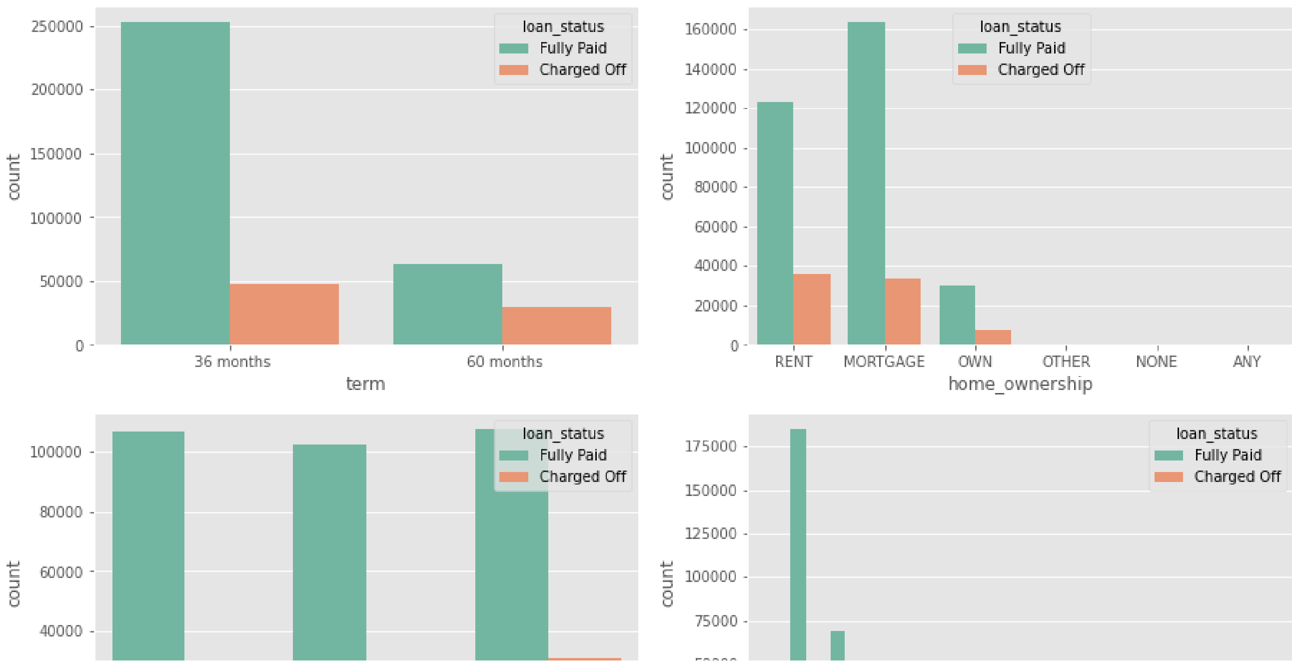
Out[28]:

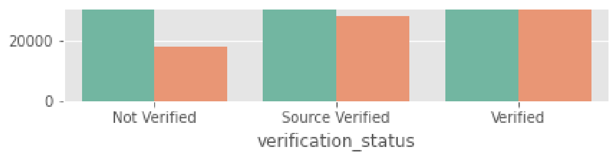
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util',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

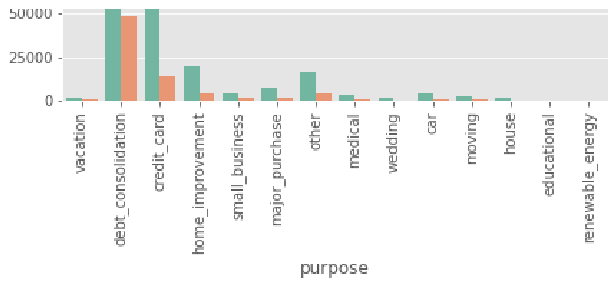
term ,home_ownership , verification_status and purpose

In [29]:

```
plt.figure(figsize=(15, 20))

plt.subplot(4, 2, 1)
sns.countplot(x='term', data=loan_tap, hue='loan_status',palette=rgb_values)

plt.subplot(4, 2, 2)
sns.countplot(x='home_ownership', data=loan_tap, hue='loan_status',palette=rgb_values)

plt.subplot(4, 2, 3)
sns.countplot(x='verification_status', data=loan_tap, hue='loan_status',palette=rgb_values)

plt.subplot(4, 2, 4)
g = sns.countplot(x='purpose', data=loan_tap, hue='loan_status',palette=rgb_values)
g.set_xticklabels(g.get_xticklabels(), rotation=90);
```







- **36 months term is most popular among customers**
- **People with home_ownership Mortgage are more likely to fully pay the loan**

In [29]:

```

```

Missing values

In [30]:

```
def missingValue(df):
    #Identifying Missing data.
    total_null = df.isnull().sum().sort_values(ascending = False)
    percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascending = False)
    print("Total records = ", df.shape[0])

    md = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In Percent'])
    return md
missing_df = missingValue(loan_tap)
missing_df[missing_df['Total Missing'] > 0]
```

```
Total records =  393714
```

Out[30]:

	Total Missing	In Percent
mort_acc	37586	9.55
emp_title	22775	5.78
emp_length	18267	4.64
title	1748	0.44
pub_rec_bankruptcies	532	0.14
revol_util	269	0.07

In [31]:

```
missing_df.head()
```

Out[31]:

	Total Missing	In Percent
mort_acc	37586	9.55
emp_title	22775	5.78
emp_length	18267	4.64
title	1748	0.44
pub_rec_bankruptcies	532	0.14

In [31]:

```

```

CLEANING, PREPROCESSING, FEATURE ENGINEERING

emp_length

In [32]:

```
loan_tap["emp_length"].unique()
```

Out[32]:

```
array(['10+ years', '4 years', '< 1 year', '6 years', '9 years',
       '2 years', '3 years', '8 years', '7 years', '5 years', '1 year',
       nan], dtype=object)
```

In [33]:

```
# converting emp_length to numerical column and assign nan values to zero

def emp_length_convert(df, column):
```

```
    df[column] = df[column].str.replace('\+ years', '')
    df[column] = df[column].str.replace('< 1 year', str(0))
    df[column] = df[column].str.replace(' years', '')
    df[column] = df[column].str.replace(' year', '')
    df[column] = pd.to_numeric(df[column])
    #df[column].fillna(value = 0, inplace = True)
    return df
loan tap=emp length convert(loan tap, 'emp length')
```

In [34]:

```
loan tap['emp length'].unique()
```

Out[34]:

```
array([10.,  4.,  0.,  6.,  9.,  2.,  3.,  8.,  7.,  5.,  1., nan])
```

team

In [35]:

```
loan tap["term"].unique()
```

Out[35]:

```
array([' 36 months', ' 60 months'], dtype=object)
```

In [36]:

```
# converting term column to numeric data type

def term_numeric(df, column):
    df[column] = pd.to_numeric(df[column].str.replace(' months', ''))
    return df

loan tap=term numeric(loan tap, 'term')
```

In [37]:

```
loan tap["term"].unique()
```

Out[37]:

```
array([36, 60])
```

date columns

In [38]:

```
loan tap.head(2).T
```

Out[38]:

	0	1
loan_amnt	10000.0	8000.0
term	36	36
int_rate	11.44	11.99
installment	329.48	265.68
grade	B	B
sub_grade	B4	B5
emp_title	Marketing	Credit analyst
emp_length	10.0	4.0
home_ownership	RENT	MORTGAGE
annual_inc	117000.0	65000.0
verification_status	Not Verified	Not Verified
issue_d	Jan-2015	Jan-2015
loan_status	Fully Paid	Fully Paid
purpose	vacation	debt_consolidation
title	Vacation	Debt consolidation
dti	26.24	22.05
earliest_cr_line	Jun-1990	Jul-2004
open_acc	16.0	17.0
pub_rec	0.0	0.0
revol_bal	36369.0	20131.0
revol_util	41.8	53.3
total_acc	25.0	27.0
initial_list_status	w	f
application_type	INDIVIDUAL	INDIVIDUAL
mort_acc	0.0	3.0
pub_rec_bankruptcies	0.0	0.0

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

In [39]:

```
# issue_d , earliest_cr_line
#earliest_cr_line :The month the borrower's earliest reported credit line was opened
#issue_d : The month which the loan was funded

print(f"""issue_d          :  min --> {loan_tap["issue_d"].min()} , max --> {loan_tap["issue_d"].max()}""")
print(f"""earliest_cr_line :  min --> {loan_tap["earliest_cr_line"].min()} , max --> {loan_tap["earliest_cr_line"].max()}""")
```

```
issue_d          :  min --> Apr-2008 , max --> Sep-2016
earliest_cr_line :  min --> Apr-1955 , max --> Sep-2013
```

In [40]:

```
loan_tap['earliest_cr_line_date'] = pd.to_datetime(loan_tap['earliest_cr_line'], format='%b-%Y')
loan_tap['mths_since_earliest_cr_line'] = round(pd.to_numeric((pd.to_datetime('2023-01-01') - loan_tap['earliest_cr_line_date']) / np.timedelta64(1, 'M')))
```

In [41]:

```
loan_tap['issue_d_date'] = pd.to_datetime(loan_tap['issue_d'], format='%b-%Y')
loan_tap['mths_since_issue_d'] = round(pd.to_numeric((pd.to_datetime('2023-01-01') - loan_tap['issue_d_date']) / np.timedelta64(1, 'M')))
```

In [42]:

```
loan_tap[loan_tap['mths_since_issue_d']<0].shape,loan_tap[loan_tap['mths_since_earliest_cr_line']<0].shape
```

Out[42]:

```
((0, 31), (0, 31))
```

In [42]:

In [42]:

In [43]:

```
# issue_d , earliest_cr_line
#earliest_cr_line :The month the borrower's earliest reported credit line was opened
#issue_d : The month which the loan was funded

print(f"""issue_d          :  min --> {loan_tap["issue_d"].min(),loan_tap["mths_since_issue_d"].max()} , max --> {loan_tap["issue_d"].max(),loan_tap["mths_since_issue_d"].min()}""")
print(f"""earliest_cr_line :  min --> {loan_tap["earliest_cr_line"].min(),loan_tap["mths_since_earliest_cr_line"].max()} , max --> {loan_tap["earliest_cr_line"].max(),loan_tap["mths_since_earliest_cr_line"].min()}""")
```

```
issue_d          :  min --> ('Apr-2008', 187.0) , max --> ('Sep-2016', 73.0)
earliest_cr_line :  min --> ('Apr-1955', 948.0) , max --> ('Sep-2013', 111.0)
```

Drop columns

We know that grade is just a sub feature of sub_grade, So we are goinig to drop it.

In [44]:

```
loan_tap['zip_code'] = loan_tap['address'].apply(lambda address:address[-5:])
loan_tap.drop(['earliest_cr_line', 'issue_d','issue_d_date','grade','earliest_cr_line_date','address',"title","emp_title"], axis=1, inplace=True)
```

In [44]:

Missing value treatment

In [45]:

```
loan_tap["loan_status"].unique()
```

Out[45]:

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

In [46]:

```
loan_tap["application_type"].unique()
```

Out[46]:

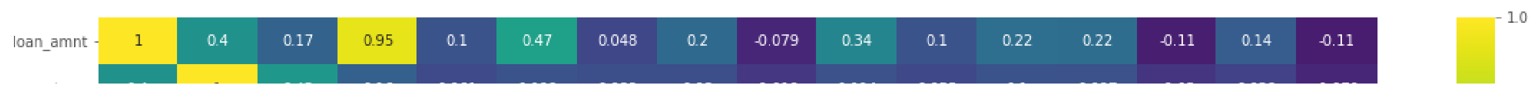
```
array(['INDIVIDUAL', 'JOINT', 'DIRECT_PAY'], dtype=object)
```

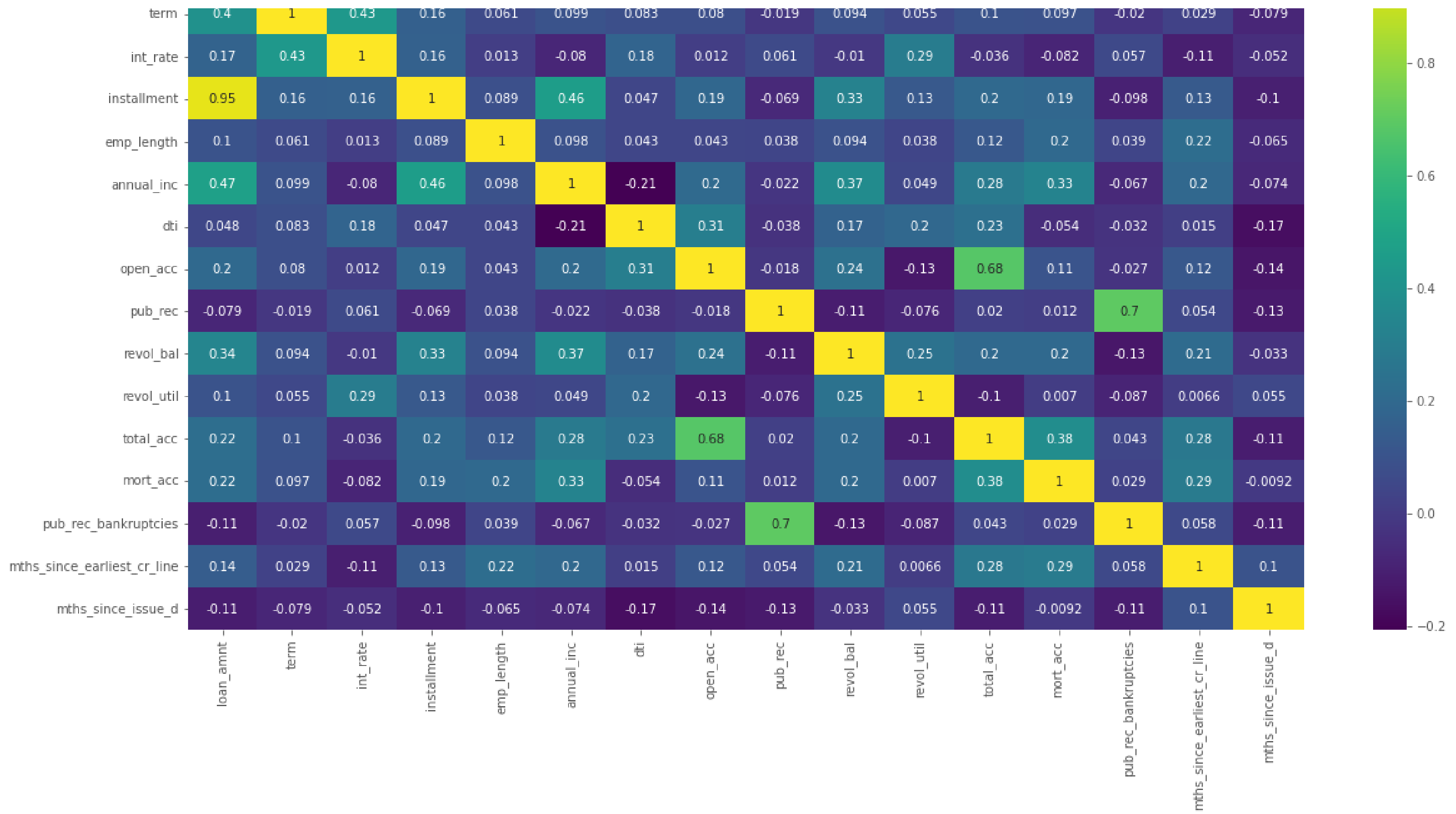
In [47]:

```
plt.figure(figsize=(20, 10))
sns.heatmap(loan_tap.corr(), annot=True, cmap='viridis')
```

Out[47]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bf8e97430>
```





- Loan amount and Installement amount are highly corelated.
- Open_acc and total accounts are highly correlated
- public records and public record banckruptcies are highly correlated.
- Annual income and the loan amount are modelrately correlated.
- Loan_status and term are modelrately corelated
- Loan_status and Intrest rate are modelrately corelated
- Loan_amont and (total account,mortgage accounts and revolution balance are moderately corelated).
- Total accounts and mortagage accounts are corelated.
- Annual income and total accounts are reasonably correlated.
- dti and open_acc are reasonbly corelated.
- Installments and mort_acc are slightly corelated.

Looks like the total_acc feature correlates with the mort_acc , this makes sense! Let's try this fillna() approach. We will group the dataframe by the total_acc and calculate the mean value for the mort_acc per total_acc entry. To get the result below:

In [48]:

```
total_acc_avg = loan_tap.groupby(by='total_acc').mean().mort_acc

def fill_mort_acc(total_acc, mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc].round()
    else:
        return mort_acc
loan_tap['emp_length'].fillna(0, inplace=True)
loan_tap['mort_acc'] = loan_tap.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']), axis=1)
```

In [49]:

```
loan_tap['mort_acc'].unique()
```

Out[49]:

```
array([ 0.,  3.,  1.,  4.,  2.,  6.,  5., 10.,  7., 12., 11.,  8.,  9.,
       13., 14., 22., 34., 15., 25., 19., 16., 17., 32., 18., 24., 21.,
       20., 31., 28., 30., 23., 26., 27.])
```

In [50]:

```
loan_tap["loan_status"].unique()
```

Out[50]:

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

In [51]:

```
def map_func(x):
  if x>1:
    return 1
  else:
```

```
    return 0
loan_tap["pub_rec"]=loan_tap["pub_rec"].apply(map_func)
loan_tap['mort_acc']=loan_tap['mort_acc'].apply(map_func)
loan_tap["pub_rec_bankruptcies"]=loan_tap["pub_rec_bankruptcies"].apply(map_func)
loan_tap['home_ownership'] = loan_tap['home_ownership'].replace(['NONE', 'ANY'], 'OTHER')
```

In [52]:

```
loan_tap.revol_util.unique()
```

Out[52]:

```
array([ 41.8 ,  53.3 ,  92.2 , ...,  56.26, 111.4 , 128.1 ])
```

In [53]:

```
missing_df = missingValue(loan_tap)
missing_df[missing_df['Total Missing'] > 0]
```

```
Total records =  393714
```

Out[53]:

	Total Missing	In Percent
revol_util	269	0.07

In [54]:

```
loan_tap.dropna(inplace=True)
```

Duplicated data checks

In [55]:

```
print(f"Data shape: {loan_tap.shape}")

# # Remove duplicate Features
# data = data.T.drop_duplicates()
# data = data.T

# # Remove Duplicate Rows
data.drop_duplicates(inplace=True)

print(f"Data shape: {loan_tap.shape}")
```

```
Data shape: (393445, 24)
Data shape: (393445, 24)
```

In [55]:

```

```

Train test split

In [56]:

```
loan_tap.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 393445 entries, 0 to 396029
Data columns (total 24 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   loan_amnt                     393445 non-null  float64
 1   term                          393445 non-null  int64  
 2   int_rate                      393445 non-null  float64
 3   installment                   393445 non-null  float64
 4   sub_grade                     393445 non-null  object 
 5   emp_length                    393445 non-null  float64
 6   home_ownership                393445 non-null  object 
 7   annual_inc                    393445 non-null  float64
 8   verification_status           393445 non-null  object 
 9   loan_status                   393445 non-null  object 
 10  purpose                       393445 non-null  object 
 11  dti                           393445 non-null  float64
 12  open_acc                      393445 non-null  float64
 13  pub_rec                       393445 non-null  int64  
 14  revol_bal                     393445 non-null  float64
 15  revol_util                    393445 non-null  float64
 16  total_acc                     393445 non-null  float64
 17  initial_list_status           393445 non-null  object 
 18  application_type              393445 non-null  object 
 19  mort_acc                      393445 non-null  int64  
 20  pub_rec_bankruptcies          393445 non-null  int64  
 21  mths_since_earliest_cr_line   393445 non-null  float64
 22  mths_since_issue_d            393445 non-null  float64
 23  zip_code                      393445 non-null  object 
dtypes: float64(12), int64(4), object(8)
memory usage: 75.0+ MB
```

In [57]:

```
loan_tap.home_ownership.unique()
```

Out[57]:

```
array(['RENT', 'MORTGAGE', 'OWN', 'OTHER'], dtype=object)
```

In [58]:

```
loan_tap["loan_status"]=loan_tap["loan_status"].map({"Fully Paid":0,"Charged Off":1}).astype("int64")
```

In [60]:

```
categorical_cols = [col for col in loan_tap.select_dtypes(include='object').columns.tolist()]
```

In [61]:

```
categorical_cols
```

Out[61]:

```
['sub_grade',
 'home_ownership',
 'verification_status',
 'purpose',
 'initial_list_status',
 'application_type',
 'zip_code']
```

In [63]:

```
loan_tap.loan_status.unique()
```

Out[63]:

```
array([0, 1])
```

In [71]:

```
#!pip install category_encoders
```

In [72]:

```
#x=loan_tap[col_Considered].drop(['loan_status'],axis=1)
x=loan_tap.drop(['loan_status'],axis=1)
y=loan_tap['loan_status']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42)
```

In [72]:

```

```

In [73]:

```
loan_tap.head(2).T
```

Out[73]:

	0	1
loan_amnt	10000.0	8000.0
term	36	36
int_rate	11.44	11.99
installment	329.48	265.68
sub_grade	B4	B5
emp_length	10.0	4.0
home_ownership	RENT	MORTGAGE
annual_inc	117000.0	65000.0
verification_status	Not Verified	Not Verified
loan_status	0	0
purpose	vacation	debt_consolidation
dti	26.24	22.05
open_acc	16.0	17.0
pub_rec	0	0
revol_bal	36369.0	20131.0
revol_util	41.8	53.3
total_acc	25.0	27.0
initial_list_status	w	f
application_type	INDIVIDUAL	INDIVIDUAL
mort_acc	0	1
pub_rec_bankruptcies	0	0
mths_since_earliest_cr_line	391.0	222.0
mths_since_issue_d	96.0	96.0
zip_code	22690	05113

In [73]:

```

```

Encoding

One Hot

In [74]:

```
"""loan_tap_dummy = pd.get_dummies(loan_tap, columns=categorical_cols, drop_first=True)

x=loan_tap_dummy.drop(['loan_status'],axis=1)
y=loan_tap_dummy['loan_status']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42)"""
print()
```

Target

In [75]:

```
from category_encoders import TargetEncoder
for i in categorical_cols:
  te = TargetEncoder()
  X_train[i] = te.fit_transform(X_train[i], y_train)
  X_test[i] = te.transform(X_test[i])
print()
```

In [76]:

```
X_train.head().T
```

Out[76]:

	371457	218763	242337	9057	284169
loan_amnt	14000.000000	20000.000000	28200.000000	6850.000000	15000.000000
term	36.000000	36.000000	36.000000	36.000000	36.000000
int_rate	6.620000	8.490000	11.990000	19.990000	11.990000
installment	429.860000	631.260000	936.510000	254.540000	498.150000
sub_grade	0.049363	0.099386	0.123718	0.345552	0.123718
emp_length	3.000000	10.000000	10.000000	8.000000	5.000000
home_ownership	0.226642	0.170783	0.226642	0.226642	0.170783
annual_inc	188000.000000	120000.000000	72000.000000	70000.000000	70000.000000
verification_status	0.223250	0.146691	0.223250	0.223250	0.146691
purpose	0.207688	0.172632	0.207688	0.207688	0.207688
dti	9.600000	9.740000	26.220000	34.730000	9.770000
open_acc	18.000000	17.000000	11.000000	8.000000	7.000000
pub_rec	0.000000	0.000000	0.000000	0.000000	0.000000
revol_bal	16591.000000	10384.000000	21628.000000	3846.000000	11113.000000
revol_util	17.400000	16.600000	79.500000	91.600000	62.100000
total_acc	29.000000	26.000000	33.000000	26.000000	27.000000
initial_list_status	0.193549	0.201379	0.193549	0.201379	0.201379
application_type	0.196632	0.196632	0.196632	0.196632	0.196632
mort_acc	1.000000	1.000000	1.000000	1.000000	0.000000
pub_rec_bankruptcies	0.000000	0.000000	0.000000	0.000000	0.000000
mths_since_earliest_cr_line	461.000000	322.000000	304.000000	292.000000	281.000000
mths_since_issue_d	114.000000	84.000000	108.000000	105.000000	109.000000
zip_code	0.201205	0.195643	0.000000	0.000000	0.000000

In [76]:

In [76]:

Standardization

In [76]:

In [77]:

```
scaler = MinMaxScaler()
X_train[X_train.columns] = scaler.fit_transform(X_train)
```

```
X_test[X_train.columns]=scaler.transform(X_test)
```

In [78]:

```
X_train_scaled,X_test_scaled=X_train,X_test
```

Multi Multicollinearity checks

In [79]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def multicollinearity_assumption(features,feature_names=None):
    """
    Multicollinearity: Assumes that predictors are not correlated with each other. If there is
                       correlation among the predictors, then either remove prepdictors with high
                       Variance Inflation Factor (VIF) values or perform dimensionality reduction

                       This assumption being violated causes issues with interpretability of the
                       coefficients and the standard errors of the coefficients.
    """
    from statsmodels.stats.outliers_influence import variance_inflation_factor
    print('Assumption 3: Little to no multicollinearity among predictors')

    # Plotting the heatmap
    plt.figure(figsize=(20, 10))
    sns.heatmap(pd.DataFrame(features, columns=feature_names).corr(), annot=True, cmap='viridis')
    plt.title('Correlation of Variables')
    plt.show()

    print('Variance Inflation Factors (VIF)')
    print('> 10: An indication that multicollinearity may be present')
    print('> 100: Certain multicollinearity among the variables')
    print('-------------------------------------')

    # Gathering the VIF for each variable

    VIF = [variance_inflation_factor(features.values, i) for i in range(features.shape[1])]
    for idx, vif in enumerate(VIF):
        print('{0}: {1}'.format(feature_names[idx], vif))

    # Gathering and printing total cases of possible or definite multicollinearity
    possible_multicollinearity = sum([1 for vif in VIF if vif > 10])
    definite_multicollinearity = sum([1 for vif in VIF if vif > 100])
    print()
    print('{0} cases of possible multicollinearity'.format(possible_multicollinearity))
    print('{0} cases of definite multicollinearity'.format(definite_multicollinearity))
    print()

    if definite_multicollinearity == 0:
        if possible_multicollinearity == 0:
            print('Assumption satisfied')
        else:
            print('Assumption possibly satisfied')
            print()
            print('Coefficient interpretability may be problematic')
            print('Consider removing variables with a high Variance Inflation Factor (VIF)')

    else:
        print('Assumption not satisfied')
        print()
        print('Coefficient interpretability will be problematic')
        print('Consider removing variables with a high Variance Inflation Factor (VIF)')
```

In [80]:

```
# compute the vif for all given features
def compute_vif(considered_features,df):

    X = df[considered_features]
    # the calculation of variance inflation requires a constant
    X['intercept'] = 1

    # create dataframe to store vif values
    vif = pd.DataFrame()
    vif["Variable"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif = vif[vif['Variable']!='intercept']
    return vif
```

In [81]:

```
# features to consider removing
considered_features = list(X_train_scaled.columns)
# compute vif
compute_vif(considered_features,X_train_scaled).sort_values('VIF', ascending=False)
```

Out[81]:

	Variable	VIF
0	loan_amnt	58.998831
3	installment	51.048280
2	int_rate	18.889098
4	sub_grade	18.760378

	Variable	VIF
1	term	6.714805
15	total_acc	2.332265
11	open_acc	2.139446
7	annual_inc	1.731579
18	mort_acc	1.567015
10	dti	1.438711
12	pub_rec	1.416551
13	revol_bal	1.410198
19	pub_rec_bankruptcies	1.401954
21	mths_since_issue_d	1.387032
6	home_ownership	1.353913
14	revol_util	1.324383
16	initial_list_status	1.275775
20	mths_since_earliest_cr_line	1.211998
8	verification_status	1.163421
5	emp_length	1.073222
9	purpose	1.043114
22	zip_code	1.039582
17	application_type	1.005638

In [82]:

```
# features to consider removing
# compute vif values after removing a feature
considered_features.remove('loan_amnt')
# compute vif
compute_vif(considered_features,X_train_scaled).sort_values('VIF', ascending=False)
```

Out[82]:

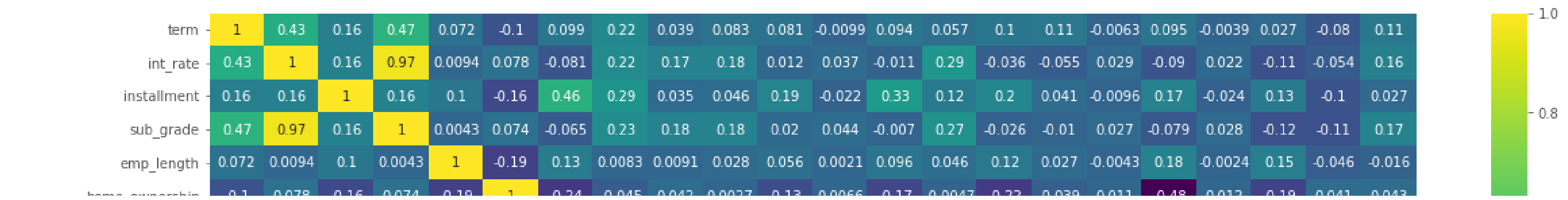
	Variable	VIF
3	sub_grade	18.752445
1	int_rate	18.320957
14	total_acc	2.332087
10	open_acc	2.139264
6	annual_inc	1.724699
17	mort_acc	1.566618
2	installment	1.470939
9	dti	1.438572
11	pub_rec	1.416480
0	term	1.410509
12	revol_bal	1.406429
18	pub_rec_bankruptcies	1.401954
20	mths_since_issue_d	1.386820
5	home_ownership	1.353905
13	revol_util	1.324266
15	initial_list_status	1.275561
19	mths_since_earliest_cr_line	1.211779
7	verification_status	1.163348
4	emp_length	1.073124
8	purpose	1.042978
21	zip_code	1.039484
16	application_type	1.005604

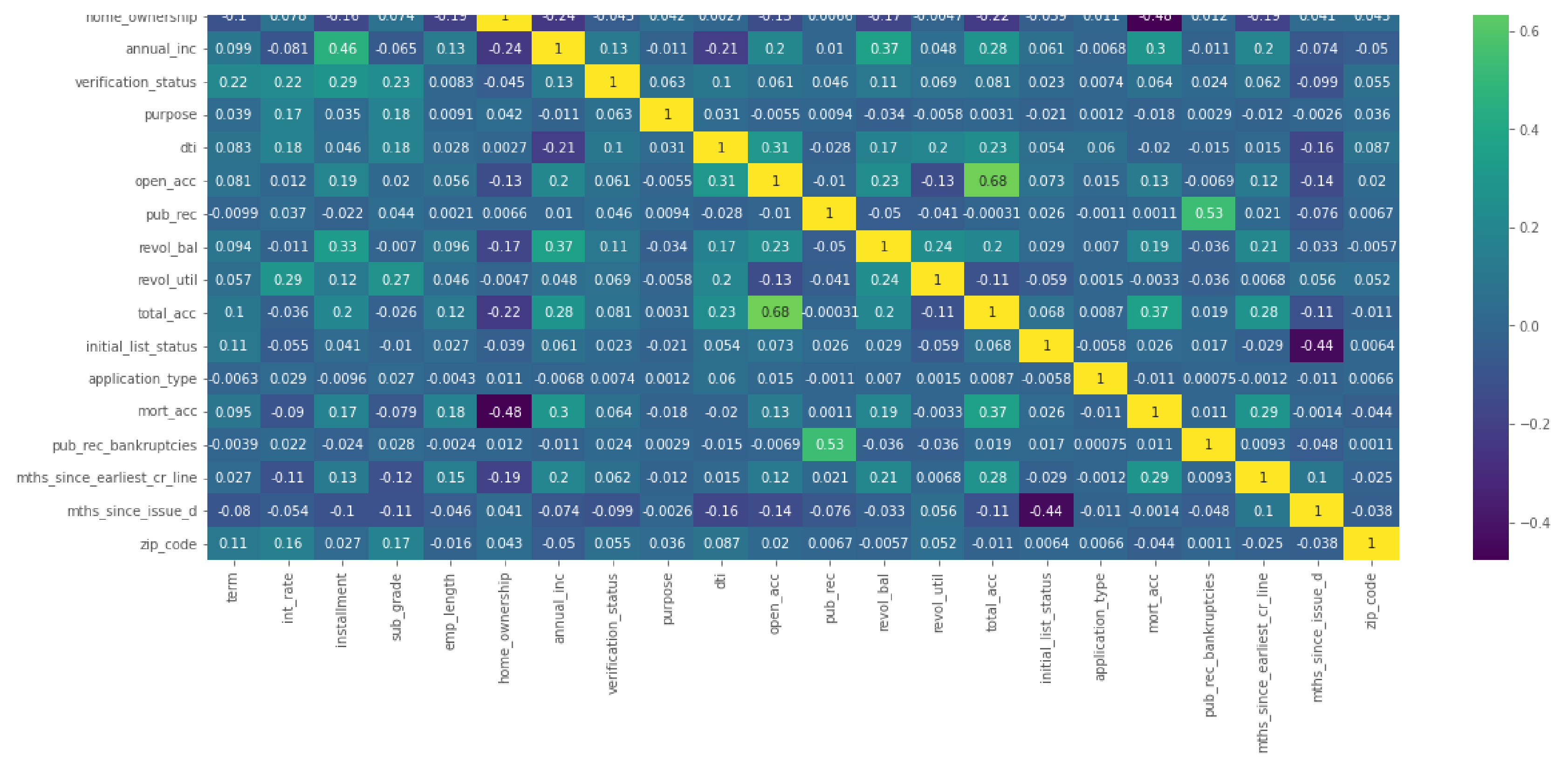
In [83]:

```
plt.figure(figsize=(20, 10))
sns.heatmap(X_train_scaled[considered_features].corr(), annot=True, cmap='viridis')
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bf3bca9d0>
```





In [84]:

```
# features to consider removing
# compute vif values after removing a feature
considered_features.remove('sub_grade')
# compute vif
compute_vif(considered_features,X_train_scaled).sort_values('VIF', ascending=False)
```

Out[84]:

	Variable	VIF
13	total_acc	2.331955
9	open_acc	2.138892
5	annual_inc	1.723358
1	int_rate	1.605647
16	mort_acc	1.565207
2	installment	1.470462
8	dti	1.438292
10	pub_rec	1.416106
11	revol_bal	1.406313
17	pub_rec_bankruptcies	1.401888
0	term	1.357604
4	home_ownership	1.352514
19	mths_since_issue_d	1.341105
12	revol_util	1.320837
14	initial_list_status	1.272006
18	mths_since_earliest_cr_line	1.210803
6	verification_status	1.163345
3	emp_length	1.071384
7	purpose	1.039535
20	zip_code	1.036955
15	application_type	1.005587

In [84]:

```
```

In [85]:

```
X_train_scaled,X_test_scaled=X_train_scaled[considered_features],X_test_scaled[considered_features]
```

Utills

In [86]:

```
#categoricals = list(X.select_dtypes('object').columns)
```

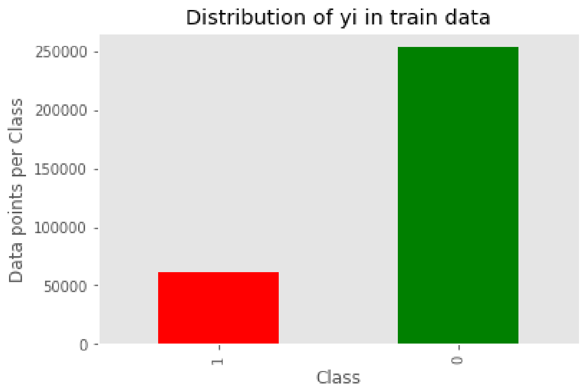
```
#categoricals = list(X.select_dtypes('object').columns)
#numericals = list(X.select_dtypes('int64').columns)
def encode_cats(categoricals, numericals,X):
    """
    Takes in a list of categorical columns and a list of numerical columns and returns the dataframe with encoded variables
    """
    ohe = OneHotEncoder(sparse=False, drop='first')
    cat_matrix = ohe.fit_transform(X.loc[:, categoricals])
    X_ohe = pd.DataFrame(cat_matrix,
                         columns=ohe.get_feature_names(categoricals), #create meaningful column names
                         index=X.index) #keep the same index values

    return pd.concat([X.loc[:, numericals], X_ohe], axis=1)
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
def Distribution_after_TrainTestValSplit(y_train_merge,y_test_merge):
    #####Ploting Label Distribution Over Train,test,validation######################
    # it returns a dict, keys as class labels and values as the number of data points in that class
    train_class_distribution = y_train_merge.value_counts().sort_values()
    test_class_distribution = y_test_merge.value_counts().sort_values()
    #cv_class_distribution = y_cv_merge.value_counts().sort_values()

    my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
    train_class_distribution.plot(kind='bar', color=my_colors)
    plt.xlabel('Class')
    plt.ylabel('Data points per Class')
    plt.title('Distribution of yi in train data')
    plt.grid()
    plt.show()

    # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
    # -(train_class_distribution.values): the minus sign will give us in decreasing order
    sorted_yi = np.argsort(-train_class_distribution.values)
    for i in sorted_yi:
        print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_class_distribution.values[i]/y_train_merge.shape[0]*100), 3), '%)')


    print('-'*80)
    my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
    test_class_distribution.plot(kind='bar', color=my_colors)
    plt.xlabel('Class')
    plt.ylabel('Data points per Class')
    plt.title('Distribution of yi in test data')
    plt.grid()
    plt.show()
```

```
    # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
    # -(train_class_distribution.values): the minus sign will give us in decreasing order
    sorted_yi = np.argsort(-test_class_distribution.values)
    """for i in sorted_yi:
        print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_class_distribution.values[i]/y_test_merge.shape[0]*100), 3), '%)')

    print('-'*80)
    my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
    cv_class_distribution.plot(kind='bar', color=my_colors)
    plt.xlabel('Class')
    plt.ylabel('Data points per Class')
    plt.title('Distribution of yi in cross validation data')
    plt.grid()
    plt.show()

    # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
    # -(train_class_distribution.values): the minus sign will give us in decreasing order
    sorted_yi = np.argsort(-train_class_distribution.values)
    for i in sorted_yi:"""
        #print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_distribution.values[i]/y_test_merge.shape[0]*100), 3), '%)')

def standard_scaler(X_train,  X_test,numerical_cols):
    """
    Input: Features (numpy arrays)
    Output: Scaled data
    """
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    return X train scaled,  X test scaled
```

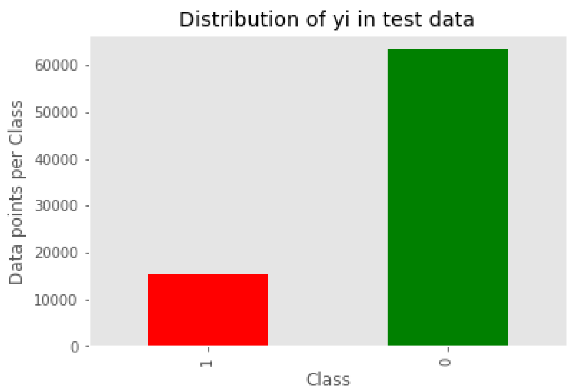
Train and Test data Distribution before SOMTE

In [87]:

```
Distribution after TrainTestValSplit(y train,y test)
```




```
Number of data points in class 2 : 252852 ( 80.333 %)
Number of data points in class 1 : 61904 ( 19.667 %)
--------------------------------------------------------------------------------
```



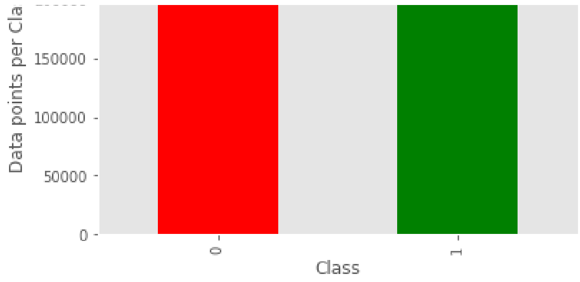
Train and Test data Distribution After SOMTE

In [88]:

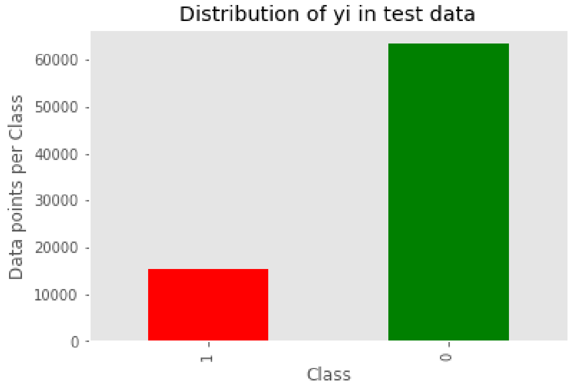
```
sm = SMOTE(random_state=42)
X train smote, y train smote = sm.fit resample(X train scaled, y train)
```

In [89]:

```
Distribution after TrainTestValSplit(y train smote,y test)
```




```
Number of data points in class 1 : 252852 ( 50.0 %)
Number of data points in class 2 : 252852 ( 50.0 %)
--------------------------------------------------------------------------------
```



In [89]:

```

```

In [89]:

```

```

In [89]:

```

```

In [89]:

```

```

In [89]:

```

```

In [89]:

```

```

Modeling

In [90]:

```python
class LogisticRegressionWithThreshold(LogisticRegression):
    def predict(self, X, threshold=None):
        if threshold == None: # If no threshold passed in, simply call the base class predict, effectively threshold=0.5
            return LogisticRegression.predict(self, X)
        else:
            y_scores = LogisticRegression.predict_proba(self, X)[:, 1]
            y_pred_with_threshold = (y_scores >= threshold).astype(int)

            return y_pred_with_threshold

    def threshold_from_optimal_tpr_minus_fpr(self, X, y):
        y_scores = LogisticRegression.predict_proba(self, X)[:, 1]
        fpr, tpr, thresholds = roc_curve(y, y_scores)

        optimal_idx = np.argmax(tpr - fpr)

        return thresholds[optimal_idx], tpr[optimal_idx] - fpr[optimal_idx]

    def threshold_from_optimal_f_score(self, X, y):
        y_scores = LogisticRegression.predict_proba(self, X)[:, 1]
        precisions, recalls, thresholds = precision_recall_curve(y, y_scores)

        fscores = (2 * precisions * recalls) / (precisions + recalls)

        optimal_idx = np.argmax(fscores)

        return thresholds[optimal_idx], fscores[optimal_idx]

    def threshold_from_desired_precision(self, X, y, desired_precision=0.9):
        y_scores = LogisticRegression.predict_proba(self, X)[:, 1]
        precisions, recalls, thresholds = precision_recall_curve(y, y_scores)
```

```
        desired_precision_idx = np.argmax(precisions >= desired_precision)

        return thresholds[desired_precision_idx], recalls[desired_precision_idx]

    def threshold_from_desired_recall(self, X, y, desired_recall=0.9):
        y_scores = LogisticRegression.predict_proba(self, X)[:, 1]
        precisions, recalls, thresholds = precision_recall_curve(y, y_scores)

        desired_recall_idx = np.argmin(recalls >= desired_recall)

        return thresholds[desired_recall_idx], precisions[desired_recall_idx]
```

In [95]:

In [ ]:

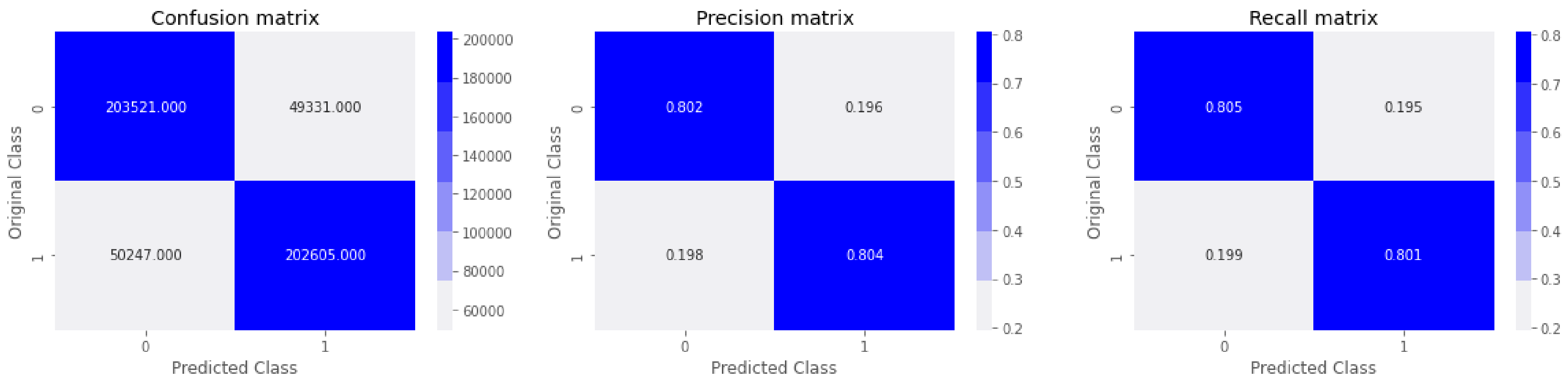
Recall Focus

`Focus is to play safe and ensure that there are no/less NPA generated from the model`

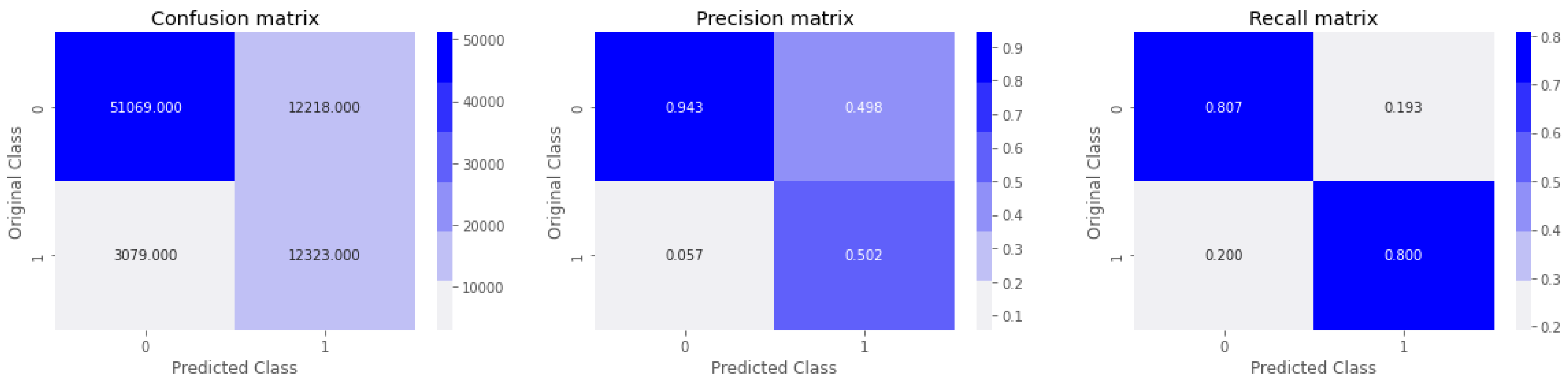
```
Here we need to reduce `False negatives` where Actually Charged off but predicted as Fully payed.

it is very risky or company may go into loss due to this because company giving loans to Charged off customers.

At 0.39 cutoff , we can ensure that we can be more safe and provide loans to only those who are really worth giving but in this mode we loose the business by not taking any risk that you would need to take to gain more business.
```

In [96]:

```
#"Fully Paid":0,"Charged Off":1
lrt = LogisticRegressionWithThreshold()
lrt.fit(X_train_smote, y_train_smote)
y_pred = lrt.predict(X_train_smote)
predicted_y=lrt.predict(X_test_scaled)
plot_confusion_matrix(y_train_smote, y_pred)
print(classification_report(y_train_smote, y_pred))
plot_confusion_matrix(y_test, predicted_y)
print(classification_report(y_test, predicted_y))
```




```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80    252852
           1       0.80      0.80      0.80    252852

    accuracy                           0.80    505704
   macro avg       0.80      0.80      0.80    505704
weighted avg       0.80      0.80      0.80    505704
```



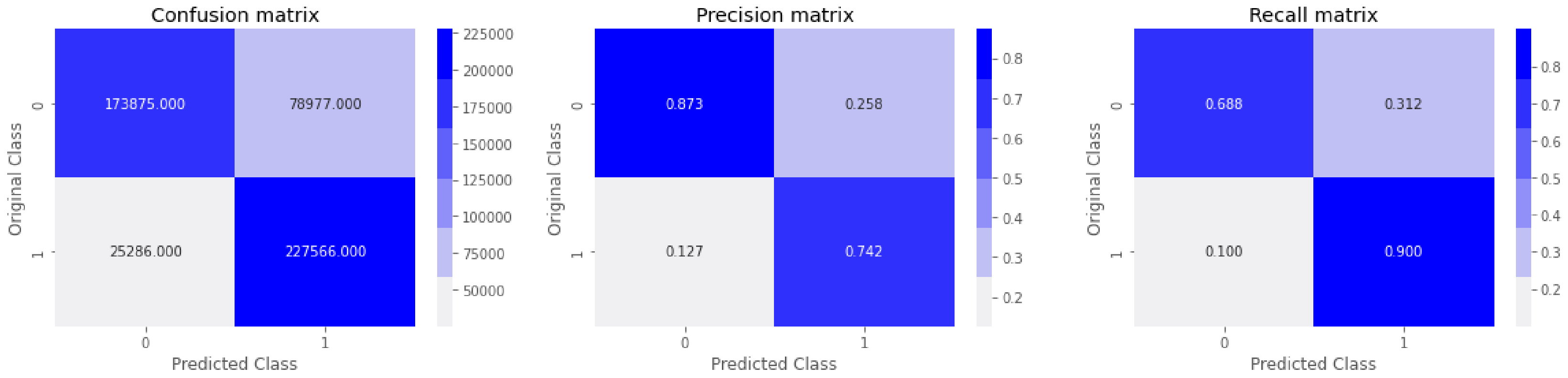

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87     63287
           1       0.50      0.80      0.62     15402

    accuracy                           0.81     78689
```

```
   macro avg       0.72      0.80      0.74     78689
weighted avg       0.86      0.81      0.82     78689
```

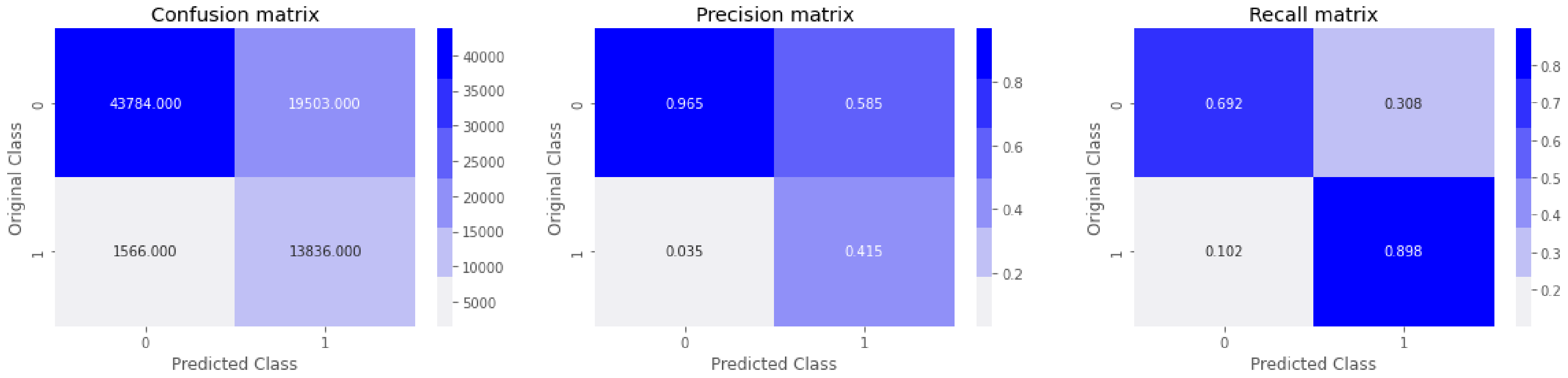
In [97]:

```python
threshold, precision = lrt.threshold_from_desired_recall(X_train_smote, y_train_smote, 0.9)
y_pred = lrt.predict(X_train_smote, threshold)
predicted_y=lrt.predict(X_test_scaled,threshold)
print("threshold : ",threshold)
print('-'*50+'Train Data Performance'+'-'*50)
plot_confusion_matrix(y_train_smote, y_pred)
print(classification_report(y_train_smote, y_pred))

print('-'*50+'Test Data Performance'+'-'*50)
plot_confusion_matrix(y_test, predicted_y)
print(classification_report(y_test, predicted_y))
precision, recall, thresholds = precision_recall_curve(y_test, predicted_y)
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
#display plot
plt.show()
```

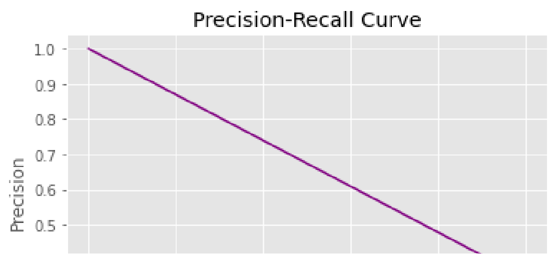
```
threshold :  0.3985921637389079
--------------------------------------------------Train Data Performance--------------------------------------------------
```

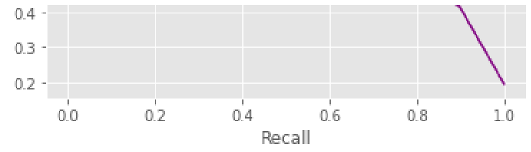



```
              precision    recall  f1-score   support

           0       0.87      0.69      0.77    252852
           1       0.74      0.90      0.81    252852

    accuracy                           0.79    505704
   macro avg       0.81      0.79      0.79    505704
weighted avg       0.81      0.79      0.79    505704

--------------------------------------------------Test Data Performance--------------------------------------------------
```



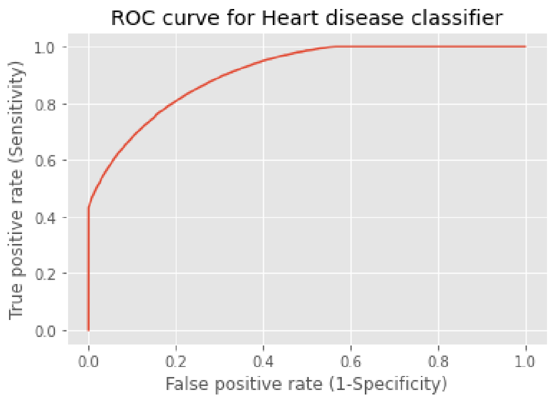

```
              precision    recall  f1-score   support

           0       0.97      0.69      0.81     63287
           1       0.42      0.90      0.57     15402

    accuracy                           0.73     78689
   macro avg       0.69      0.80      0.69     78689
weighted avg       0.86      0.73      0.76     78689
```





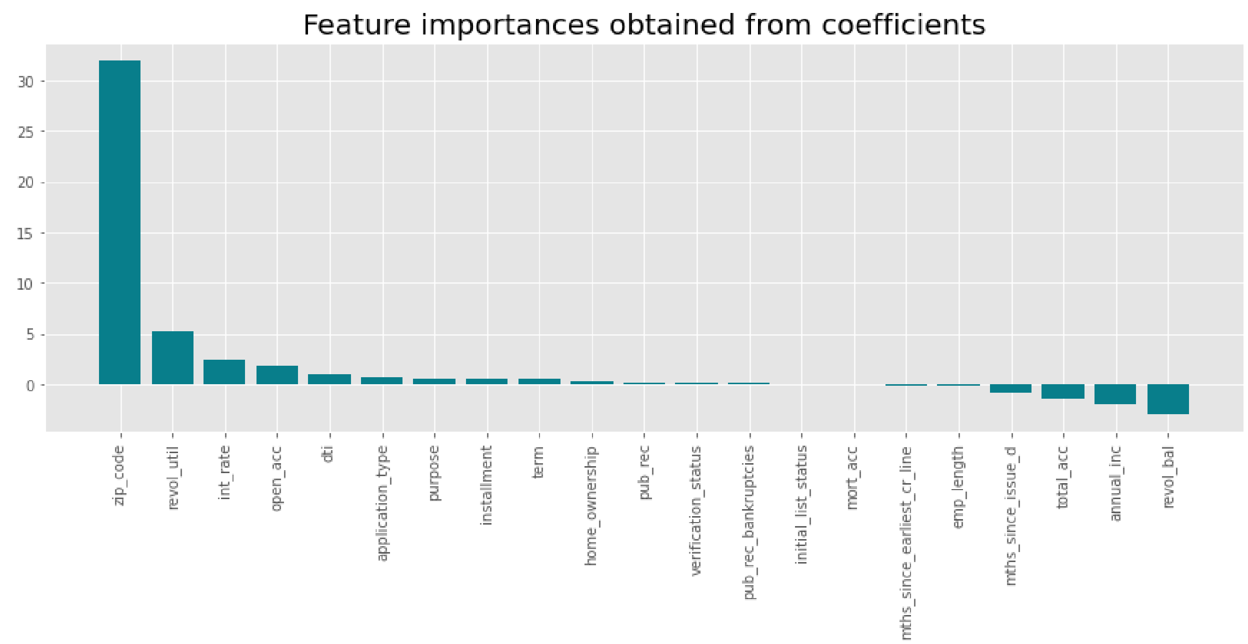
In [98]:

```
from sklearn.metrics import roc_curve
y_pred_prob_yes=lrt.predict_proba(X_test_scaled)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_yes[:,1])
plt.plot(fpr,tpr)
# plt.xlim([0.0, 1.0])
# plt.ylim([0.0, 1.0])
plt.title('ROC curve for Heart disease classifier')
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')
plt.grid(True)
```



In [99]:

```
plt.figure(figsize=(15, 5))

importances = pd.DataFrame(data={
    'Attribute': X_train_scaled.columns,
    'Importance': lrt.coef_[0]
})
importances = importances.sort_values(by='Importance', ascending=False)

plt.bar(x=importances['Attribute'], height=importances['Importance'], color='#087E8B')
plt.title('Feature importances obtained from coefficients', size=20)
plt.xticks(rotation='vertical')
plt.show()
```



In [ ]:

Precession Focus

`Focus to get more intrest on giving loans`

    Here we need to reduce `False positives` where Actually Fully payed but predicted as Charged off.

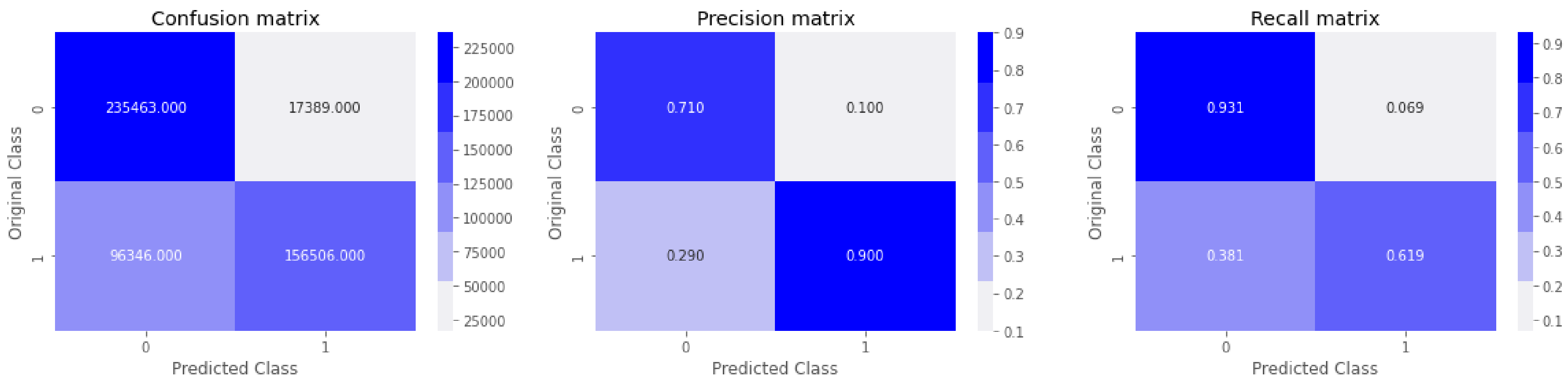
    it is very risky or company may go into loss due to this because company giving loans to Charged off customers.

At point 0.65 we can ensure that we will generate a good business on intrest and not much compromising on the risk involved in giving loans .

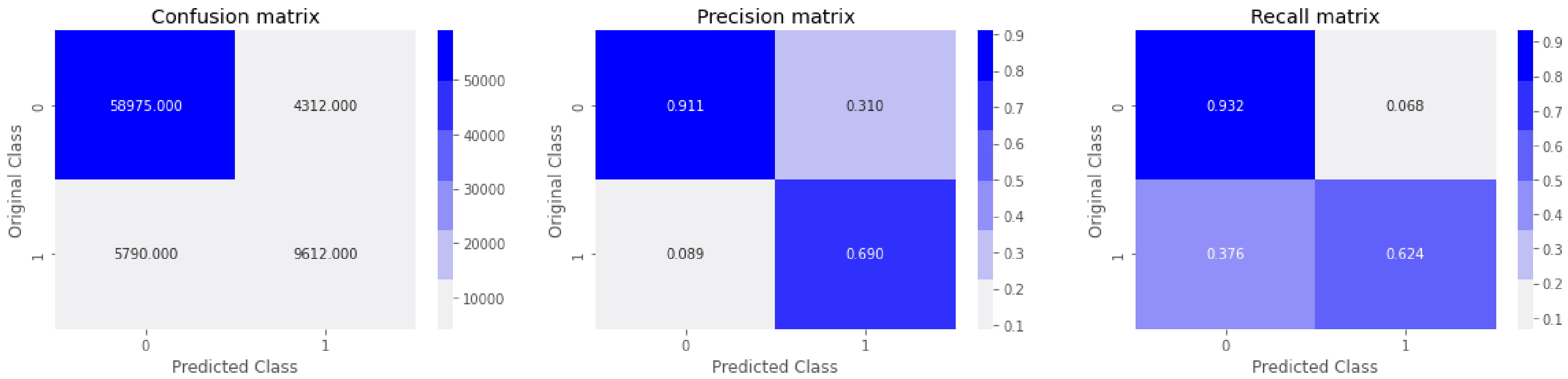
In [103]:

```python
lrt = LogisticRegressionWithThreshold()
lrt.fit(X_train_smote, y_train_smote)

threshold, precision = lrt.threshold_from_desired_precision(X_train_smote, y_train_smote, 0.9)
y_pred = lrt.predict(X_train_smote, threshold)
predicted_y=lrt.predict(X_test_scaled,threshold)

print("threshold",threshold)

print('-'*50+'Train Data Performance'+'-'*50)
plot_confusion_matrix(y_train_smote, y_pred)
print(classification_report(y_train_smote, y_pred))

print('-'*50+'Test Data Performance'+'-'*50)
plot_confusion_matrix(y_test, predicted_y)
print(classification_report(y_test, predicted_y))
precision, recall, thresholds = precision_recall_curve(y_test, predicted_y)
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
#display plot
plt.show()
```

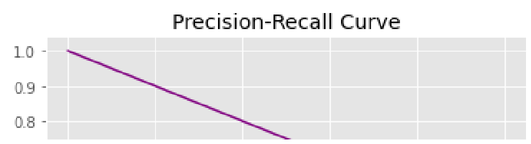
```
threshold 0.6527283838147945
--------------------------------------------------Train Data Performance--------------------------------------------------
```

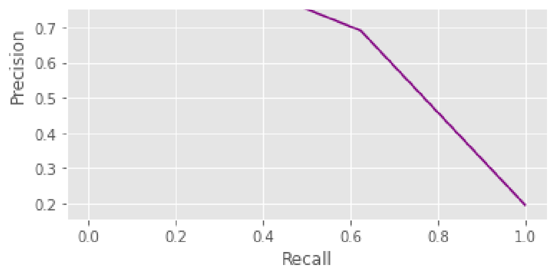



```
              precision    recall  f1-score   support

           0       0.71      0.93      0.81    252852
           1       0.90      0.62      0.73    252852

    accuracy                           0.78    505704
   macro avg       0.80      0.78      0.77    505704
weighted avg       0.80      0.78      0.77    505704

--------------------------------------------------Test Data Performance--------------------------------------------------
```




```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92     63287
           1       0.69      0.62      0.66     15402

    accuracy                           0.87     78689
   macro avg       0.80      0.78      0.79     78689
weighted avg       0.87      0.87      0.87     78689
```

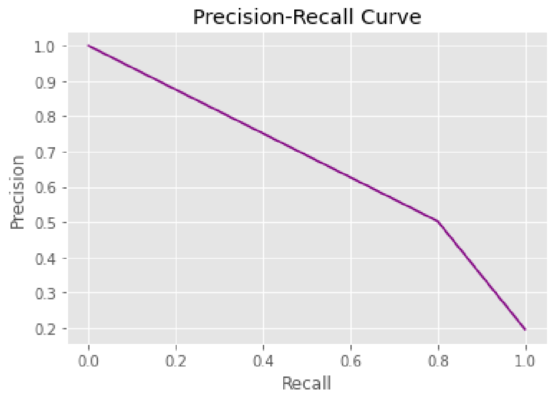




In [106]:

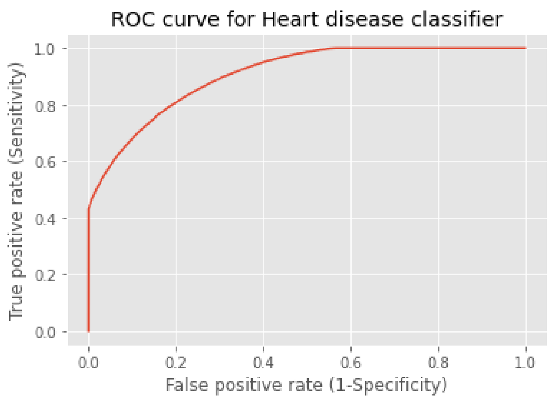
```
predicted_y=lrt.predict(X_test_scaled)
print(classification_report(y_test, predicted_y))

precision, recall, thresholds = precision_recall_curve(y_test, predicted_y)
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
#display plot
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87     63287
           1       0.50      0.80      0.62     15402

    accuracy                           0.81     78689
   macro avg       0.72      0.80      0.74     78689
weighted avg       0.86      0.81      0.82     78689
```



In [107]:

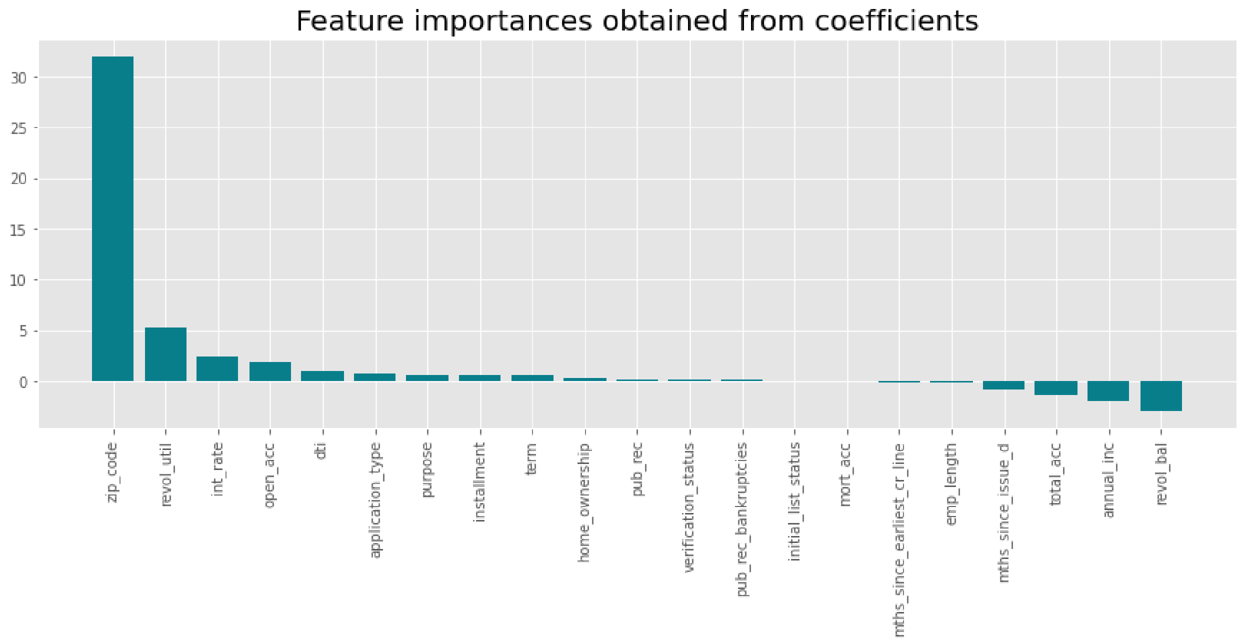
```
from sklearn.metrics import roc_curve
y_pred_prob_yes=lrt.predict_proba(X_test_scaled)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_yes[:,1])
plt.plot(fpr,tpr)
# plt.xlim([0.0, 1.0])
# plt.ylim([0.0, 1.0])
plt.title('ROC curve for Heart disease classifier')
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')
plt.grid(True)
```



In [108]:

```
plt.figure(figsize=(15, 5))

importances = pd.DataFrame(data={
    'Attribute': X_train_scaled.columns,
    'Importance': lrt.coef_[0]
})
importances = importances.sort_values(by='Importance', ascending=False)
```

```
plt.bar(x=importances['Attribute'], height=importances['Importance'], color='#087E8B')
plt.title('Feature importances obtained from coefficients', size=20)
plt.xticks(rotation='vertical')
plt.show()
```



In [ ]:

F1-score Focus

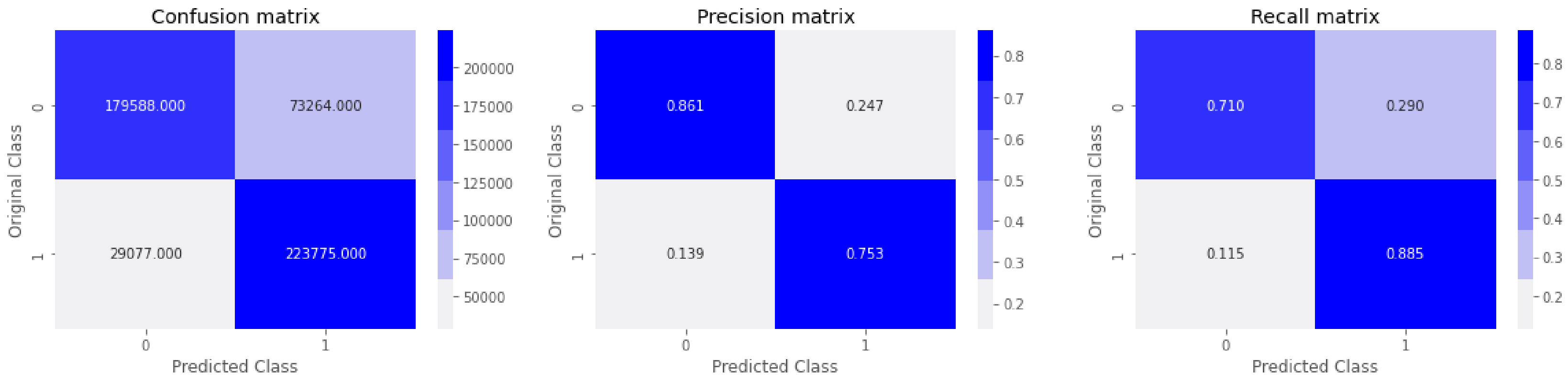
`Focus to get more intrest on giving loans` **and** `Focus is to play safe and ensure that there are no/less NPA generated from the model`

```
Here we need to reduce `False negatives` and `False positives`

At point 0.41 we can try to balanced both risk and generate a good revenue using intrest.
```

In [109]:

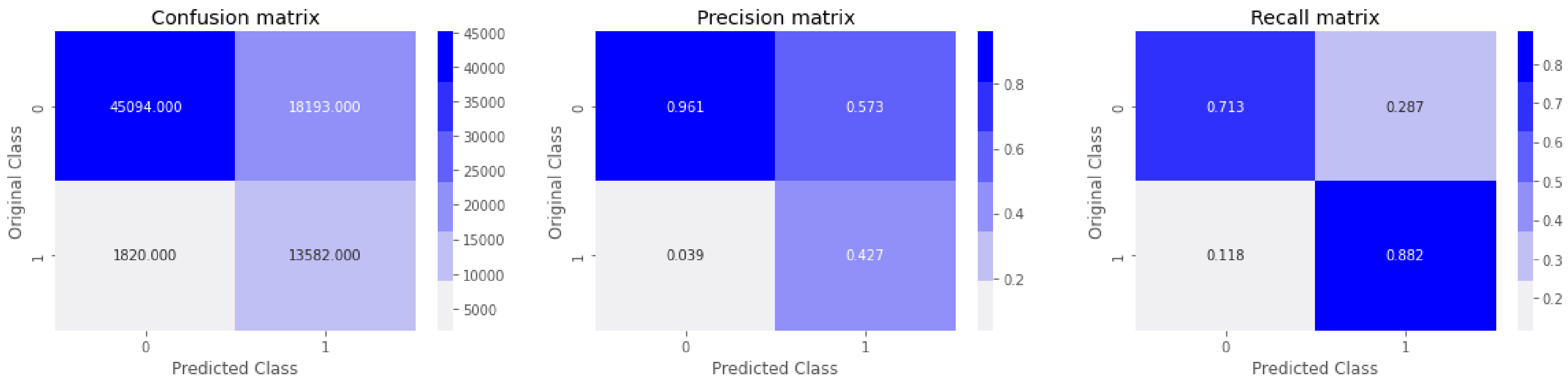
```
lrt = LogisticRegressionWithThreshold()
lrt.fit(X_train_smote, y_train_smote)

threshold, precision = lrt.threshold_from_optimal_f_score(X_train_smote, y_train_smote)
y_pred = lrt.predict(X_train_smote, threshold)
predicted_y=lrt.predict(X_test_scaled,threshold)

print("threshold : ",threshold)

print('-'*50+'Train Data Performance'+'-'*50)
plot_confusion_matrix(y_train_smote, y_pred)
print(classification_report(y_train_smote, y_pred))

print('-'*50+'Test Data Performance'+'-'*50)
plot_confusion_matrix(y_test, predicted_y)
print(classification_report(y_test, predicted_y))
precision, recall, thresholds = precision_recall_curve(y_test, predicted_y)
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
#display plot
plt.show()
```

```
threshold :  0.4167939535986942
--------------------------------------------------Train Data Performance--------------------------------------------------
```

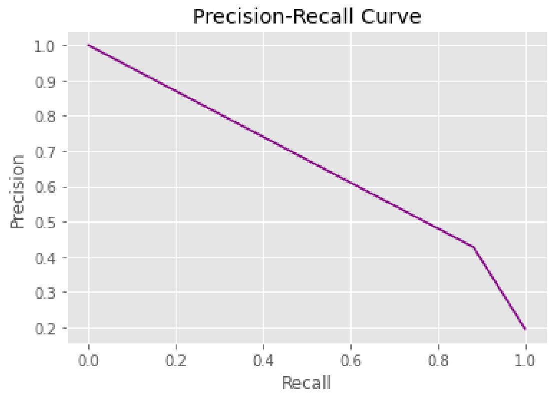


```
              precision    recall  f1-score   support

           0       0.86      0.71      0.78    252852
           1       0.75      0.89      0.81    252852

    accuracy                           0.80    505704
   macro avg       0.81      0.80      0.80    505704
weighted avg       0.81      0.80      0.80    505704
```

```
-------------------------------------------------Test Data Performance-------------------------------------------------
```

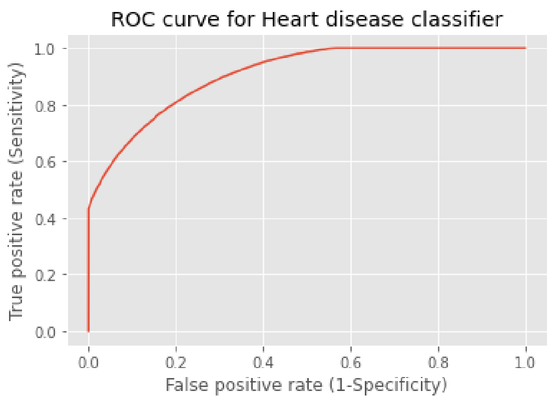



```
              precision    recall  f1-score   support

           0       0.96      0.71      0.82     63287
           1       0.43      0.88      0.58     15402

    accuracy                           0.75     78689
   macro avg       0.69      0.80      0.70     78689
weighted avg       0.86      0.75      0.77     78689
```



In [110]:

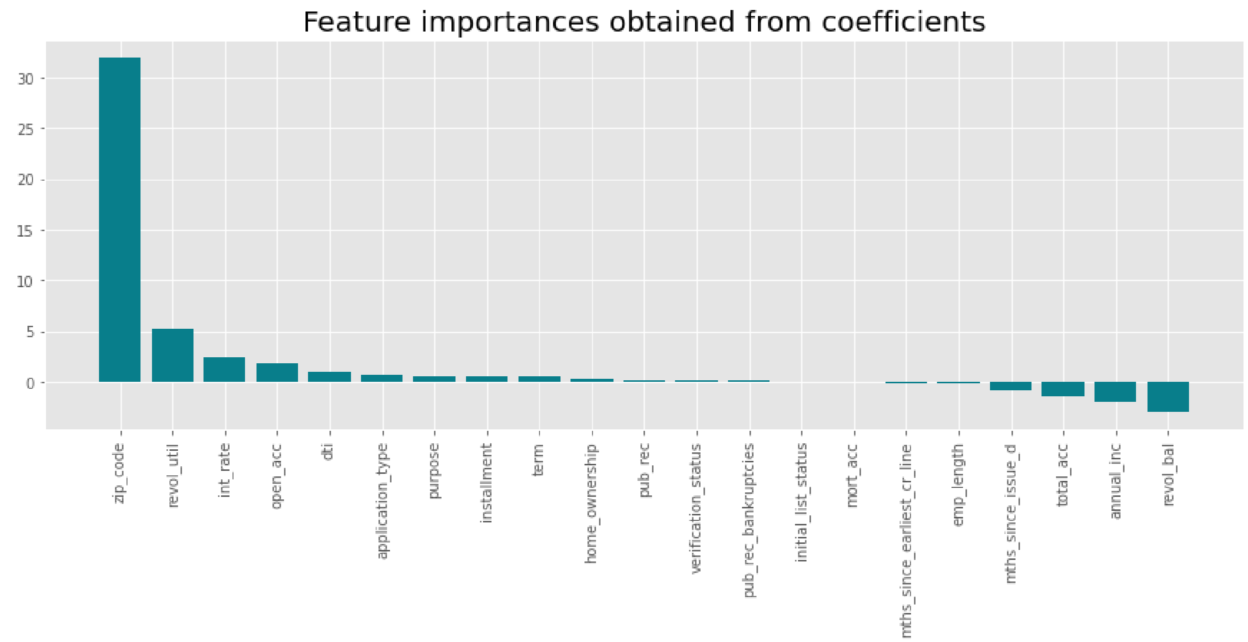
```python
from sklearn.metrics import roc_curve
y_pred_prob_yes=lrt.predict_proba(X_test_scaled)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_yes[:,1])
plt.plot(fpr,tpr)
# plt.xlim([0.0, 1.0])
# plt.ylim([0.0, 1.0])
plt.title('ROC curve for Heart disease classifier')
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')
plt.grid(True)
```



In [111]:

```python
plt.figure(figsize=(15, 5))

importances = pd.DataFrame(data={
    'Attribute': X_train_scaled.columns,
    'Importance': lrt.coef_[0]
})
```

```python
importances = importances.sort_values(by='Importance', ascending=False)

plt.bar(x=importances['Attribute'], height=importances['Importance'], color='#087E8B')
plt.title('Feature importances obtained from coefficients', size=20)
plt.xticks(rotation='vertical')
plt.show()
```



Answers to the Questionnaire:

Tradeoff Questions:

Recall Focus

`Focus is to play safe and ensure that there are no/less NPA generated from the model`

```
Here we need to reduce `False negatives` where Actually Charged off but predicted as Fully payed.

it is very risky or company may go into loss due to this because company giving loans to Charged off customers.

At 0.39 cutoff , we can ensure that we can be more safe and provide loans to only those who are really worth giving but in this mode we loose the business by not taking any risk that you would need to take to gain more business.
```

Precession Focus

`Focus to get more intrest on giving loans`

```
Here we need to reduce `False positives` where Actually Fully payed but predicted as Charged off.

it is very risky or company may go into loss due to this because company giving loans to Charged off customers.

At point 0.65 we can ensure that we will generate a good business on intrest and not much
compromising on the risk involved in giving loans .
```

F1-score Focus

`Focus to get more intrest on giving loans` **and** `Focus is to play safe and ensure that there are no/less NPA generated from the model`

```
Here we need to reduce `False negatives` and `False positives`

At point 0.41 we can try to balanced both risk and generate a good revenue using intrest.
```

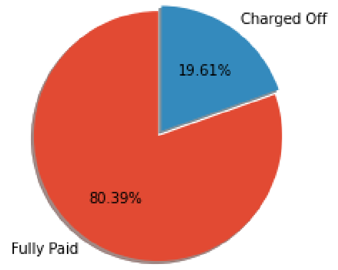
Q1 What percentage of customers have fully paid their Loan Amount?

`80.39 % of the customers have fully paid their loan amount`

In [112]:

```python
loan_tap=pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921")
plt.style.use('ggplot')
data = loan_tap["loan_status"].value_counts(normalize=True)
plt.pie(data, labels=data.index, startangle = 90, shadow = True, radius=1, explode= [0,0.05],autopct='%0.2f%%')
plt.title("Loan Status", fontsize=16, fontweight='bold')
plt.show()
```

Loan Status



Q2 Comment about the correlation between Loan Amount and Installment features

```
It is clear from the heatmap plotted above There is perfect correlation (0.95) between "loan_amnt" the "installment"
```

Q3 The majority of people have home ownership as

`Mortgage and Rent`

In [113]:

```
loan_tap.home_ownership.value_counts()[:5]
```

Out[113]:

```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
Name: home_ownership, dtype: int64
```

Q4 People with grades ‘A’ are more likely to fully pay their loan. (T/F)

`Yes`

In [ ]:

```
pd.crosstab(loan_tap['grade'],loan_tap['loan_status'],normalize='index', margins=True)
```

Out[ ]:

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

Q5 Name the top 2 afforded job titles

`Teacher & Manager`

In [ ]:

```
loan_tap.emp_title.value_counts()[:5]
```

Out[ ]:

```
Teacher              4389
Manager              4250
Registered Nurse     1856
RN                   1846
Supervisor           1830
Name: emp_title, dtype: int64
```

Q6 Thinking from a bank's perspective, which metric should our primary focus be on..ROC AUC or Precision or Recall or F1 Score?

Precision will ensure that there are very less no of False positives which means Prediction is positive and ground truth is negative. if False positives are more then you will be loosing the potential customers who is worth giving loans.

Recall will ensure that there are very less no of false negative i.e prediction is negative and ground truth is postive. when this is the case we may end up giving loans to untrustworthy customers. Hence when we focus on recall this will minmize the risk involved in giving loans to false customers. when focus is recall we will play safe.

F1 score will create a good balance between Precision and recall . if we try to achive the reasonable F1 score then we may not be loosing too many potential customers and not giving loans to too many unworthy customers.

ROC AUC score is not helpful with imbalanced data `

ROC AUC score is not helpful with imbalanced data.

Q7 How does the gap in precision and recall affect the bank?

`Higher the more is the loss either in terms of not gaining intrest or giving loans to bad customers. The gap between precision and recall should be as low as possible.`

Q8 Which were the features that heavily affected the outcome?

Zipcode , revol_inc,int_rate,open_acc are impacting positively.

revol_bal,annual_inc,total_acc are impacting negatively.

Q9 Will the results be affected by geographical location?

`Based on the coefficients obtained we can say- Yes! geographical location turns out to be important feature for the predictions`

Actionable Insights

- **80.39 % of the customers have fully paid their loan amount. Data is heavily imbalanced**
- **Teachers, Managers and Registered Nurses loans most of the loan is availed from these thre profession loans with high intersest rate are more likely to be unpaid.**

annual_inc

mean annual_inc of Fully Paid 95 CI :

```
[71148.63595275 75768.59659525]
```

mean annual_inc of Charged Off 95 CI :

```
[64070.9905965 68153.192662 ]
```

int_rate

mean int_rate of Fully Paid 95 CI :

```
[12.8306405 13.3868295]
```

mean int_rate of Charged Off 95 CI :

```
[15.64542975 16.16089675]
```

- `loans with high intersest rate are more likely to be unpaid`.

dti

mean dti of Fully Paid 95 CI :

```
[16.3511975  17.36024425]
```

mean dti of Charged Off 95 CI :

```
[19.0942805  20.09364375]
```

- **People with grades ‘A’,‘B’,‘C’ are more likely(80% of the time) to fully pay their loan.**
- **F and G subgrades don't get paid back that often**

Recommendations

- **location do matter! Company should be vigilant about where exactly the customer is coming from.**
- **NPA (non-performing asset) is a real problem, it’s important we play safe and shouldn’t disburse loans to anyone and at the same time Focus to get more intrest on giving loans.**
- **As Teachers & Managers have high chunk on borrowers and with decent and good credit worthyness, respective team can work on making stretegies on how to attract customers from other professions as well. *Annual income does a key role and this will definetly a key contributor to decide if someone will pay the loan completely or charged.**
- **We may wish to work on calculating more reasonable interest rates and new intrest metrics what so ever. We have to be rationale on why we are keeping interest rate high? as loans with high interest rates are more likely to be unplaid.**
- **dti_ratio (debt-to-income ratio) is reasonably contributing to deciding if some will charge off or pay the loan fully. if someone not able to get loan due to High debt to income ratio then we can go ahead**

```
In [ ]:
```