Context

LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen. The data science team at LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals. LoanTap deploys formal credit to salaried individuals and businesses 4 main financial instruments: 1.Personal Loan 1.EMI Free Loan 1.Personal Overdraft 1.Advance Salary Loan This case study will focus on the underwriting process behind Personal Loan only

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?

Data dictionary:

1. 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.
2. term : The number of payments on the loan. Values are in months and can be either 36 or 60.
3. int_rate : Interest Rate on the loan
4. installment : The monthly payment owed by the borrower if the loan originates.
5. grade : LoanTap assigned loan grade
6. sub_grade : LoanTap assigned loan subgrade
7. emp_title :The job title supplied by the Borrower when applying for the loan.*
8. 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.
9. home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.
10. annual_inc : The self-reported annual income provided by the borrower during registration.
11. verification_status : Indicates if income was verified by LoanTap, not verified, or if the income source was verified
12. issue_d : The month which the loan was funded
13. loan_status : Current status of the loan - Target Variable
14. purpose : A category provided by the borrower for the loan request.
15. title : The loan title provided by the borrower
16. 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.
17. earliest_cr_line :The month the borrower's earliest reported credit line was opened
18. open_acc : The number of open credit lines in the borrower's credit file.
19. pub_rec : Number of derogatory public records
20. revol_bal : Total credit revolving balance
21. revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
22. total_acc : The total number of credit lines currently in the borrower's credit file
23. initial_list_status : The initial listing status of the loan. Possible values are – W, F

24. application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
25. mort_acc : Number of mortgage accounts.
26. pub_rec_bankruptcies : Number of public record bankruptcies
27. Address: Address of the individual

Concept Used:

1. Exploratory Data Analysis
2. Feature Engineering
3. Logistic Regression
4. Precision Vs Recall Tradeoff

Problem Statement

1. Import the dataset and do usual exploratory data analysis steps like checking the structure & characteristics of the dataset
2. Check how much target variable (Loan_Status) depends on different predictor variables (Use count plots, box plots, heat maps etc)
3. Check correlation among independent variables and how they interact with each other
4. Simple Feature Engineering steps:E.g.: Creation of Flags- If value greater than 1.0 then 1 else 0. This can be done on:
   4.1.Pub_rec
   4.2.Mort_acc
   4.3.Pub_rec_bankruptcies
5. Missing values and Outlier Treatment
6. Scaling - Using MinMaxScaler or StandardScaler
7. Use Logistic Regression Model from Sklearn/Statsmodel library and explain the results
8. Results Evaluation:
9. Classification Report
10. ROC AUC curve
11. Precision recall curve
12. Tradeoff Questions:
13. How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.
14. Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone
15. Provide actionable Insights & Recommendations

Importing the Dataset and Exploratory Data Analysis

In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import warnings
from pandas.core.common import SettingWithCopyWarning
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)
warnings.filterwarnings('ignore')
# parameter grid
```

```
import re
import matplotlib.pyplot as plt
ltap=pd.read_csv("logistic_regression.csv")
ltap_org=ltap.copy()
```

In [2]:

```
print("Total number of Rows :{} \nTotal number of Columns :{}".format(ltap.shape[0],ltap.shape[1])
print(ltap.info())
```

```
Total number of Rows :396030
Total number of Columns :27
<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
None
```

Checking Duplicate Values in Dataset

In [3]:

```
i=ltap.shape[0]
ltap.drop_duplicates(inplace=True)
if i==ltap.shape[0]:
    print("There are no Duplicates in Loan data Set.")
else:
    print("There are " ,ltap.shape[0]-i," Duplicates in the dataset")
```

```
There are no Duplicates in Loan data Set.
```

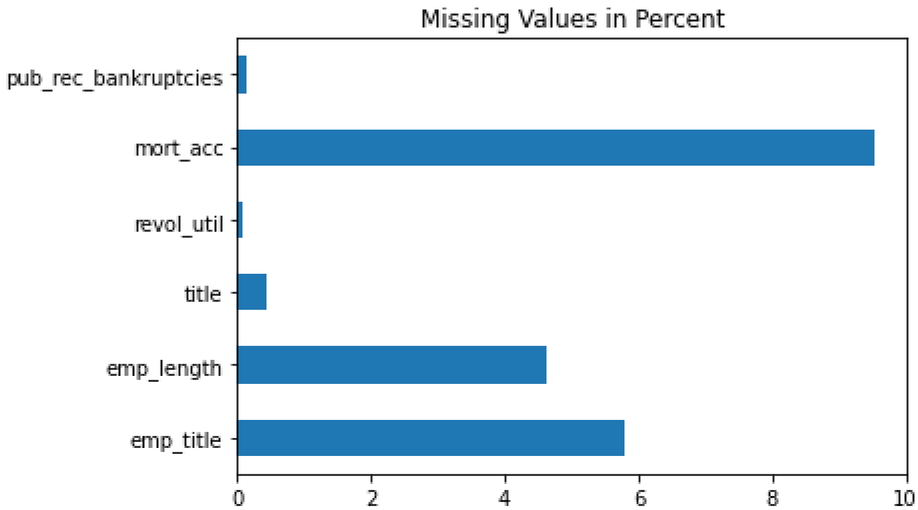
Missing Value and Treatemnt

In [4]:

```
print("Total Rows {} and Total Columns {}".format(ltap.shape[0],ltap.shape[1]))
df=ltap.isna().sum()
round(df[(df>0)]*100/ltap.shape[0],2).plot(kind="barh")
plt.title("Missing Values in Percent")
```

```
plt.show()
print("Columns and Missing Value Counts in Numbers")
print(df[(df>0)])
```

```
Total Rows 396030 and Total Columns 27
NumExpr defaulting to 8 threads.
```




```
Columns and Missing Value Counts in Numbers
emp_title               22927
emp_length              18301
title                    1755
revol_util                276
mort_acc                37795
pub_rec_bankruptcies      535
dtype: int64
```

In [5]:
```
## Count of Missing Values in Different Columns
df[(df>0)]
```

Out[5]:
```
emp_title               22927
emp_length              18301
title                    1755
revol_util                276
mort_acc                37795
pub_rec_bankruptcies      535
dtype: int64
```

In [6]:
```
ltap['emp_title'].value_counts()
```

Out[6]:
```
Teacher                            4389
Manager                            4250
Registered Nurse                   1856
RN                                 1846
Supervisor                         1830
                                   ...
whalen tire                           1
Automobile Club of S. Calif           1
Pharmacy techician                    1
Lot supervisor                        1
rahmani eye institute                 1
Name: emp_title, Length: 173105, dtype: int64
```

Filling the Missing value

In [7]:
```
ltap['emp_title'].fillna('Not Available',inplace=True)

#print(ltap.replace(r'year'," ", regex=True))
ltap['emp_length']=ltap['emp_length'].replace(r'years',"", regex=True)
```

```python
ltap['emp_length']=ltap['emp_length'].replace(r'year',"", regex=True)
ltap['emp_length']=ltap['emp_length'].replace(r'\+',"", regex=True)
ltap['emp_length']=ltap['emp_length'].replace(r'\<',"", regex=True)
ltap['emp_length']=ltap['emp_length'].astype(float)
ltap['emp_length'].fillna(ltap['emp_length'].mean(),inplace=True)
ltap['emp_length']=ltap['emp_length'].round(2)
## Missing title replaced with emp title
ltap['title'].fillna(ltap['emp_title'],inplace=True)

## Filling with Mean revol_util values

ltap['revol_util'].fillna(round(ltap['revol_util'].mean(),2),inplace=True)

## Filling the mort_acc missing value with mean
ltap['mort_acc'].fillna(ltap['mort_acc'].mean(),inplace=True)

## Updated Pub_rec_bankruptcies with mean values
ltap['pub_rec_bankruptcies'].fillna(ltap['pub_rec_bankruptcies'].mean(),inplace=True)

## Removing the months and converting term to integer column
ltap['term']=ltap['term'].replace(r'months',"", regex=True)
ltap['term']=ltap['term'].astype(int)
```

In [8]:

```python
ltap.isnull().sum()
```

Out[8]:

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

Handling Object columns

In [9]:

```python
## Maping the Grade from 0-7 based upon values
codes = {'A':7, 'B':6, 'C':5,'D':4,'E':3,'F':2,'G':1}
ltap['grade'] = ltap['grade'].map(codes)

## All Fully Paid Loans are 1 and partially paid are 0
ltap['loan_status']=ltap['loan_status'].apply(lambda x : 1 if x=='Fully Paid' else 0 )
```

In [10]:
```
ltap['application_type'].value_counts()
```

Out[10]:
```
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: application_type, dtype: int64
```

Chi Square Test to remove unwanted columns

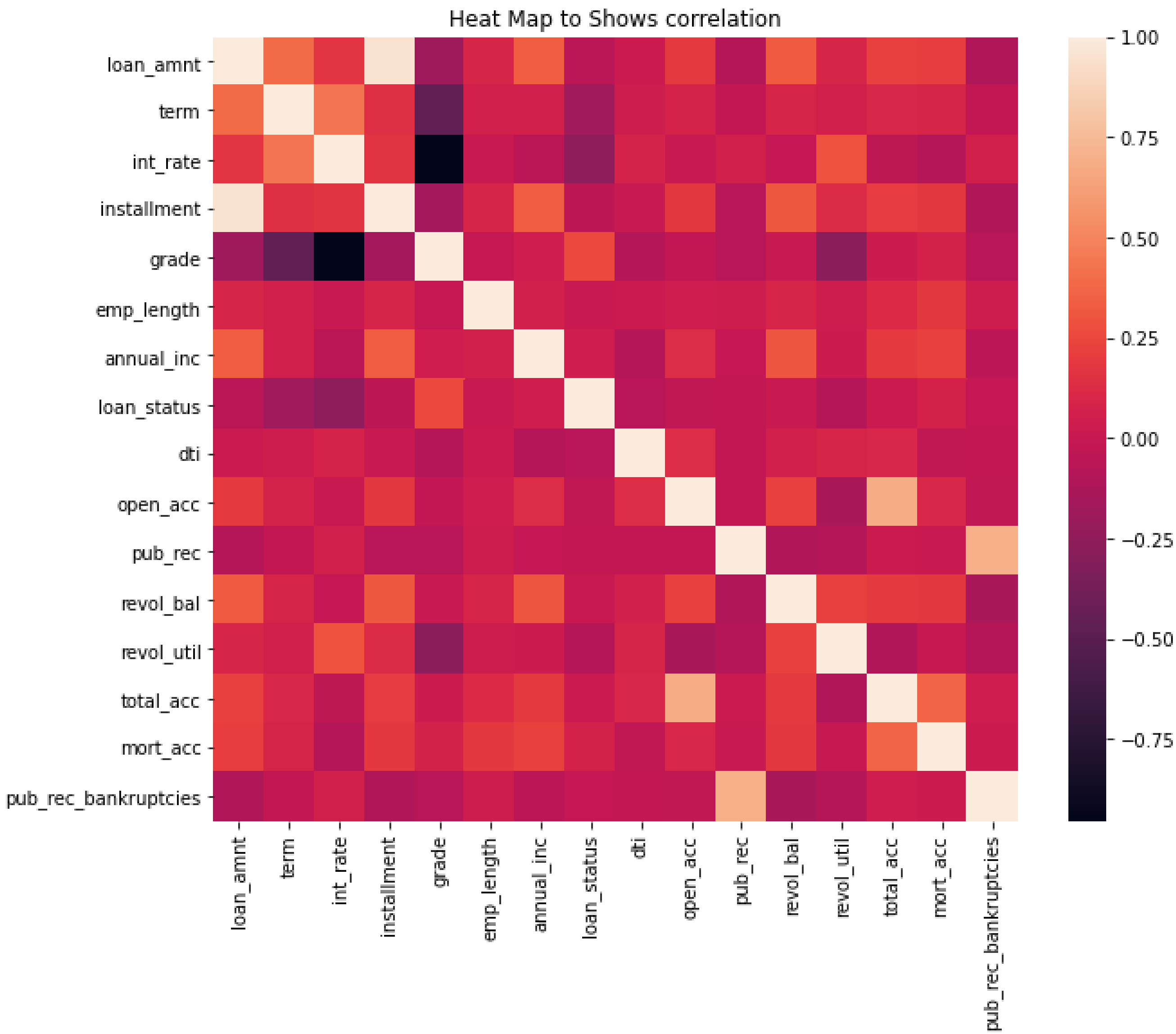
In [11]:
```
from scipy.stats import chi2_contingency

chi2_contingency(pd.crosstab(ltap['home_ownership'], ltap['loan_status'],normalize='index'))

ans={}
for col in ltap.columns:
    stat,p,dof,cont=chi2_contingency(pd.crosstab(ltap[col], ltap['loan_status']))
    if p < 0.5:
        #print(col,"===>",round(p,1))
        ans[col]=round(p,1)

df_chi = pd.DataFrame.from_dict(ans.items())
df_chi.columns=["Columns","Value"]

ans1=df_chi['Columns'].values.tolist()
print("List of Columns with Value 0 ",ans1)
df_chi
```

```
List of Columns with Value 0  ['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grad
e', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
'loan_status', 'purpose', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_util', 'total_a
cc', 'initial_list_status', 'application_type', 'mort_acc', 'pub_rec_bankruptcies', 'address']
```

Out[11]:

	Columns	Value
0	loan_amnt	0.0
1	term	0.0
2	int_rate	0.0
3	installment	0.0
4	grade	0.0
...	...	...
20	initial_list_status	0.0
21	application_type	0.0
22	mort_acc	0.0
23	pub_rec_bankruptcies	0.0
24	address	0.2

In [12]:
```
plt.figure(figsize=(10,8))
sns.heatmap(ltap.corr())
plt.title("Heat Map to Shows correlation")
plt.show()
```



Observations

1. Interest rate and Grade shows a correlation of 1 either one of them to be used
2. Term show very high correlation with Grade .
3. Grade has high correlation between loan amount,interest_rate and term.

In [13]:

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

Out[13]:

```
Oct-2000    3017
Aug-2000    2935
Oct-2001    2896
Aug-2001    2884
Nov-2000    2736
            ...
Jul-1960       1
Dec-1956       1
Sep-1961       1
Sep-1960       1
Jul-1955       1
Name: earliest_cr_line, Length: 684, dtype: int64
```

Mean Encoding of Object Columns and droping of columns

In [14]:

```
## Mean Encoding of application_type
Mean_encoded_loan_status     = ltap.groupby(['application_type'])['loan_status'].mean().to_dict()
```

```
ltap['application_type'] = ltap['application_type'].map(Mean_encoded_loan_status)

## Mean Encoding of Sub_grade
Mean_encoded_sub_grade     = ltap.groupby(['sub_grade'])['loan_status'].mean().to_dict()
ltap['sub_grade'] = ltap['sub_grade'].map(Mean_encoded_sub_grade)

## Mean Encoding of Sub_grade
Mean_encoded_home_ownership     = ltap.groupby(['home_ownership'])['loan_status'].mean().to_dict()
ltap['home_ownership'] = ltap['home_ownership'].map(Mean_encoded_home_ownership)
ltap['home_ownership'] = ltap['home_ownership'].round(2)

## Mean Encoding of Sub_grade
Mean_encoded_verification_status     = ltap.groupby(['verification_status'])['loan_status'].mean()
ltap['verification_status'] = ltap['verification_status'].map(Mean_encoded_verification_status)

## Mean Encoding of purpose
Mean_encoded_purpose     = ltap.groupby(['purpose'])['loan_status'].mean().to_dict()
ltap['purpose'] = ltap['purpose'].map(Mean_encoded_purpose)

## Mean Encoding of purpose
Mean_encoded_purpose     = ltap.groupby(['purpose'])['loan_status'].mean().to_dict()
ltap['purpose'] = ltap['purpose'].map(Mean_encoded_purpose)
```

In [15]:

```
## extracting only state from address columns and droping the orginal address column
ltap['address1'] = ltap['address'].str[-8:]
ltap['address1']=ltap['address1'].apply(lambda x :" ".join(re.findall("[a-zA-Z]+", x)))
ltap.drop('address',axis=1,inplace=True)

Mean_encoded_address1     = ltap.groupby(['address1'])['loan_status'].mean().to_dict()
ltap['address1'] = ltap['address1'].map(Mean_encoded_address1)
```

In [16]:

```
## Droping various columns in the dataframe
ltap.drop('issue_d',axis=1,inplace=True)
ltap.drop('initial_list_status',axis=1,inplace=True)
ltap.drop('emp_title',axis=1,inplace=True)
ltap.drop('title',axis=1,inplace=True)
ltap.drop('earliest_cr_line',axis=1,inplace=True)
```

Checking Duplicate Values in Dataset

In [17]:

```
i=ltap.shape[0]
ltap.drop_duplicates(inplace=True)
if i==ltap.shape[0]:
    print("There are no Duplicates in Data Set:")
#yul.describe(include='object')
```

```
There are no Duplicates in Data Set:
```

Checking on Outliers in Dataset

1. Below Data Frame Shows the Outliers for Entire Dataset .

In [18]:

```
## Data Frame which takes input as dataframe and return the columns with numeric value as datafram
def Quartile(df):
    L1=df.select_dtypes(include=np.number).columns.tolist()
    df1=pd.DataFrame(columns=['Q1','Q2(Median)','Q3','IQR','Lower_Bound','Upper_Bound','Outlier_lc
    n=len(df)
    for i in range(len(df1.index)):
```

```
        q1 = round(df[L1[i]].quantile(0.25),1)
        q2 = round(df[L1[i]].quantile(0.50),1)
        q3 = round(df[L1[i]].quantile(0.75),1)
        iqr = q3 - q1
        lower_bound = q1 -(1.5 * iqr)
        upper_bound = q3 +(1.5 * iqr)
        low_out=round(len(df[df[L1[i]]<lower_bound])/n,3)
        hi_out=round(len(df[df[L1[i]]>upper_bound])/n,3)
        df1.iloc[i]=[q1,q2,q3,iqr,lower_bound,upper_bound,low_out,hi_out]
        #return (q1,q2,q3,iqr,lower_bound,upper_bound)
        #df1=df1.astype(int)
    return df1

df=Quartile(ltap)
df.reset_index()
df
```

Out[18]:

	Q1	Q2(Median)	Q3	IQR	Lower_Bound	Upper_Bound	Outlier_lower%	Outlier_up
loan_amnt	8000.0	12000.0	20000.0	12000.0	-10000.0	38000.0	0.0	
term	36.0	36.0	36.0	0.0	36.0	36.0	0.0	
int_rate	10.5	13.3	16.5	6.0	1.5	25.5	0.0	
installment	250.3	375.4	567.3	317.0	-225.2	1042.8	0.0	
grade	4.0	5.0	6.0	2.0	1.0	9.0	0.0	
...	...	...	...	...	...	...	...	
total_acc	17.0	24.0	32.0	15.0	-5.5	54.5	0.0	
application_type	0.8	0.8	0.8	0.0	0.8	0.8	0.001	
mort_acc	0.0	1.0	3.0	3.0	-4.5	7.5	0.0	
pub_rec_bankruptcies	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
address1	0.8	0.8	0.8	0.0	0.8	0.8	0.176	

In [19]:

```
df[(df['Outlier_lower%']>0)|(df['Outlier_upper%']>0)]
```

Out[19]:

	Q1	Q2(Median)	Q3	IQR	Lower_Bound	Upper_Bound	Outlier_lower%	Outlier_u
term	36.0	36.0	36.0	0.0	36.0	36.0	0.0	
int_rate	10.5	13.3	16.5	6.0	1.5	25.5	0.0	
installment	250.3	375.4	567.3	317.0	-225.2	1042.8	0.0	
home_ownership	0.8	0.8	0.8	0.0	0.8	0.8	0.499	
annual_inc	45000.0	64000.0	90000.0	45000.0	-22500.0	157500.0	0.0	
...	...	...	...	...	...	...	...	
total_acc	17.0	24.0	32.0	15.0	-5.5	54.5	0.0	
application_type	0.8	0.8	0.8	0.0	0.8	0.8	0.001	
mort_acc	0.0	1.0	3.0	3.0	-4.5	7.5	0.0	
pub_rec_bankruptcies	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
address1	0.8	0.8	0.8	0.0	0.8	0.8	0.176	

In [20]:

```
df1=((df['Outlier_lower%']>0)|(df['Outlier_upper%']>0).tolist())
df1=df1[df1]
```

In [21]:

```
cols=['int_rate','installment','home_ownership','annual_inc','loan_status','purpose','dti','open_a
'total_acc','application_type','mort_acc','pub_rec_bankruptcies','address1']
```

In [22]:

```
df1
```

Out[22]:

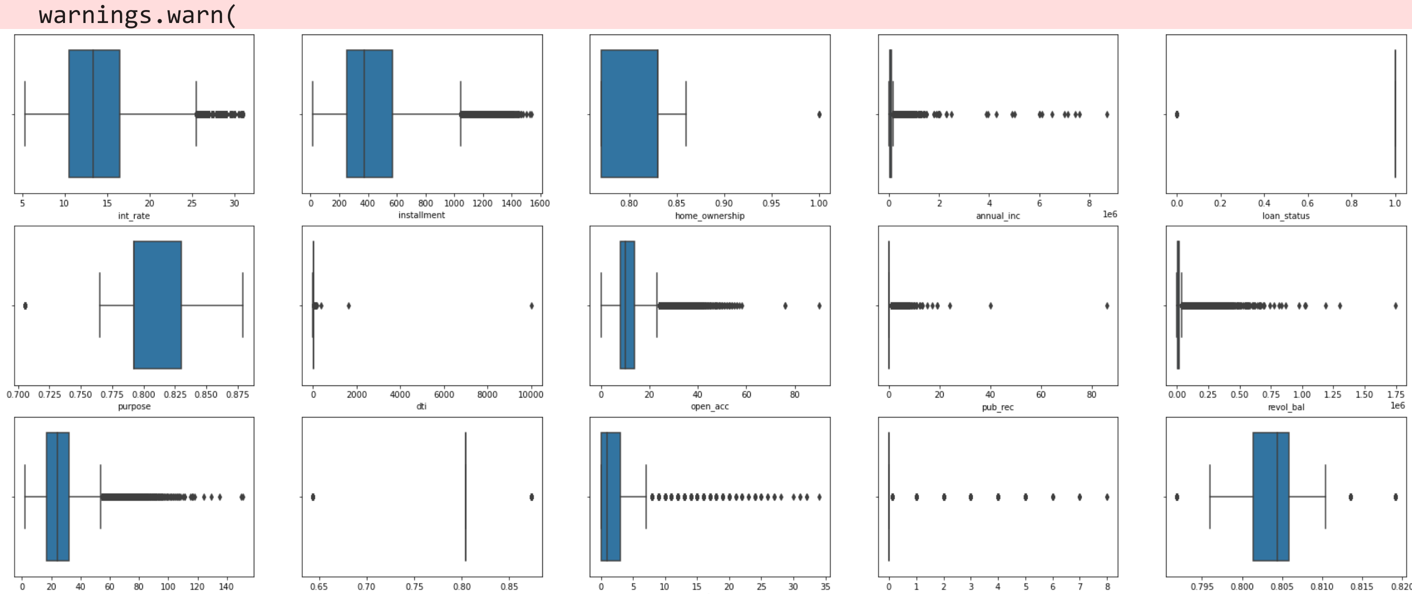
```
term                    True
int_rate                True
installment             True
home_ownership          True
annual_inc              True
loan_status             True
purpose                 True
dti                     True
open_acc                True
pub_rec                 True
revol_bal               True
total_acc               True
application_type        True
mort_acc                True
pub_rec_bankruptcies    True
address1                True
Name: Outlier_lower%, dtype: bool
```

In [23]:

```
plt.figure(figsize=(30,12))
plt.subplot(5,3,5)

for i in range(len(cols)):
    plt.subplot(3,5,i+1)
    sns.boxplot(ltap[cols[i]])
    plt.title
```

```
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
```

```
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
C:\Users\naren\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the foll
owing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
```



In [25]:

```
cols=['int_rate','installment',
```

```
'home_ownership',
'annual_inc',
'loan_status',
'purpose',
'dti',
'open_acc',
'pub_rec',
'revol_bal',
'total_acc',
'application_type',
'mort_acc',
'pub_rec_bankruptcies',
'address1']

Q=ltap.shape[0]
print("No of Rows before Removing Outliers is ",ltap.shape[0])
Q1 = ltap[cols].quantile(0.05)
Q3 = ltap[cols].quantile(0.95)
IQR = Q3 - Q1

ltap = ltap[~((ltap[cols] < (Q1 - 1.5 * IQR)) |(ltap[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]


print("Rows Removed :",Q-ltap.shape[0],"Percentage of Rows Removed :",round(((Q -ltap.shape[0])/Q
```

```
No of Rows before Removing Outliers is  396030
Rows Removed : 13064 Percentage of Rows Removed : 3.3  Total Rows After Removal: 382966
```

Observations

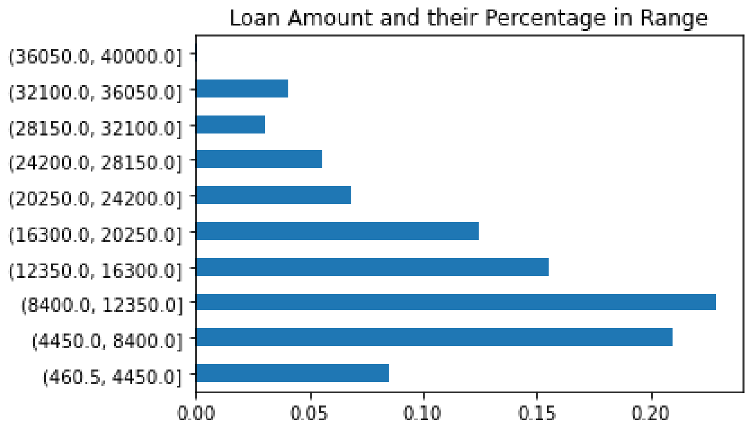
1. Putting the standard value as .25 and .75 as lower and upper bound removes around 45% of outliers which is valueble information loss
2. Have changed the percentage to 5% and 95% for lower and upper bound

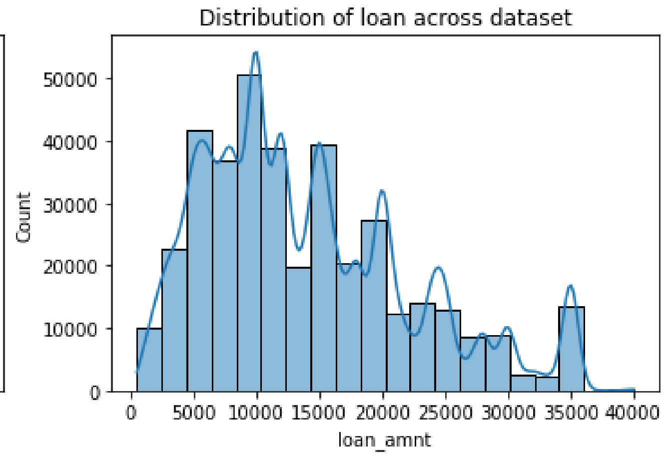
Univariate Analysis and Bivarety

loan_amnt

In [26]:

```
plt.figure(figsize=(12,8))
#plt.subplot(2,2,2)

plt.subplot(2,2,1)
pd.cut(ltap['loan_amnt'], bins = 10).value_counts(normalize=True,).sort_index().plot(kind='barh')
plt.title("Loan Amount and their Percentage in Range")

plt.subplot(2,2,2)
sns.histplot(ltap['loan_amnt'],bins=20,kde=True)
plt.title("Distribution of loan across dataset")
plt.show()
```





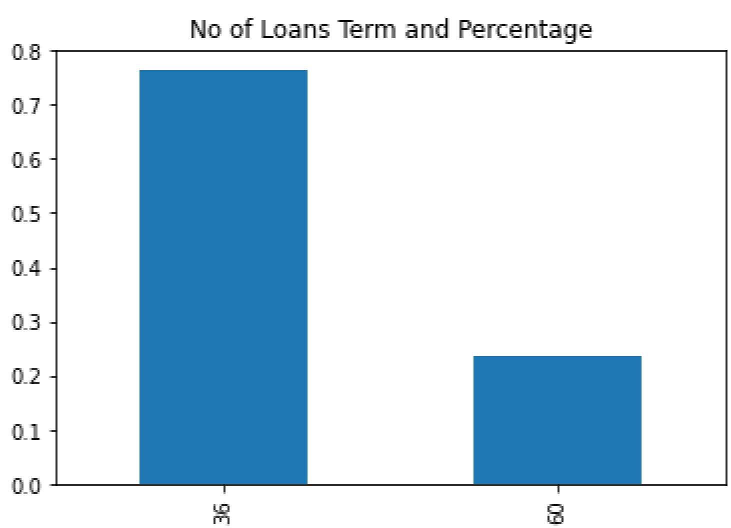
Observation Loan Amount

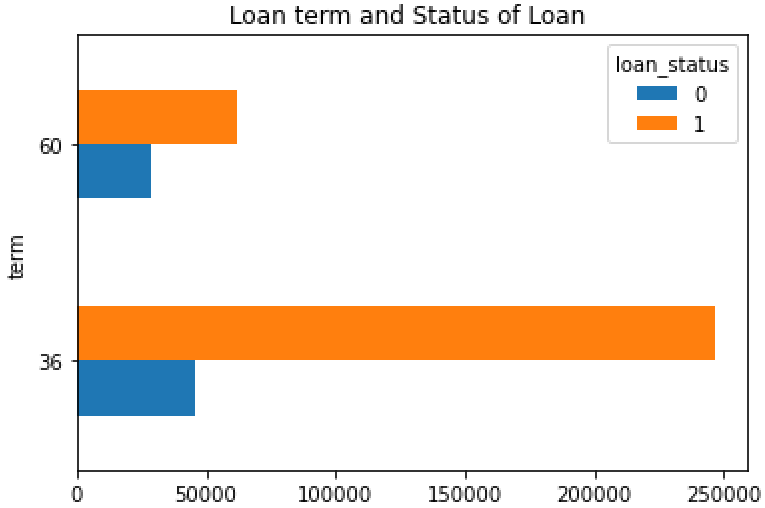
1. Maximum Loan is between 8400 -12350 which around 25% of all Loans
2. Higher Amount Loan is less than 10% .
3. Maximum Loans are betweeeen 5K -15K
4. Company Should focus in above category more , may charge fees to earn more revenue
5. They can also get differntial rates with higher amount less processing fees and vice versa

Term

In [27]:

```
ltap['term'].value_counts(normalize=True).plot(kind='bar')
plt.title("No of Loans Term and Percentage")

pd.crosstab(ltap['term'],ltap['loan_status']).plot(kind='barh')
plt.title("Loan term and Status of Loan ")
plt.show()
```





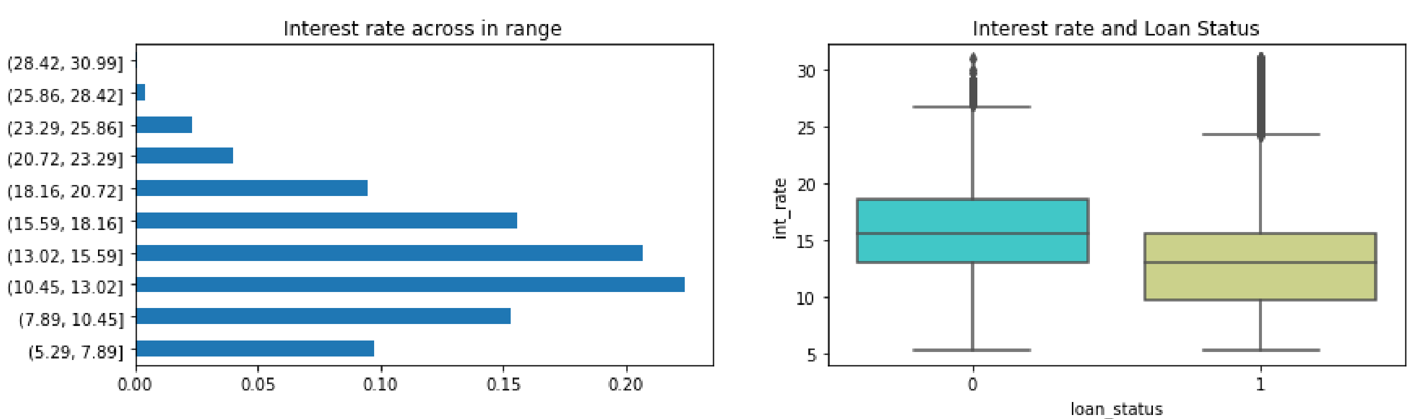
Observation Loan Term

1. Loan with tenure of 36 are high percentage of full paid (status 1)
2. Loan tenure 36 is 75% of data 60 Months are 25%
3. Loan Tap should focus shorter loan term for better profitability
4. High Loan term has high defult ratio as well so loantap should have additional check for loan tenure in excess of 36 Months
5. Loan term is just fixed at 36 and 60 month , more terms like 24 Months or 12 Month should be introduced for increasing buisness.

Interest Rate

In [28]:

```
plt.figure(figsize=(14,8))

plt.subplot(2,2,1)
pd.cut(ltap['int_rate'],bins=10,precision=2).value_counts(normalize=True).sort_index().plot(kind='
plt.title("Interest rate across in range")

plt.subplot(2,2,2)
sns.boxplot(x='loan_status',y='int_rate',data=ltap, palette='rainbow')
plt.title("Interest rate and Loan Status")

plt.show()
```



Observation Interest rate

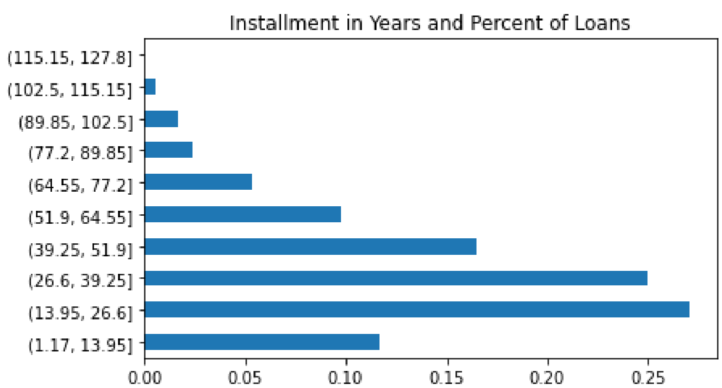
1. Lower Interest rate does not mean more loans
2. People are are taking more loans between 10-20% interest rate .

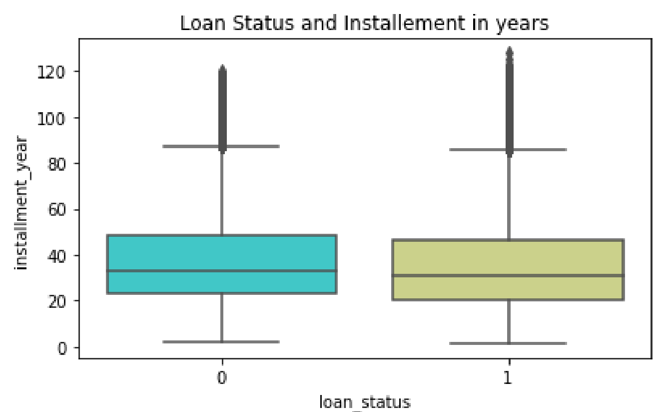
3. Most of low interest rate loan are fully paid as comparied high interst rate loans
4. Profitability can be increased by charging fees for interest between 10-20 range .

Installment

In [29]:
```
## New Columns Installment_year
ltap['installment_year']=round(ltap['installment']/12,1)
```

In [30]:
```
plt.figure(figsize=(14,8))
plt.subplot(2,2,1)

pd.cut(ltap['installment_year'],bins=10,precision=2).value_counts(normalize=True).sort_index().plo
plt.title("Installment in Years and Percent of Loans")

plt.subplot(2,2,2)
sns.boxplot(x='loan_status',y='installment_year',data=ltap, palette='rainbow')
plt.title("Loan Status and Installement in years")
plt.show()
```





Observation Interest rate

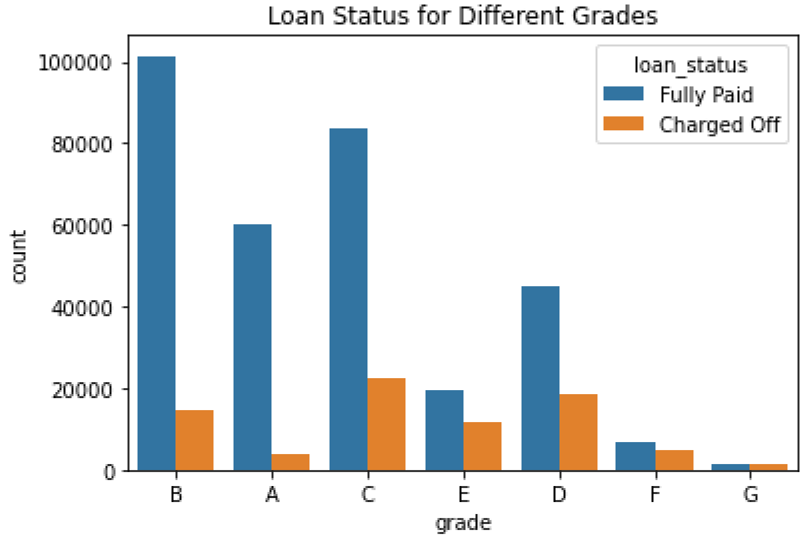
1. Most Loans has installement between 14 to 26 Years
2. Installment 1-14 is on lower side
3. There are outliers for installments above 60 years as they donot look realisitic
4. Fully paid and charged off loan has almost same median value.
5. No of Years greater than 60 year looks abnormal as the loan are not granted for such a long period.

Grade

In [31]:
```
ltap['loan_status'].value_counts()
# Here 0 is partially paid and 1 is fully paid loans
```

Out[31]:
```
1    308279
0     74687
Name: loan_status, dtype: int64
```

In [32]:
```
sns.countplot(data=ltap_org,x='grade',hue='loan_status')
plt.title("Loan Status for Different Grades")
plt.show()
```

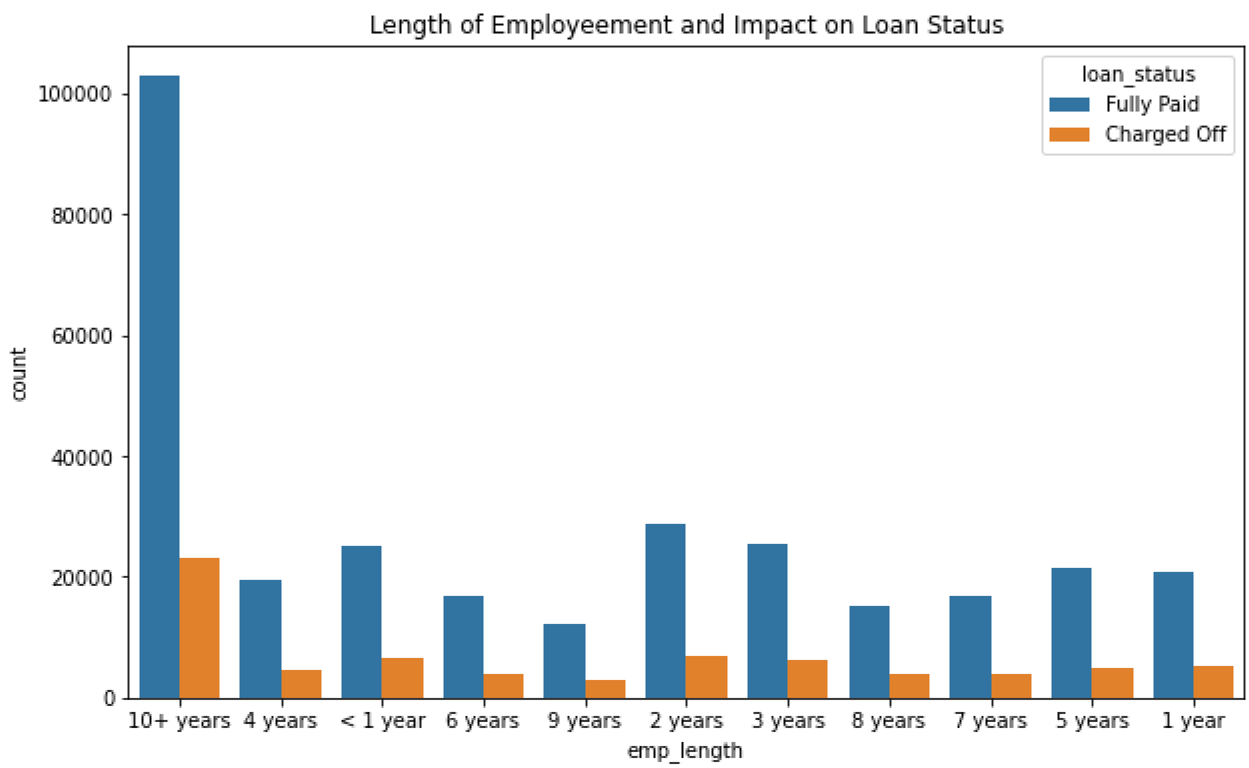


Observation Grade

1. Grade A-G were changed 7-1
2. Grade 5-6-7 has maximum people fully paying loans
3. 1-4 has lower percent of people paying loans fully
4. Lower grades has approx 30-50% partially paid loans

In [33]:

```
plt.figure(figsize=(10,6))
sns.countplot(data=ltap_org,x='emp_length',hue='loan_status')
plt.title("Length of Employeement and Impact on Loan Status")
plt.show()
```



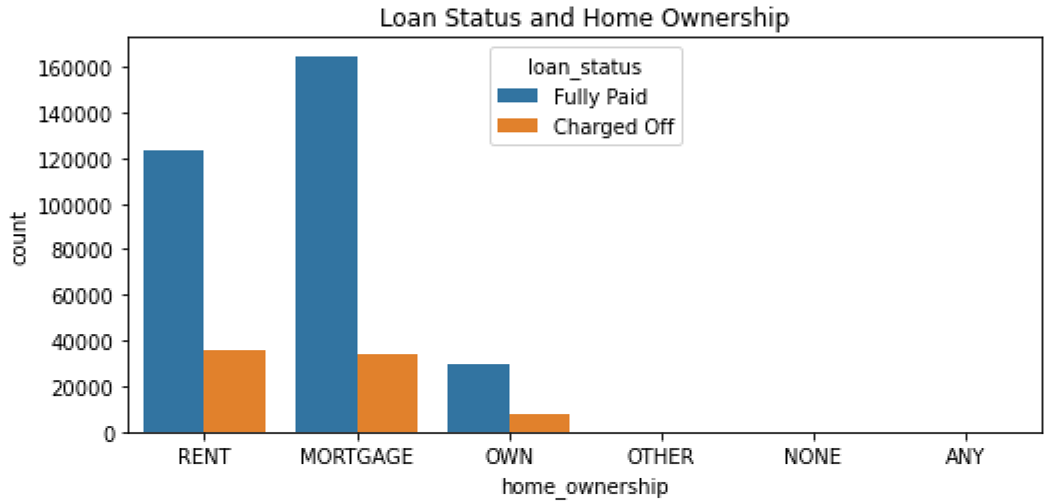
Observation Length of Employement

1. Fully Paid loans are highest when length of employement is 10+ Year
2. Middle level 2-9 the Unpaid loans are highest
3. More focus either on Short term or very long term loan for profitability.

home_ownership

In [34]:

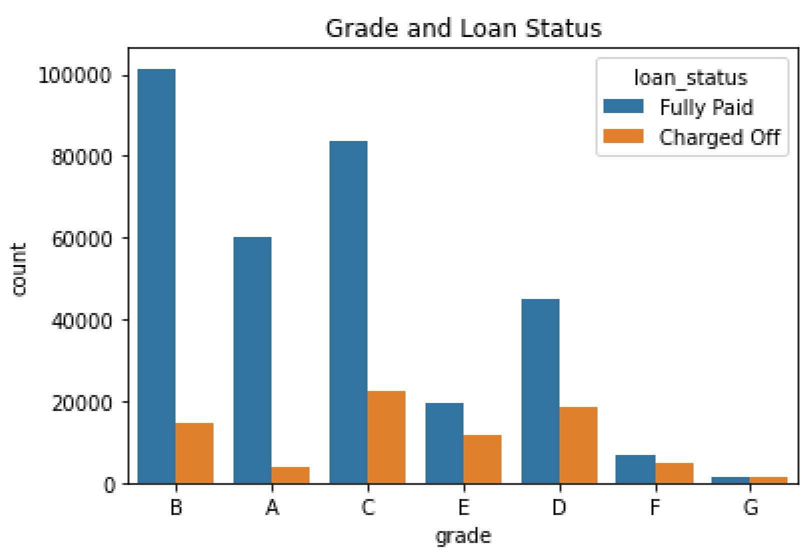
```
plt.figure(figsize=(18,8))
plt.subplot(2,2,1)
sns.countplot(data=ltap_org,x='home_ownership',hue='loan_status')
plt.title("Loan Status and Home Ownership")
plt.show()
```



Grade

In [35]:

```
sns.countplot(data=ltap_org,x='grade',hue='loan_status')
plt.title("Grade and Loan Status")
plt.show()
```



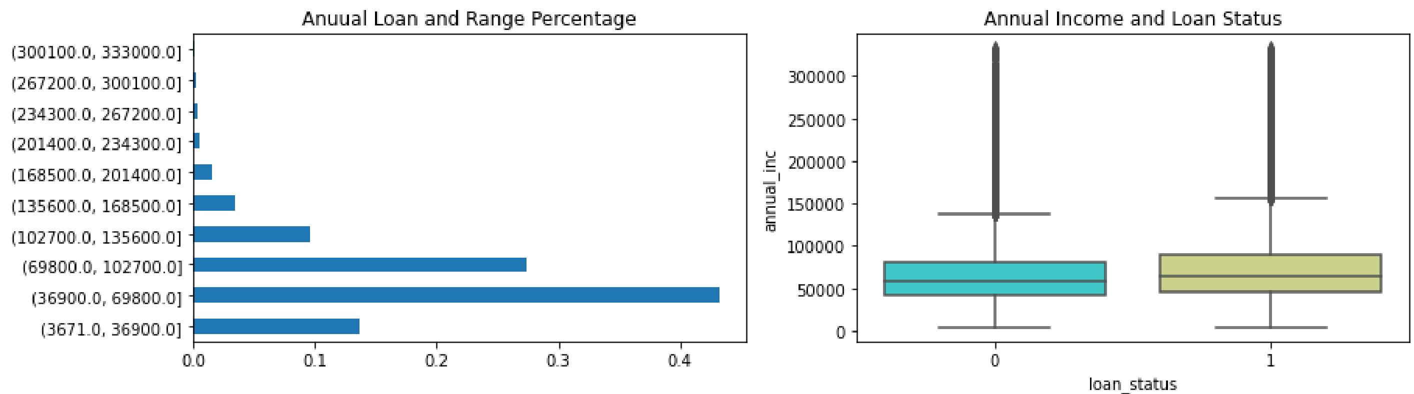
Observation on Grade and Home Ownership

1. Most Loand in A, B and C grade have high ration of fully paid
2. Grade F and G has lowest fully paid status
3. for Mortgage and rent have high fully paid loans .

In [36]:

```
plt.figure(figsize=(14,8))
plt.subplot(2,2,1)
pd.cut(ltap['annual_inc'],bins=10,precision=2).value_counts(normalize=True).sort_index().plot(kinc
plt.title("Anuual Loan and Range Percentage")
```

```
plt.subplot(2,2,2)
sns.boxplot(x='loan_status',y='annual_inc',data=ltap, palette='rainbow')
plt.title("Annual Income and Loan Status")
plt.show()
plt.show()
```

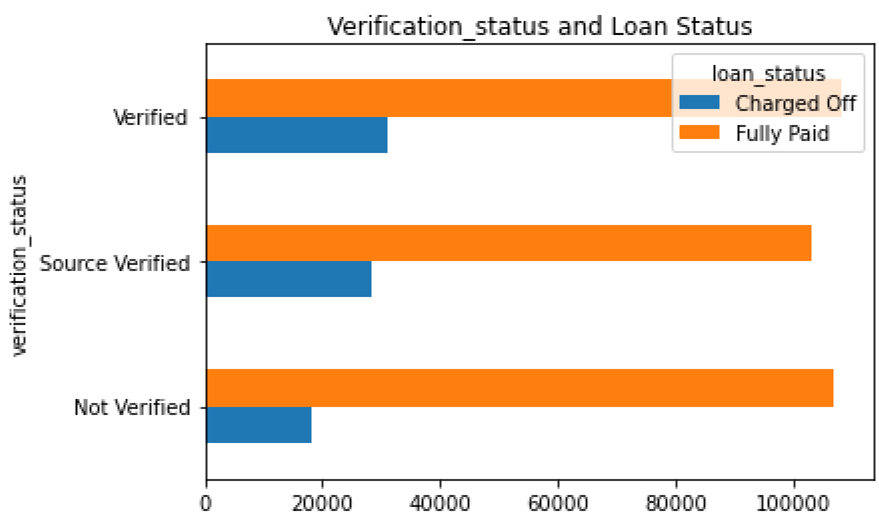


Observation on Grade and Home Ownership

1. The higest percentage in lower income group 3671 to 69800
2. Annual Income and Loan status and equal median meaning people are paying and paying are same leven for different income group

verification_status

In [37]:
```
pd.crosstab(ltap_org['verification_status'],ltap_org['loan_status']).plot(kind='barh')
plt.title("Verification_status and Loan Status")
plt.show()
```



Description

1. Full Paid loan are approx same for All category of Verification status
2. Verified loand have high charged off loan as compared to Not verified loans
3. As the Verification is not relavent company should not spend time, effort and money or verification

In [38]:
```
ltap.info()
```

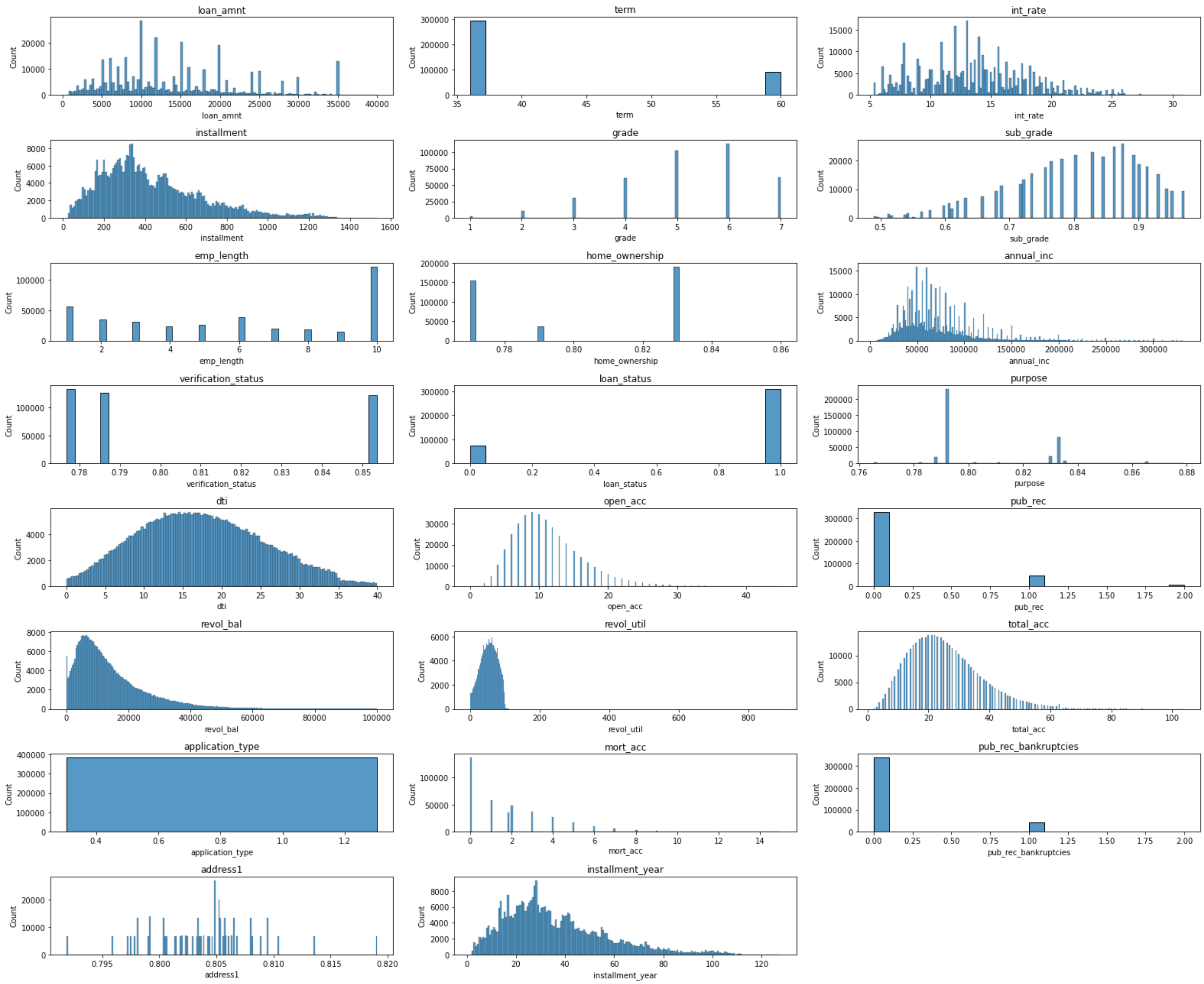
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 382966 entries, 0 to 396029
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
```

```
 ---  ------                --------------  -----
 0    loan_amnt             382966 non-null  float64
 1    term                  382966 non-null  int32
 2    int_rate              382966 non-null  float64
 3    installment           382966 non-null  float64
 4    grade                 382966 non-null  int64
 5    sub_grade             382966 non-null  float64
 6    emp_length            382966 non-null  float64
 7    home_ownership        382966 non-null  float64
 8    annual_inc            382966 non-null  float64
 9    verification_status   382966 non-null  float64
 10   loan_status           382966 non-null  int64
 11   purpose               382966 non-null  float64
 12   dti                   382966 non-null  float64
 13   open_acc              382966 non-null  float64
 14   pub_rec               382966 non-null  float64
 15   revol_bal             382966 non-null  float64
 16   revol_util            382966 non-null  float64
 17   total_acc             382966 non-null  float64
 18   application_type      382966 non-null  float64
 19   mort_acc              382966 non-null  float64
 20   pub_rec_bankruptcies  382966 non-null  float64
 21   address1              382966 non-null  float64
 22   installment_year      382966 non-null  float64
dtypes: float64(20), int32(1), int64(2)
memory usage: 76.7 MB
```

Drawing Stats using Histogram from Various Columns

In [39]:

```
L1=ltap.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(22,18))
i=len(L1)//3
for i in range(len(L1)):
    plt.subplot(8,3,i+1)
    sns.histplot(ltap[L1[i]])
    plt.title(L1[i])

plt.tight_layout()
plt.show()
```



In [40]:

```
ltap.describe()
```

Out[40]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_length
count	382966.000000	382966.000000	382966.000000	382966.000000	382966.000000	382966.000000	382966.000000
mean	14000.898317	41.679982	13.600915	428.150003	5.191257	0.804954	6.022490
std	8267.238501	10.200865	4.446034	247.326547	1.324420	0.104077	3.434731
min	500.000000	36.000000	5.320000	16.080000	1.000000	0.489130	1.000000
25%	7900.000000	36.000000	10.370000	249.540000	4.000000	0.736197	3.000000
50%	12000.000000	36.000000	13.330000	373.560000	5.000000	0.826304	6.020000
75%	19600.000000	36.000000	16.290000	562.217500	6.000000	0.891487	10.000000
max	40000.000000	60.000000	30.990000	1533.810000	7.000000	0.971323	10.000000

Description

1. Loan Amount,term variation in data
2. Interest rate,grade,emp_length seems equally distributed
3. Address 1 has all state code

Scaling the Dataset

In [41]:

```
from sklearn.preprocessing import StandardScaler

# We can obser the dataframe is highly imbalance so we will do undersampling to make the result as
print(ltap['loan_status'].value_counts(normalize=True))

## Value of minority class i.e 0 taken
df1=ltap[ltap['loan_status']==1].sample(74687)
df2=ltap[ltap['loan_status']==0]
result = df2.append(df1)

scale= StandardScaler()
scaled_data = scale.fit_transform(result)
scaled_data=pd.DataFrame(data=scaled_data,columns=ltap.columns)


X=scaled_data.drop(['loan_status','installment'],axis=1)
y=scaled_data['loan_status']


## Here we had to standardis the data the dataframe has different scale for loan amoubnt term and
```

```
1    0.804977
0    0.195023
Name: loan_status, dtype: float64
```

Train Test Split and Model Building

In [42]:

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, random_state=30)
print(X_train.shape,y_train.shape,y_train.shape,y_test.shape)
```

```
(100080, 21) (100080,) (100080,) (49294,)
```

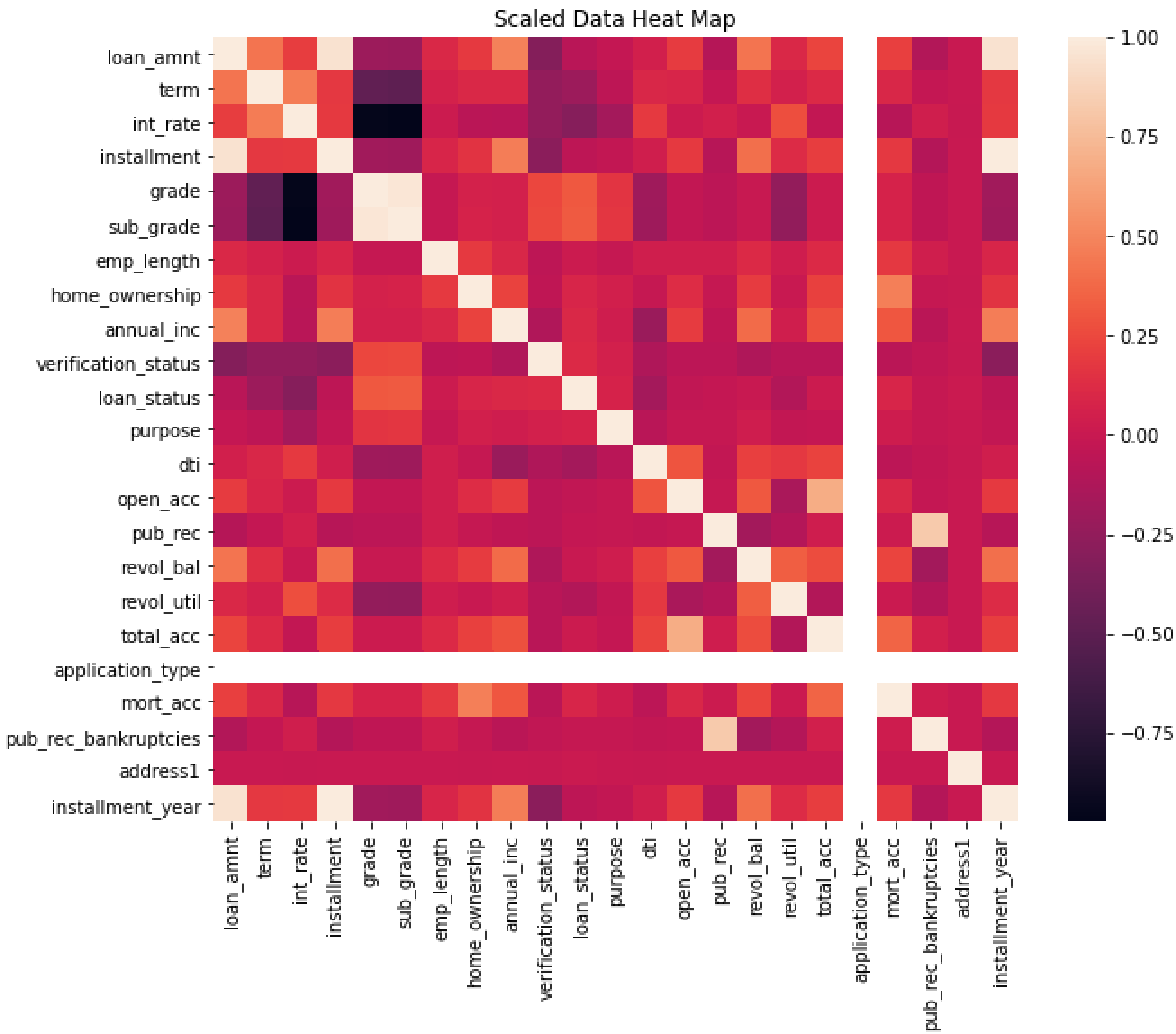
In [43]:

```
model1 = LogisticRegression()
#model1 = LogisticRegression()
model1.fit(X_train,y_train)
y_pred = model1.predict(X_test)
print(f1_score(y_test, y_pred).round(2))
print(confusion_matrix(y_pred, y_test))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
0.66
[[15722  8092]
 [ 9039 16441]]
Accuracy: 0.6524729175964621
```

In [44]:

```
plt.figure(figsize=(10,8))
sns.heatmap(scaled_data.corr())
plt.title("Scaled Data Heat Map")
plt.show()
```



In [45]:

```
Equ_2=result.drop(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_leng
Equ_2.shape
X=Equ_2.drop('loan_status',axis=1)
y=Equ_2['loan_status']
```

In [46]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, random_state=30)
model2 = LogisticRegression()
model2.fit(X_train,y_train)
y_pred = model2.predict(X_test)
print(f1_score(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(confusion_matrix(y_pred, y_test))
```

```
0.33957124567568303
Accuracy: 0.5081551507282833
[[18816 18300]
 [ 5945  6233]]
```

Hyper Parameter Tunning Using Grid Search CV.

In [47]:

```
from sklearn.model_selection import GridSearchCV


scale= StandardScaler()
```

```
scaled_data = scale.fit_transform(result)
scaled_data=pd.DataFrame(data=scaled_data,columns=ltap.columns)

parameters = {
    'penalty' : ['none', 'l1', 'l2', 'elasticnet'],
    'C'       : [100, 10, 1.0, 0.1, 0.01],
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
logreg = LogisticRegression()
clf = GridSearchCV(logreg,                    # model
                   param_grid = parameters,   # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=10)                     # number of folds
clf.fit(X_train,y_train)
print("Tuned Hyperparameters :", clf.best_params_)
print("Accuracy :",clf.best_score_)
```

```
Tuned Hyperparameters : {'C': 100, 'penalty': 'none', 'solver': 'newton-cg'}
Accuracy : 0.5262789768185452
```

New Model with Model with Hyper Parameter

In [48]:

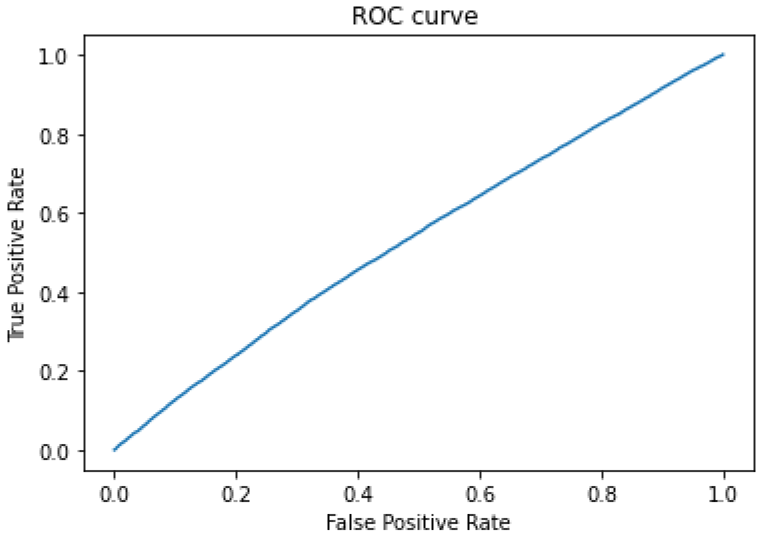
```
df1=ltap[ltap['loan_status']==1].sample(74687)
df2=ltap[ltap['loan_status']==0]
result = df2.append(df1)

scale= StandardScaler()
scaled_data = scale.fit_transform(result)
scaled_data=pd.DataFrame(data=scaled_data,columns=ltap.columns)
X=scaled_data.drop(['loan_status','installment'],axis=1)
y=scaled_data['loan_status']

model3 = LogisticRegression(C=10, penalty= 'l2', solver='newton-cg')
#model1 = LogisticRegression()
model3.fit(X_train,y_train)
y_pred = model3.predict(X_test)
print(f1_score(y_test, y_pred).round(2))
print(confusion_matrix(y_pred, y_test))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
0.56
[[10840  9571]
 [13921 14962]]
Accuracy: 0.5234308435103664
```

ROC AUC Curve

In [49]:

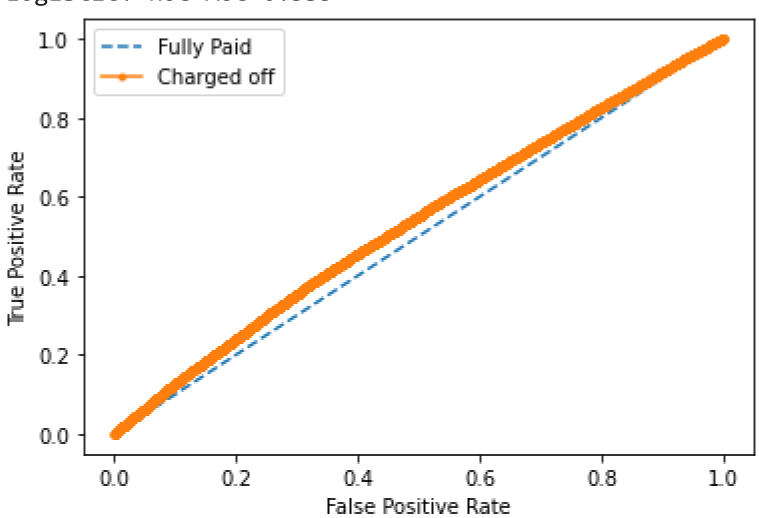
```
#define metrics
y_pred_proba = model3.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title("ROC curve ")
plt.show()
```



In [50]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = model3.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
# summarize scores
print('Fully Paid: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='Fully Paid')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Charged off')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
plt.show()
# show the plot
```

```
Fully Paid: ROC AUC=0.500
Logistic: ROC AUC=0.535
```



Final Observations and Recommendations

1. Model is performing better with More parameters .
2. Grid Search parameter and Logistic model does not have any major difference in Accuracy
3. There are Large no of fully paid loans evident in ROC curve
4. Confusion matix shows the high percent of False positive people who are terms fully paid whereas they are not
5. All False negative where model is predicting people are people are paying of where actually they are not .
6. NPA is Big problem and there are number of parameters verification which can be ignored, Annual Income , Installement have direct correlation .
7. Checks on parameters which are relavent in EDA mentioned above and remove the checks for unwanted one will help in reducing cost and improving profitability
8. Model can do better with More data points on the problem we used the undersampling method to train the model .
9. CHi square is not giving the accurate result in terms of variable which can be removed this needs to be used in conjunction with correlation matrix .
10. There are larege outliers in dataset which is happering the performance .
11. Outliers in Installement years going beyond 50 Years looks to be data entry / input issue this needs to be corrected.
12. Smote can also be testing for oversampling of samples instead on undersampling .

In [ ]:

In [ ]:

In [ ]:

In [ ]: