Lending Club Loan - Predicting Charged-Off

LendingClub-Loan-Analysis Lending loans to ‘risky’ applicants is the largest source of financial loss(called credit loss) for any bank/lending company. If we are able to identify these risky loan applicants, then such loans can be reduced thereby cutting down the amount of credit loss. Identification of such applicants using Data Analysis is the aim of this case study. Lending Club (a peer-to-peer lending company) wants to understand the driving factors behind loan default. The company can utilise this knowledge for its portfolio and risk assessment. 2 types of risks are associated with the bank’s decision: 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 companyData Used : The data used was acquired from Kaggle, open-sourced by LendingClub itself to welcome Data Scientist help them identify driving factors behind loan default, using historic data of loan applications. Be sure to checkout the Data Dictionary for the meaning of each column in the dataset. The data given contains the information about past loan applicants and whether they ‘defaulted’ or not. The aim is to identify patterns which indicate if a person is likely to default, which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc. When a person applies for a loan, there are 2 types of decisions that could be taken by the company: Loan accepted - If the company approves the loan, there are 3 possible scenarios described below: Fully paid: Applicant has fully paid the loan (the principal and the interest rate) Current: Applicant is in the process of paying the installments, i.e. the tenure of the loan is not yet completed. These candidates are not labelled as 'defaulted'. Charged-off: Applicant has not paid the instalments in due time for a long period of time, i.e. he/she has defaulted on the loan. Loan rejected - The company had rejected the loan (because the candidate does not meet their requirements etc.). Since the loan was rejected, there is no transactional history of those applicants with the company and so this data is not available with the company (and thus in this dataset)

Overview of the Notebook:

Loading and inspecting the Dataset:

- Checking Shape of the Dateset - Meaningful Column names - Validating Duplicate Record - Checking Missing values - Unique values (counts & names) for each Feature - Data & Datatype validation

Target variable Analysis:

- Checking Imbalance

EDA & Pre-processing:

- Numerical variable - Categorical variable

Feature Engineering:

- Deriving New Features.

Model Building:

- Correlation Analysis - Handling Categorical variables using dummies - Train, Cross validation & Test Split - Imputation - HAndling missing values - Rescaling features - Pipeline creation - Train Model using Logistic Regression: -> Basic Model -> Advanced Model using Hyper Parmater optimization -> Advanced Model using Hyper Parmater optimization & class weights

Model Performance Evaluation:

- AUC ROC curve - Recall vs Precision - F1 score - Optimal cut-off using Precision-Recall Trade off - Comparision between Modes on performance measures.

Business Insights:

Let's Start

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid')
import warnings
warnings.filterwarnings('ignore')
from scipy import stats
from scipy.stats import kstest
import statsmodels.api as sm
### Importing Date & Time util modules
from dateutil.parser import parse
```

In [2]:

```
df = pd.read_csv(r"C:\Users\rohan\Downloads\lending_club_loan_two.csv", index_col=F
df.head()
```

Out[2]:

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

5 rows × 27 columns

Missing Value - Calculator

In [3]:
```
def missingValue(df):
    total_null = df.isnull().sum().sort_values(ascending = False)
    percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascending =
    print("Total records = ", df.shape[0])

    md = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In P
    return md
```

Numerical Variable Analysis:

- box plot - distplot

In [4]:
```
def plot_num_var(df,colname,name):
    # Visualizing our dependent variable and Skewness
    fig , (ax1,ax2) = plt.subplots(1,2,figsize=(15,5))
    fig.set_facecolor("lightgrey")

    sns.boxplot(y= colname,x='loan_status',data=df,ax=ax1)
    ax1.set_ylabel(name, fontsize=14,family = "Comic Sans MS")
    ax1.set_xlabel('Count', fontsize=14,family = "Comic Sans MS")
    ax1.set_title(name + ' by Loan Status', fontweight="bold",fontsize=15,family =

    sns.distplot(df[colname],color='y',ax=ax2,kde=True)

    mean = df[colname].mean()
    median = df[colname].median()
    mode = df[colname].mode()[0]

    label_mean= ("Mean :  {:.2f}".format(mean))
    label_median = ("Median :  {:.2f}".format(median))
    label_mode = ("Mode :  {:.2f}".format(mode))

    ax2.set_title("Distribution of " + name, fontweight="bold",fontsize=15,family =
    ax2.set_ylabel('Density', fontsize=12,family = "Comic Sans MS")
    ax2.set_xlabel(name, fontsize=12,family = "Comic Sans MS")
    ax2.axvline(mean,color="g",label=label_mean)
    ax2.axvline(median,color="b",label=label_median)
    ax2.axvline(mode,color="r",label=label_mode)
    ax2.legend()
    plt.show()
```

Categorical variables:

- Count plot - Stack bar plot

In [5]:
```
# Frequency of each feature in percentage.
def count_plt(df, colname, name,width=14,height=14,rotation=0):
    fig = plt.figure(figsize=(width, height))
    fig.set_facecolor("lightgrey")
    string = "Frequency of " + name
    ax = sns.countplot(df[colname], order=sorted(df[colname].unique()), color='#56B

    plt.xticks(rotation = rotation,fontsize=16,family="Comic Sans MS")
    plt.yticks(fontsize=16,family="Comic Sans MS")
    plt.ylabel(string, fontsize=18,family = "Comic Sans MS")
    plt.xlabel(name, fontsize=18,family = "Comic Sans MS")
    for p in ax.patches:
        ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.1, p.get_height()
```

In [6]:

```
def stack_bar(df,colname,name):
    cross_tab_pct = pd.crosstab(index=df[colname],
                                columns=df['loan_status'],normalize="index")
    cross_tab = pd.crosstab(index=df[colname],columns=df['loan_status'])

    cross_tab_pct.plot(kind='bar', stacked=True, colormap='Wistia', figsize=(10, 6)

    plt.legend(loc="upper right", ncol=2)
    plt.xlabel(name,fontsize=14,family = "Comic Sans MS")
    plt.ylabel("Loan Status",fontsize=14,family = "Comic Sans MS")
    plt.xticks(rotation=0)

    for n, x in enumerate([*cross_tab.index.values]):
        for (proportion, count, y_loc) in zip(cross_tab_pct.loc[x],
                                              cross_tab.loc[x],
                                              cross_tab_pct.loc[x].cumsum()):

            plt.text(x=n - 0.17,y=(y_loc - proportion) + (proportion / 2),
                     s=f'{count}\n({np.round(proportion * 100, 1)}%)',
                     color="black",fontsize=12,fontweight="bold")

    plt.show()
```

In [7]:

```
def stack_bar_h(df,colname,name):
    cross_tab_pct = pd.crosstab(index=df[colname],
                                columns=df['loan_status'],normalize="index")
    cross_tab = pd.crosstab(index=df[colname],columns=df['loan_status'])

    cross_tab_pct.plot(kind='barh',stacked=True, colormap='Wistia', figsize=(10, 18

    plt.legend(loc="lower right", ncol=2)
    plt.xlabel(name,fontsize=14,family = "Comic Sans MS")
    plt.ylabel("Loan Status",fontsize=14,family = "Comic Sans MS")
    plt.xticks(rotation=0)

    for n, x in enumerate([*cross_tab.index.values]):
        for (proportion, count, y_loc) in zip(cross_tab_pct.loc[x],cross_tab.loc[x]
                                              cross_tab_pct.loc[x].cumsum()):

            plt.text(x=(y_loc - proportion) + (proportion / 2),y=n - 0.11,
                     s=f'{count}\n({np.round(proportion * 100, 1)}%)',
                     color="black", fontsize=10,)

    plt.show()
```

In [8]:

```
loan_data = pd.read_csv(r"C:\Users\rohan\Downloads\lending_club_loan_two.csv")
loan_data.head()
```

Out[8]:

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

5 rows × 27 columns

Validating Duplicate Records

In [9]:
```
loan_data.shape
```
Out[9]:
```
(396030, 27)
```

In [10]:
```
loan_data.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')
```

Validating Duplicate Records

In [11]:
```
loan_data.duplicated().sum()
```
Out[11]:
```
0
```

In [12]:
```
missingValue(loan_data).head(7)
```
```
Total records =  396030
```

Out[12]:

	Total Missing	In Percent
mort_acc	37795	9.54
emp_title	22927	5.79
emp_length	18301	4.62
title	1755	0.44
pub_rec_bankruptcies	535	0.14
revol_util	276	0.07
loan_amnt	0	0.00

Inferences

- There are missing values. We will handled same during EDA and Pre-Processing the data

In [13]:
```
loan_data['loan_status'].unique()
```

Out[13]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [14]:
```
loan_data.info()
```

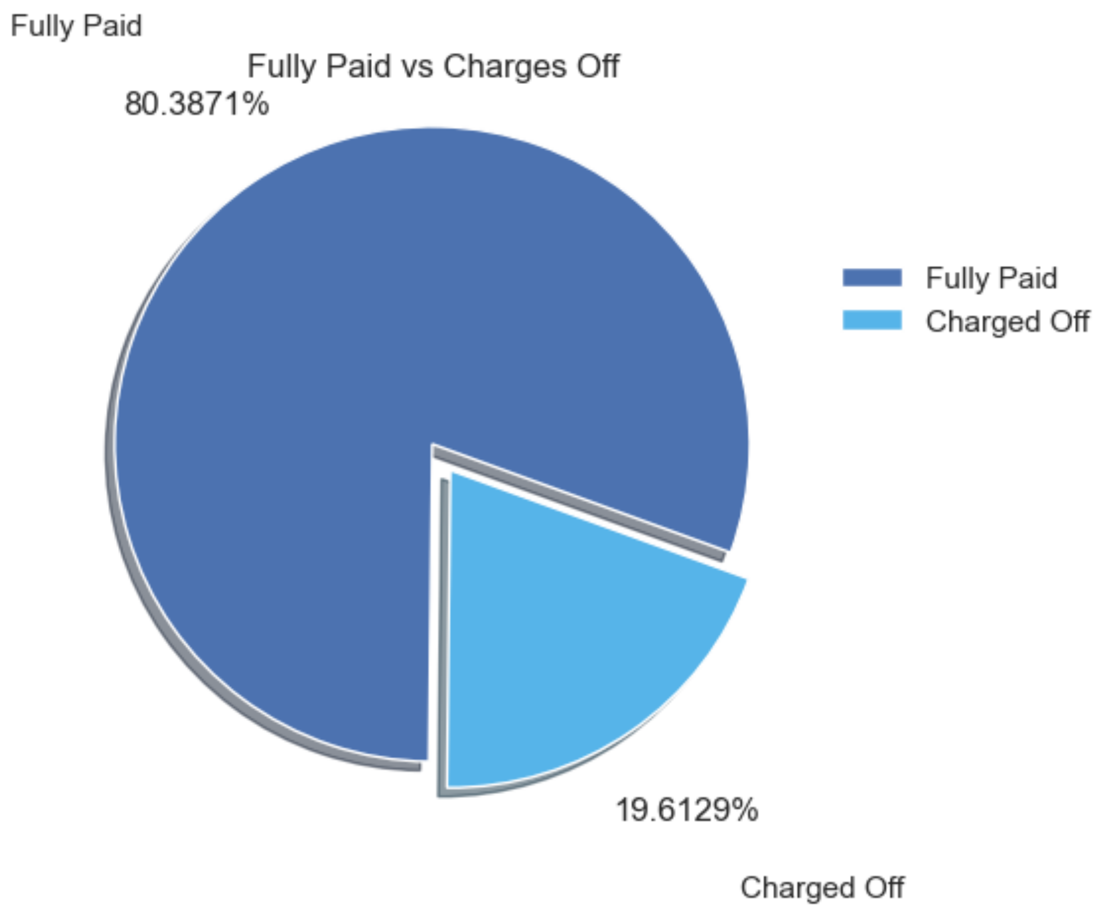
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

Target variable Analysis

In [15]:

```
fig, ax = plt.subplots()

labels = ['Fully Paid','Charged Off']
explode=(0.1,0)

loan_status = loan_data["loan_status"].value_counts()

df = pd.DataFrame({'labels': loan_status.index,
                   'values': loan_status.values
                  })
ax.pie(loan_status.values, explode=explode, labels=labels,
       colors=['b','#56B4E9'], autopct='%1.4f%%',
       shadow=True, startangle=-20,
       pctdistance=1.3,labeldistance=1.6)

ax.axis('equal')
ax.set_title("Fully Paid vs Charges Off")
ax.legend(frameon=False, bbox_to_anchor=(1.2,0.8))
```

Out[15]: <matplotlib.legend.Legend at 0x1f9c2adedf0>



Inference

- There are approximately 80.5% of unpaid loans, while 19% have been charged off, resulting in an imbalance in classification.

Pre-Processing & EDA

Numerical Variables

loan_amnt

In [16]:
```
loan_data[['loan_amnt']].describe().T
```

Out[16]:

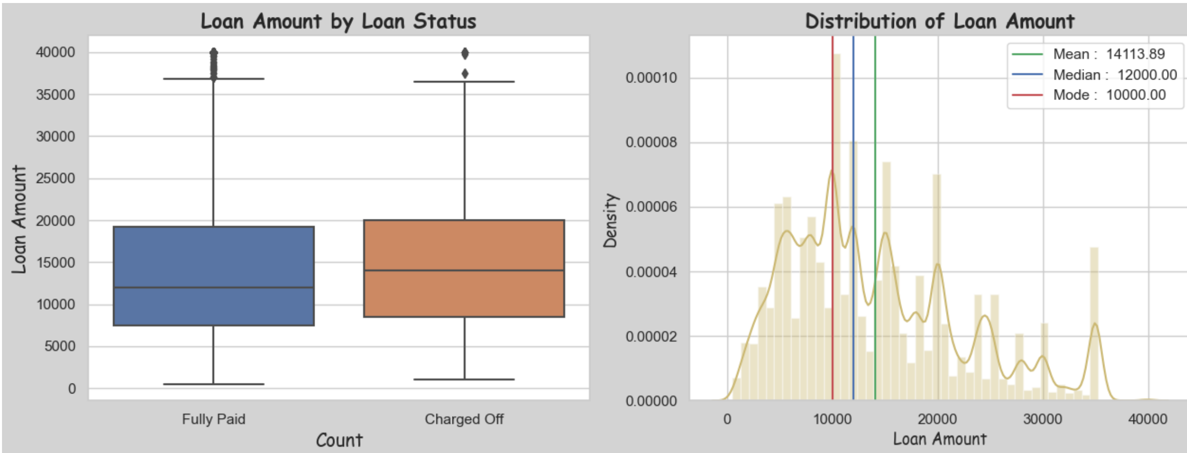
	count	mean	std	min	25%	50%	75%	max
loan_amnt	396030.0	14113.888089	8357.441341	500.0	8000.0	12000.0	20000.0	40000.0

In [17]:
```
loan_data.groupby(['loan_status'])['loan_amnt'].describe()
```

Out[17]:

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

In [18]:
```
plot_num_var(loan_data,'loan_amnt','Loan Amount')
```



Inference

- Median Loan Amount is 141 Charged-offs have a higher loan amount than fully paid with a mean loan amount of 13866 & 15126, respectively.

Interest Rate

In [19]:
```
loan_data[['int_rate']].describe().T
```

Out[19]:

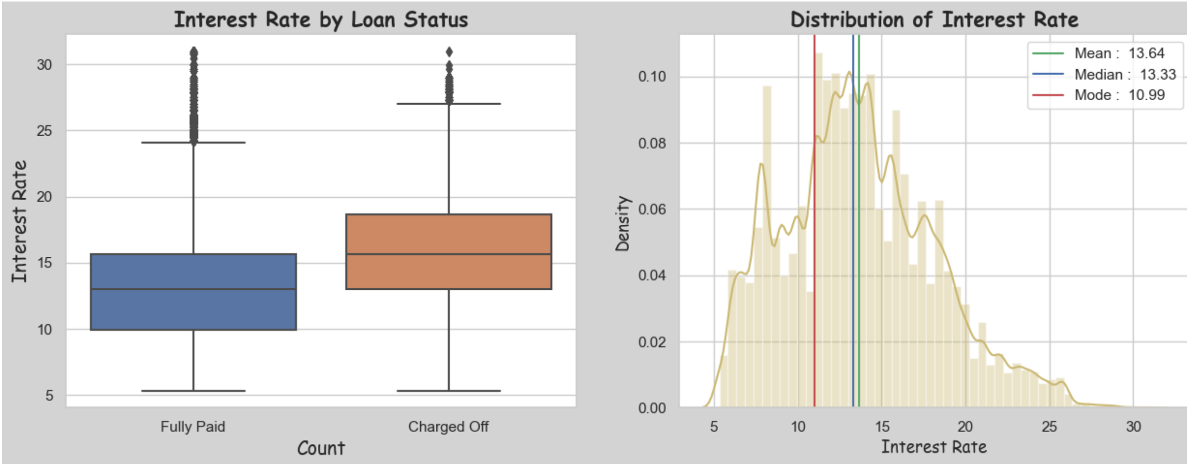
	count	mean	std	min	25%	50%	75%	max
int_rate	396030.0	13.6394	4.472157	5.32	10.49	13.33	16.49	30.99

In [20]:
```
loan_data.groupby(['loan_status'])['int_rate'].describe()
```

Out[20]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15.882587	4.388135	5.32	12.99	15.61	18.64	30.99
Fully Paid	318357.0	13.092105	4.319105	5.32	9.91	12.99	15.61	30.99

In [21]:
```
plot_num_var(loan_data,'int_rate','Interest Rate')
```



Inference

- Median interest rate of 13%, Interest rates range from 5.32% to 30.99%. Charged-offs have a higher interest rate than fully paid with a mean interest rate of 15.88% & 13.09%, respectively.

Installment

In [22]:
```
loan_data[['installment']].describe().T
```

Out[22]:

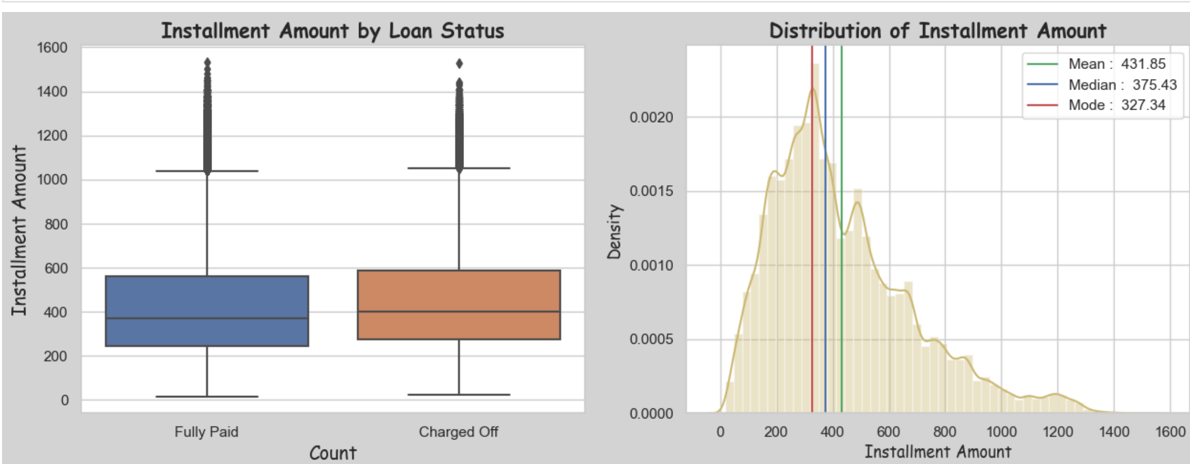
	count	mean	std	min	25%	50%	75%	max
installment	396030.0	431.849698	250.72779	16.08	250.33	375.43	567.3	1533.81

In [23]:
```
loan_data.groupby(['loan_status'])['installment'].describe()
```

Out[23]:

	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	77673.0	452.703110	249.096609	21.62	274.86	399.06	585.67	1527.00
Fully Paid	318357.0	426.761866	250.861622	16.08	244.46	369.51	562.89	1533.81

In [24]:
```
plot_num_var(loan_data,'installment','Installment Amount')
```



Inference

- Charged-offs have a slighty higher installemnt amount than Fully paid. - The mean and median installation amounts for charge-off are 452 and 399 respectively. - The mean and median installation amounts for Fully Paid are 426 and 369 respectively.

Annual Income

In [25]:
```
loan_data[['annual_inc']].describe().T
```

Out[25]:

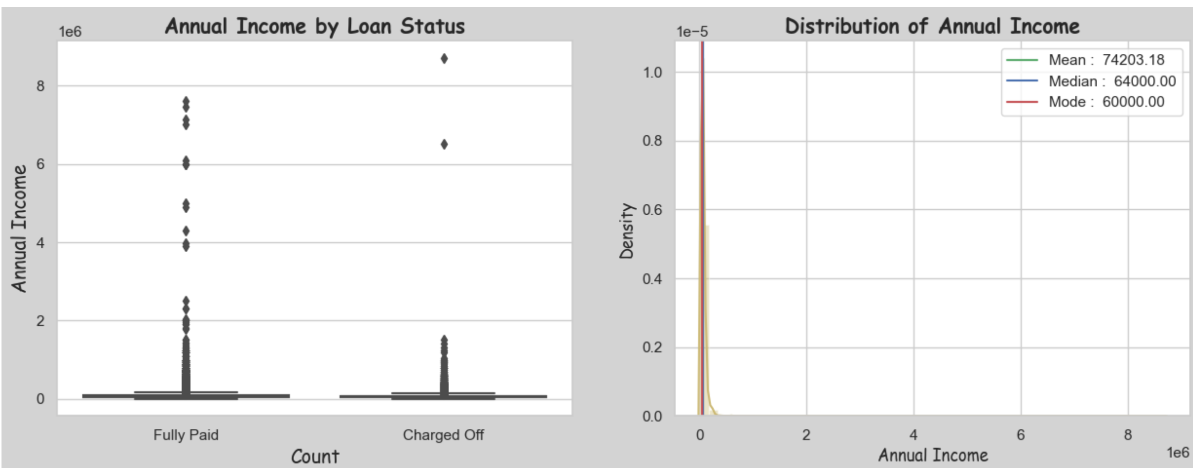
	count	mean	std	min	25%	50%	75%	max
annual_inc	396030.0	74203.175798	61637.621158	0.0	45000.0	64000.0	90000.0	8706582.0

In [26]:
```
loan_data.groupby(['loan_status'])['annual_inc'].describe()
```

Out[26]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	67535.537710	58303.457136	0.0	42000.00	59000.0	80000.0	8706582.0
Fully Paid	318357.0	75829.951566	62315.991907	600.0	46050.53	65000.0	90000.0	7600000.0

In [27]:
```
plot_num_var(loan_data,'annual_inc','Annual Income')
```



Inferences

- Based on the above graph and table, the annual income range is very wide. We should perform some transformations, like log,to get a better picture.

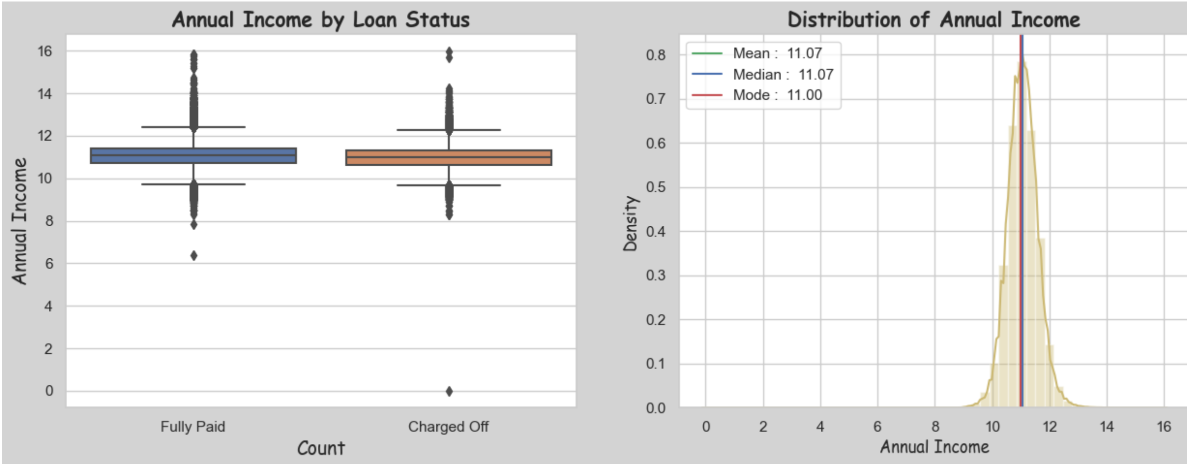
In [28]:
```
## trainsforming target variable using numpy.log1p,
loan_data["annual_inc_ln"] = np.log1p(loan_data["annual_inc"])
```

In [29]:
```
loan_data[['annual_inc_ln']].describe().T
```

Out[29]:

	count	mean	std	min	25%	50%	75%	max
annual_inc_ln	396030.0	11.067137	0.5246	0.0	10.71444	11.066654	11.407576	15.97959

In [30]:
```
plot_num_var(loan_data,'annual_inc_ln','Annual Income')
```



In [31]:
```
loan_data.groupby(['loan_status'])['annual_inc_ln'].describe()
```

Out[31]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	10.977794	0.517411	0.000000	10.645449	10.985310	11.289794	15.979590
Fully Paid	318357.0	11.088935	0.524034	6.398595	10.737516	11.082158	11.407576	15.843659

In [32]:
```
77673/loan_data.shape[0]
```

Out[32]:
```
0.1961290811302174
```

In [33]:
```
318357/loan_data.shape[0]
```

Out[33]:
```
0.8038709188697826
```

Inference:

- In terms of individual annual income, the distribution of charged off loans is similar to that of fully paid loans, - except individual with salary 0. - Logistic Regression models are not much impacted due to the presence of outliers because the sigmoid function tapers the outliers. But the presence of extreme outliers may somehow affect the performance of the model and lowering the performance.

```
Note - To improve the performance of the model we will be removing
the outliers using the repetitive process of
```

training model and detecting and removing outliers

In [34]:
```
loan_data.drop('annual_inc_ln', axis=1, inplace=True)
```

DTI:

- A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested Lending club loan, divided by the borrower's self-reported monthly income.

In [35]:
```
loan_data[['dti']].describe().T
```

Out[35]:

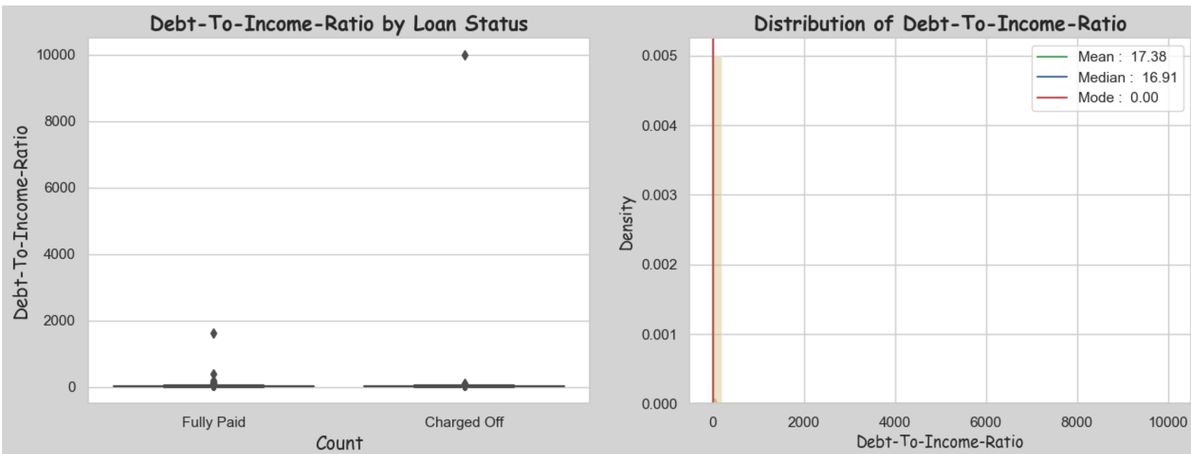
	count	mean	std	min	25%	50%	75%	max
dti	396030.0	17.379514	18.019092	0.0	11.28	16.91	22.98	9999.0

In [36]:
```
loan_data.groupby(['loan_status'])['dti'].describe()
```

Out[36]:

	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	77673.0	19.656346	36.781068	0.0	13.33	19.34	25.55	9999.0
Fully Paid	318357.0	16.824010	8.500979	0.0	10.87	16.34	22.29	1622.0

In [37]:
```
plot_num_var(loan_data,'dti','Debt-To-Income-Ratio')
```



In [38]:
```
loan_data.loc[loan_data['dti']>=50, 'loan_status'].value_counts()
```

Out[38]:
```
Fully Paid     26
Charged Off     9
Name: loan_status, dtype: int64
```

In [39]:
```
9/35
```

Out[39]:
```
0.2571428571428571
```

In [40]:
```
loan_data.loc[loan_data['dti']<=10, 'loan_status'].value_counts()
```

Out[40]:
```
Fully Paid     68242
Charged Off    10850
Name: loan_status, dtype: int64
```

In [41]:
```
10850/(68242+10850)
```

Out[41]:
```
0.13718201588024073
```

Inferences:

- The likelihood of a loan getting charged-off increases as DTI values increases

Open Credit Lines

- The number of open credit lines in the borrower's credit file.

In [42]:
```
loan_data[['open_acc']].describe().T
```

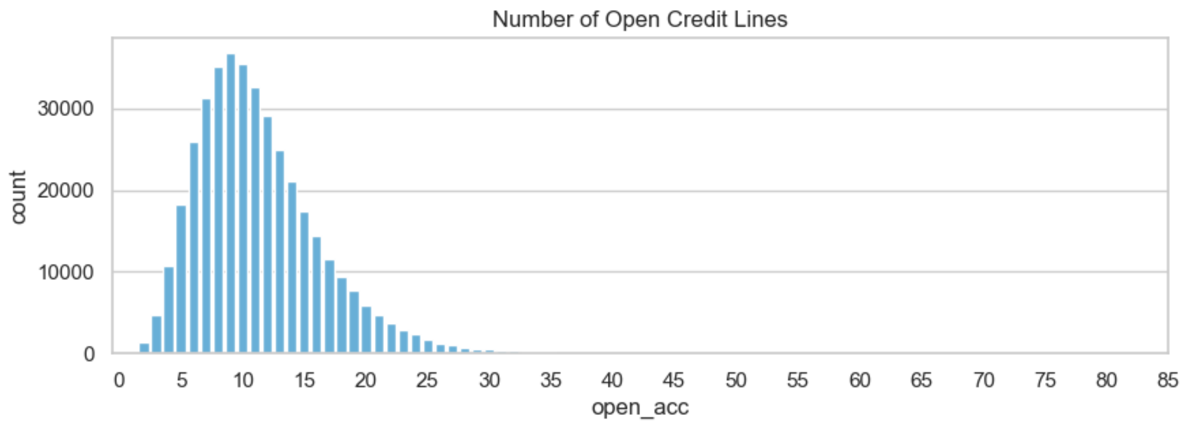
Out[42]:

	count	mean	std	min	25%	50%	75%	max
open_acc	396030.0	11.311153	5.137649	0.0	8.0	10.0	14.0	90.0

In [43]:
```
loan_data['open_acc'].nunique()
```

Out[43]:
```
61
```

In [44]:
```
plt.figure(figsize=(10,3),dpi=100)
fig.set_facecolor("lightgrey")
sns.countplot(loan_data['open_acc'], order=sorted(loan_data['open_acc'].unique()),
a, b = plt.xticks(np.arange(0, 90, 5), np.arange(0, 90, 5))
plt.title('Number of Open Credit Lines')
plt.show()
```



Public Records(pub_rec)

- Number of derogatory public records

In [45]:
```
loan_data[['pub_rec']].describe().T
```

Out[45]:

	count	mean	std	min	25%	50%	75%	max
pub_rec	396030.0	0.178191	0.530671	0.0	0.0	0.0	0.0	86.0

In [46]:
```
loan_data['pub_rec'].value_counts().head(7)
```

Out[46]:
```
0.0    338272
1.0     49739
2.0      5476
3.0      1521
4.0       527
5.0       237
6.0       122
Name: pub_rec, dtype: int64
```

In [47]:
```
loan_data.loc[loan_data['pub_rec']>=1, 'loan_status'].value_counts()
```

Out[47]:
```
Fully Paid     45424
Charged Off    12334
Name: loan_status, dtype: int64
```

In [48]:
```
12334/(12334+45424)
```

Out[48]:
```
0.21354617542158663
```

In [49]:
```
loan_data.loc[loan_data['pub_rec']>2, 'loan_status'].value_counts()
```

Out[49]:
```
Fully Paid     1932
Charged Off     611
Name: loan_status, dtype: int64
```

In [50]:
```
611/(611+1932)
```

Out[50]:
```
0.2402674007078254
```

Inferences:

- As we can see that for derogatory public record have high probability of loan getting charged-off

Revolving Balance:

- Total credit revolving balance

In [51]:
```
loan_data['revol_bal'].nunique()
```

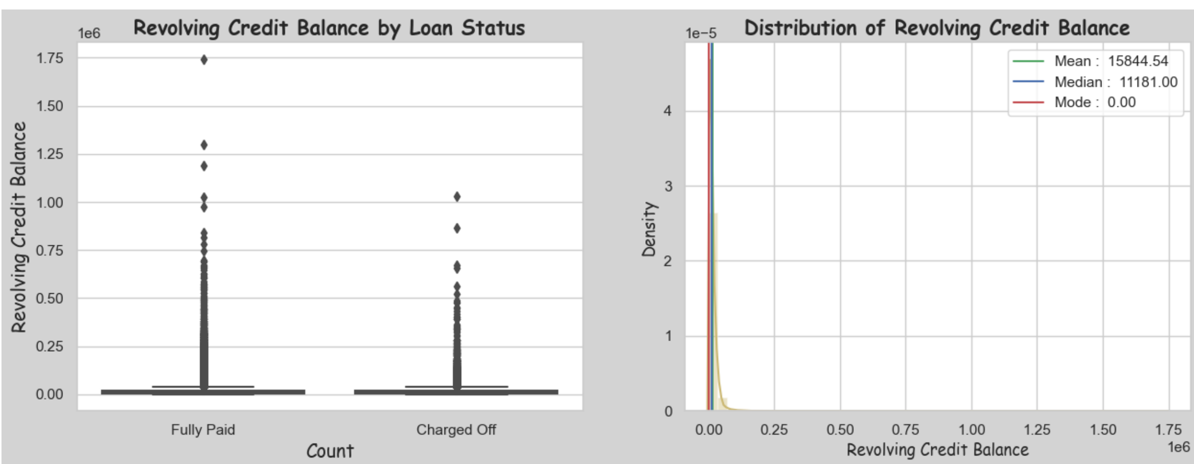
Out[51]:
```
55622
```

In [52]:
```
loan_data[['revol_bal']].describe().T
```

Out[52]:

	count	mean	std	min	25%	50%	75%	max
revol_bal	396030.0	15844.539853	20591.836109	0.0	6025.0	11181.0	19620.0	1743266.0

In [53]:
```
plot_num_var(loan_data,'revol_bal','Revolving Credit Balance')
```

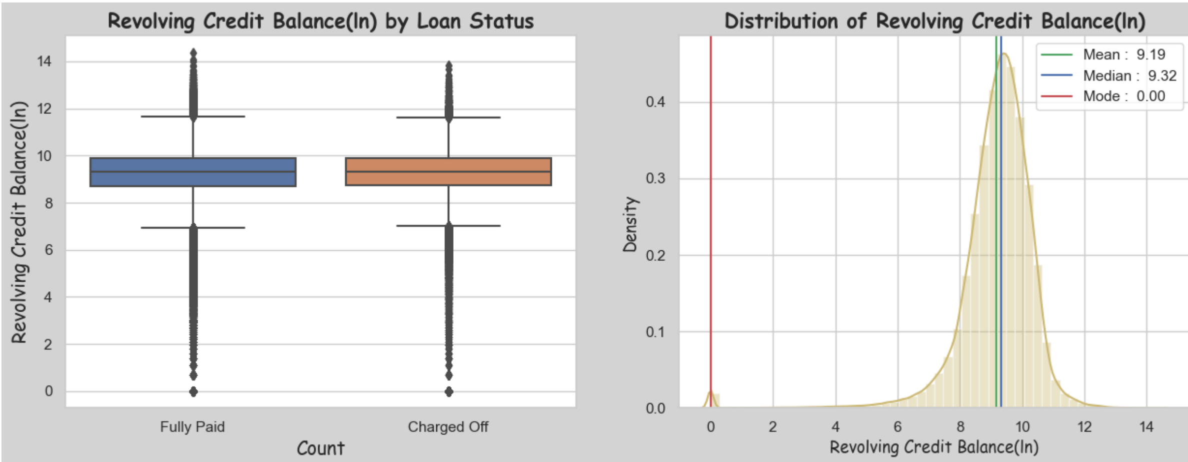


Inferences:

- Based on the above graph and table, the annual income range is very wide. We should perform some transformations, like log, to get a better picture. - We will handle the outliers later on.

In [54]:
```
## trainsforming target variable using numpy.log1p,
loan_data["revol_bal_ln"] = np.log1p(loan_data["revol_bal"])
```

In [55]:
```
plot_num_var(loan_data,'revol_bal_ln','Revolving Credit Balance(ln)')
```



In [56]:
```
loan_data.groupby(['loan_status'])['revol_bal'].describe()
```

Out[56]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15390.454701	18203.387930	0.0	6150.0	11277.0	19485.0	1030826.0
Fully Paid	318357.0	15955.327918	21132.193457	0.0	5992.0	11158.0	19657.0	1743266.0

In [57]:
```
loan_data.drop('revol_bal_ln', axis=1, inplace=True)
```

revol_util

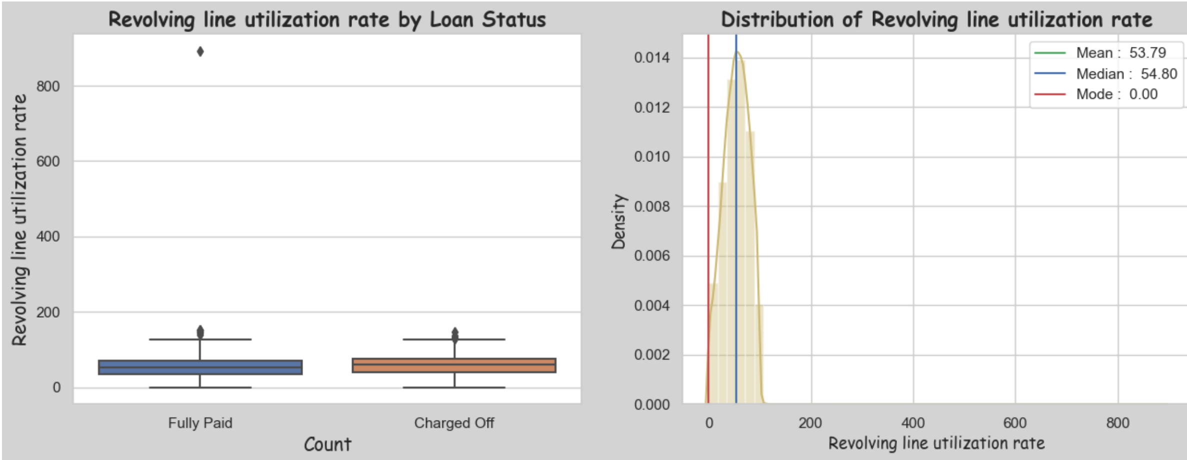
- Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.

In [58]:
```
loan_data[['revol_util']].describe().T
```

Out[58]:

	count	mean	std	min	25%	50%	75%	max
revol_util	395754.0	53.791749	24.452193	0.0	35.8	54.8	72.9	892.3

In [59]:
```
plot_num_var(loan_data,'revol_util','Revolving line utilization rate')
```



Inferences:

- Some outliers observered. We will remove later.

total_acc

- The total number of credit lines currently in the borrower's credit file.

In [60]:
```
loan_data[['total_acc']].describe().T
```

Out[60]:

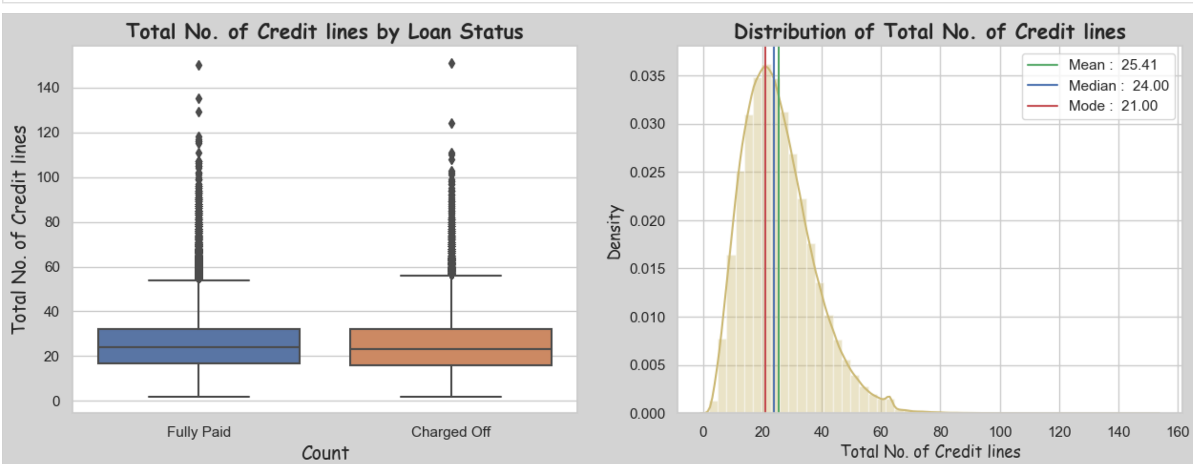
	count	mean	std	min	25%	50%	75%	max
total_acc	396030.0	25.414744	11.886991	2.0	17.0	24.0	32.0	151.0

In [61]:
```
loan_data.groupby(['loan_status'])['total_acc'].describe()
```

Out[61]:

	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	77673.0	24.984152	11.913692	2.0	16.0	23.0	32.0	151.0
Fully Paid	318357.0	25.519800	11.878117	2.0	17.0	24.0	32.0	150.0

In [62]:
```
plot_num_var(loan_data,'total_acc','Total No. of Credit lines')
```



Inferences:

- Mean difference between Charged-off and Fully paid for total number of credit lines are not much.

mort_acc

- Number of mortgage accounts.

In [63]:
```
loan_data[['mort_acc']].describe().T
```

Out[63]:

	count	mean	std	min	25%	50%	75%	max
mort_acc	358235.0	1.813991	2.14793	0.0	0.0	1.0	3.0	34.0

In [64]:
```
loan_data.groupby(['loan_status'])['mort_acc'].describe()
```

Out[64]:

	count	mean	std	min	25%	50%	75%	max
loan_status								
Charged Off	72123.0	1.501213	1.974353	0.0	0.0	1.0	2.0	23.0
Fully Paid	286112.0	1.892836	2.182456	0.0	0.0	1.0	3.0	34.0

In [65]:
```
loan_data['mort_acc'].value_counts().head(10)
```

Out[65]:
```
0.0    139777
1.0     60416
2.0     49948
3.0     38049
4.0     27887
5.0     18194
6.0     11069
7.0      6052
8.0      3121
9.0      1656
Name: mort_acc, dtype: int64
```

In [66]:
```
loan_data.loc[loan_data['mort_acc']>=10, 'loan_status'].value_counts()
```

Out[66]:
```
Fully Paid     1797
Charged Off     269
Name: loan_status, dtype: int64
```

In [67]:
```
269/(1797+269)
```

Out[67]:
```
0.13020329138431752
```

Inferences:

- According to the above analysis, people with 0 Mortgage accounts have a high risk of defaulting on their loans.

pub_rec_bankruptcies

- Number of public record bankruptcies

In [68]:
```
loan_data['pub_rec_bankruptcies'].value_counts().sort_index()
```

Out[68]:
```
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: pub_rec_bankruptcies, dtype: int64
```

In [69]:
```
loan_data.loc[loan_data['pub_rec_bankruptcies']>=1, 'loan_status'].value_counts()
```

Out[69]:
```
Fully Paid     35850
Charged Off     9265
Name: loan_status, dtype: int64
```

In [70]:
```
9265/(9265+35850)
```

Out[70]:
```
0.20536406959991133
```

Inferences:

- According to the above analysis, people with 1 or more number of public record bankruptcies have a high risk of defaulting on their loans.

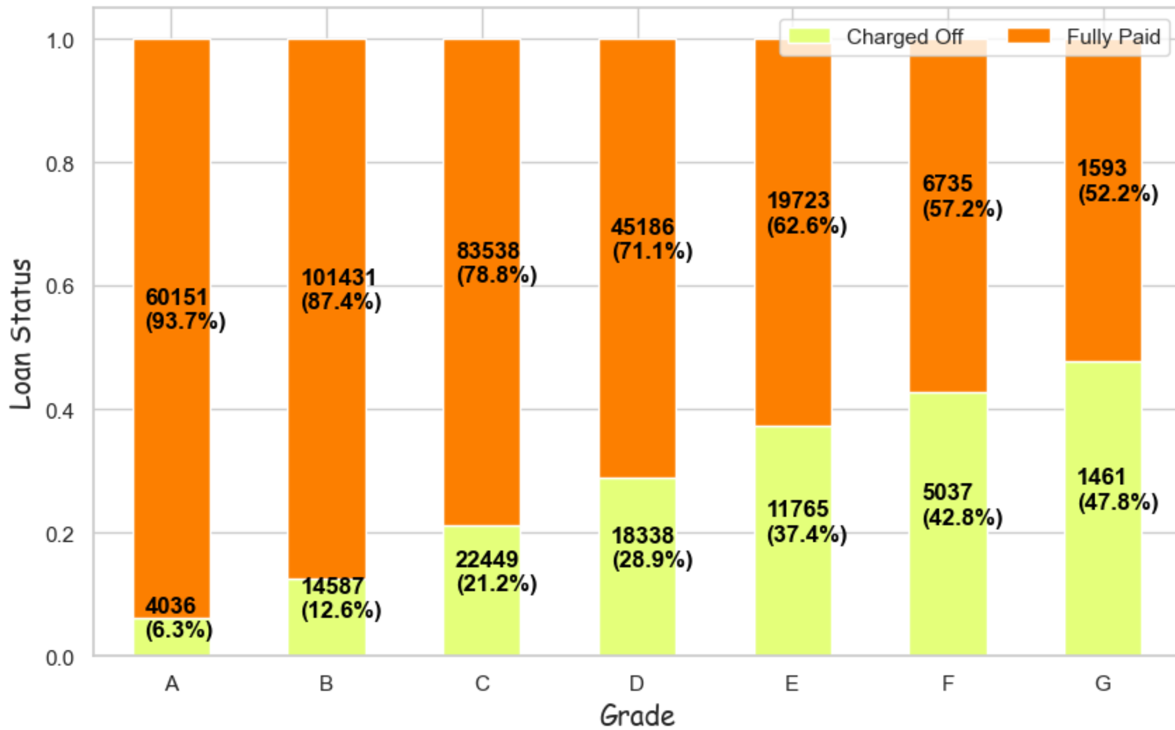
Categorical variables

Grade & Sub-grade

In [71]:
```
print(sorted(loan_data['grade'].unique()))
```

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

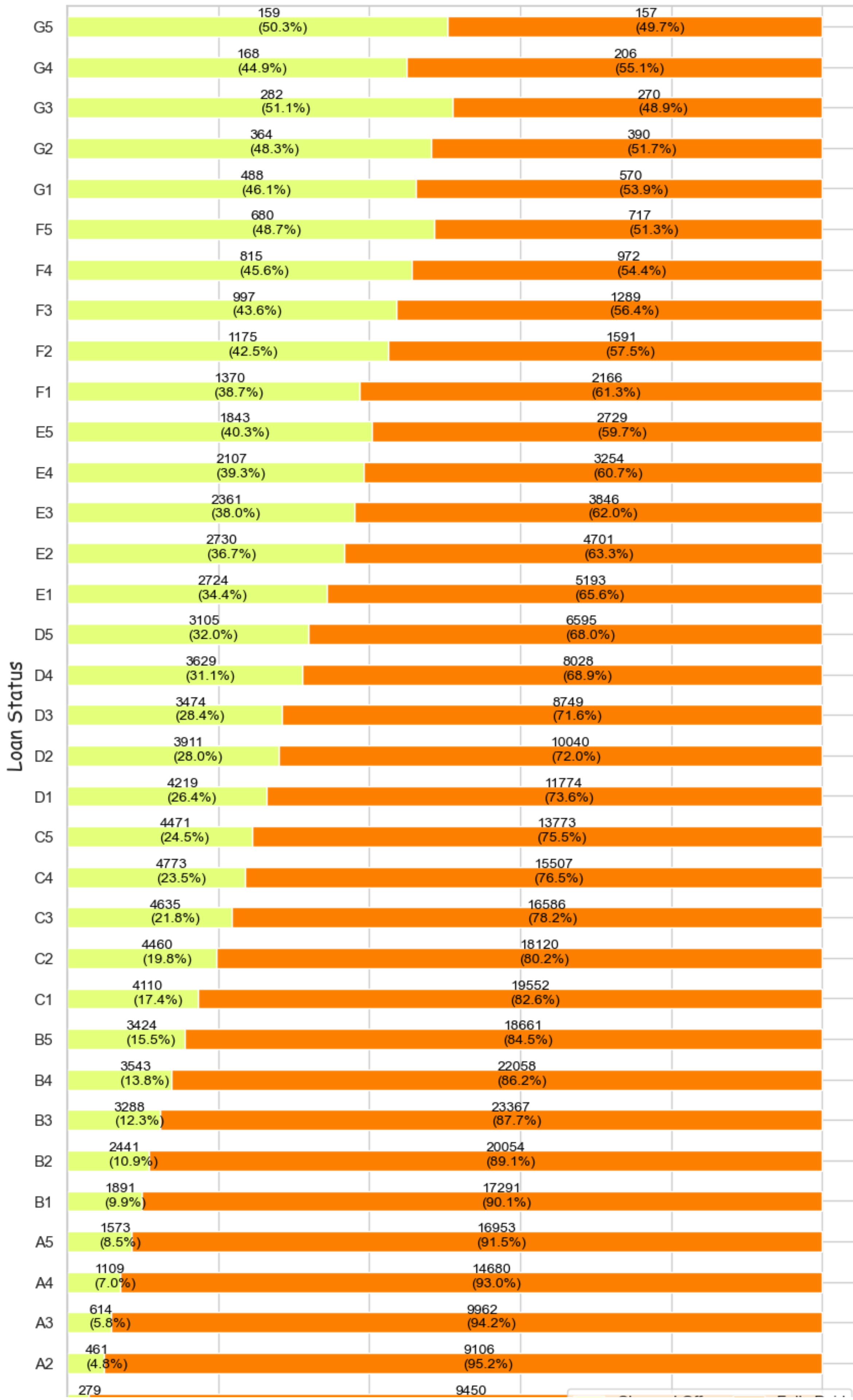
In [72]:
```
stack_bar(loan_data,'grade',"Grade")
```

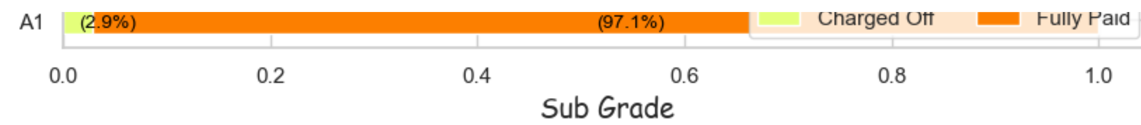


In [73]:
```
print(sorted(loan_data['sub_grade'].unique()))
```

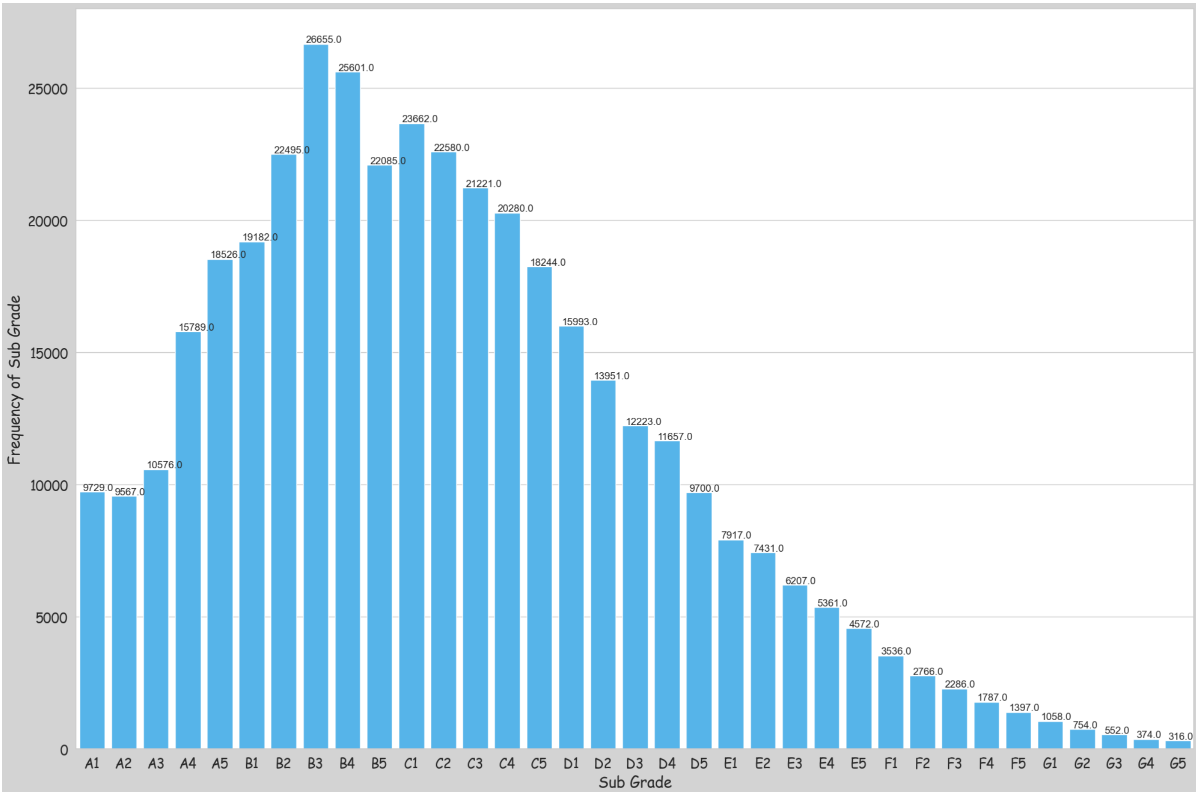
```
['A1', 'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5', 'C1', 'C2', 'C3', 'C4
', 'C5', 'D1', 'D2', 'D3', 'D4', 'D5', 'E1', 'E2', 'E3', 'E4', 'E5', 'F1', 'F2', 'F
3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5']
```

In [74]:
```
stack_bar_h(loan_data,'sub_grade',"Sub Grade")
```


Loan Status
G5 159 (50.3%) 157 (49.7%)
G4 168 (44.9%) 206 (55.1%)
G3 282 (51.1%) 270 (48.9%)
G2 364 (48.3%) 390 (51.7%)
G1 488 (46.1%) 570 (53.9%)
F5 680 (48.7%) 717 (51.3%)
F4 815 (45.6%) 972 (54.4%)
F3 997 (43.6%) 1289 (56.4%)
F2 1175 (42.5%) 1591 (57.5%)
F1 1370 (38.7%) 2166 (61.3%)
E5 1843 (40.3%) 2729 (59.7%)
E4 2107 (39.3%) 3254 (60.7%)
E3 2361 (38.0%) 3846 (62.0%)
E2 2730 (36.7%) 4701 (63.3%)
E1 2724 (34.4%) 5193 (65.6%)
D5 3105 (32.0%) 6595 (68.0%)
D4 3629 (31.1%) 8028 (68.9%)
D3 3474 (28.4%) 8749 (71.6%)
D2 3911 (28.0%) 10040 (72.0%)
D1 4219 (26.4%) 11774 (73.6%)
C5 4471 (24.5%) 13773 (75.5%)
C4 4773 (23.5%) 15507 (76.5%)
C3 4635 (21.8%) 16586 (78.2%)
C2 4460 (19.8%) 18120 (80.2%)
C1 4110 (17.4%) 19552 (82.6%)
B5 3424 (15.5%) 18661 (84.5%)
B4 3543 (13.8%) 22058 (86.2%)
B3 3288 (12.3%) 23367 (87.7%)
B2 2441 (10.9%) 20054 (89.1%)
B1 1891 (9.9%) 17291 (90.1%)
A5 1573 (8.5%) 16953 (91.5%)
A4 1109 (7.0%) 14680 (93.0%)
A3 614 (5.8%) 9962 (94.2%)
A2 461 (4.8%) 9106 (95.2%)
279 9450



In [75]:
```
count_plt(loan_data,'sub_grade','Sub Grade',width=24,height=16)
```



Inferences:

- Since the subgrade is implicit in the subgrade, we can ignore it. - The Loan Status is directly impacted by Sub-Grade. It is likely that a sub-grade will lead to a charge-off if the grade is not good

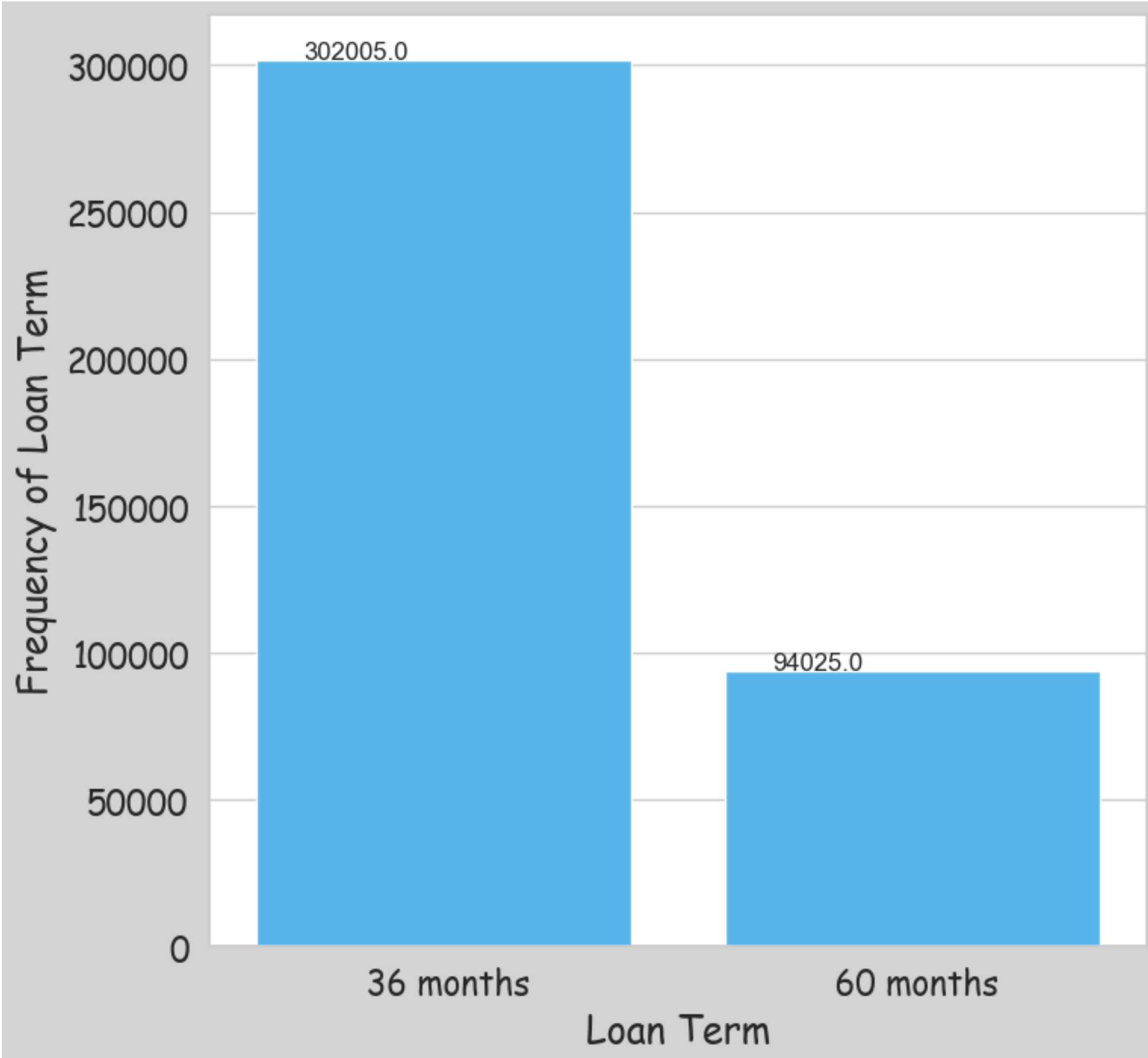
In [76]:
```
loan_data.drop('grade',axis=1,inplace=True)
```

Term

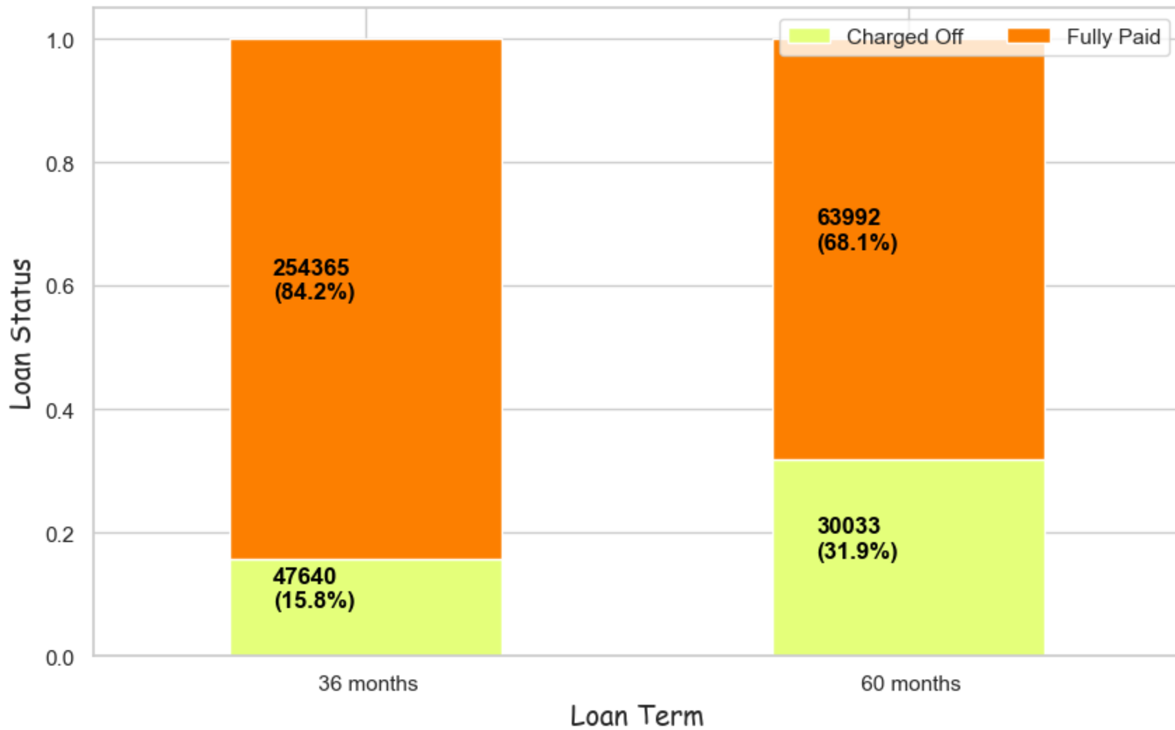
In [77]:
```
loan_data['term'].value_counts()
```

Out[77]:
```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```

In [78]:
```
count_plt(loan_data,'term','Loan Term',width=8,height=8)
```



In [79]:
```
stack_bar(loan_data,'term',"Loan Term")
```



Converting to integer value:

In [80]:
```
loan_data['term'] = loan_data['term'].apply(lambda term: np.int8(term.split()[0]))
```

Inferences:

- In comparison to 36-month (3 years) loans, 60-month (5 years) loans have a 2x higher rate of charge-offs. - A five-year loan has a probability of charged-off of 32%, which is much higher than a three-year loan.

emp_title

In [81]:
```
loan_data['emp_title'].nunique()
```

Out[81]:
```
173105
```

In [82]:
```
loan_data['emp_title'].value_counts()
```

Out[82]:
```
Teacher                       4389
Manager                       4250
Registered Nurse              1856
RN                            1846
Supervisor                    1830
                              ...
Postman                          1
McCarthy & Holthus, LLC          1
jp flooring                      1
Histology Technologist           1
Gracon Services, Inc             1
Name: emp_title, Length: 173105, dtype: int64
```

Inferences:

- The two top job titles that take most loans are teacher and manager.

In [83]:
```
loan_data.loc[loan_data['emp_title'] == 'Manager', 'loan_status'].value_counts()
```

Out[83]:
```
Fully Paid     3321
Charged Off     929
Name: loan_status, dtype: int64
```

In [84]:
```
929/(3321+929)
```

Out[84]:
```
0.21858823529411764
```

In [85]:
```
loan_data.loc[loan_data['emp_title'] == 'Technition', 'loan_status'].value_counts()
```

Out[85]:
```
Charged Off    6
Fully Paid     1
Name: loan_status, dtype: int64
```

In [86]:
```
(loan_data['emp_title'].nunique()/loan_data.shape[0])*100
```

Out[86]:
```
43.710072469257376
```

Inference:

- In total, 43% of the total records has a different employee title. However, this feature is not very useful without creating categories. Thus, it has been removed.

In [87]:
```
loan_data.drop('emp_title',axis=1,inplace=True)
```

loan_d

In [88]:
```
loan_data['issue_d'].value_counts(dropna=False)
```

Out[88]:
```
Oct-2014    14846
Jul-2014    12609
Jan-2015    11705
Dec-2013    10618
Nov-2013    10496
            ...
Jul-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: issue_d, Length: 115, dtype: int64
```

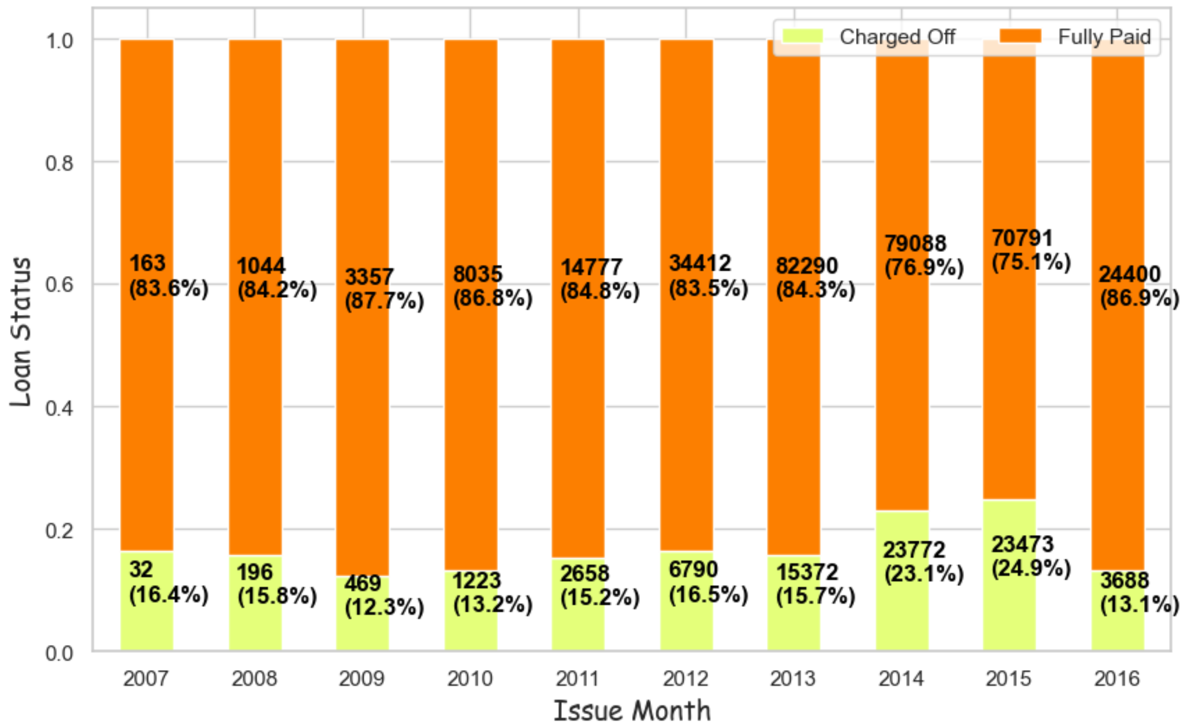
In [89]:
```
loan_data["issue_d"] = pd.to_datetime(loan_data['issue_d'])
```

In [90]:
```
loan_data['issue_d'] = loan_data['issue_d'].dt.year
```

In [91]:
```
loan_data['issue_d'].value_counts(dropna=False)
```

Out[91]:
```
2014    102860
2013     97662
2015     94264
2012     41202
2016     28088
2011     17435
2010      9258
2009      3826
2008      1240
2007       195
Name: issue_d, dtype: int64
```

In [92]:
```
stack_bar(loan_data,'issue_d',"Issue Month")
```



Inferences:

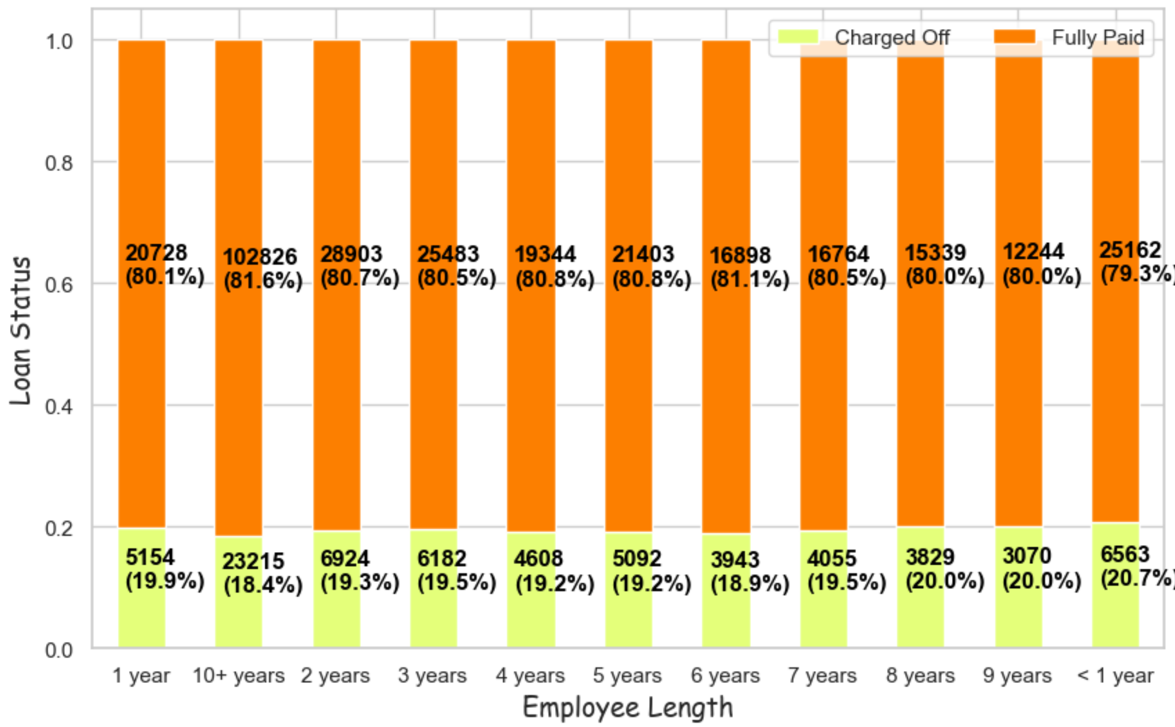
-Based on the issue month from year 2013 to 2015, a slight increase was noted for loan getting charged-off . - Data for 2016 shows less charged off than previous years, which could be due to not being full year data.

emp_length

In [93]:
```
loan_data['emp_length'].value_counts(dropna=False)
```

Out[93]:
```
10+ years    126041
2 years       35827
< 1 year      31725
3 years       31665
5 years       26495
1 year        25882
4 years       23952
6 years       20841
7 years       20819
8 years       19168
NaN           18301
9 years       15314
Name: emp_length, dtype: int64
```

In [94]:
```
stack_bar(loan_data,'emp_length',"Employee Length")
```



Inference:

- Loan status is constant with the length of the employee. We therefore removed this feature.

In [95]:
```
loan_data.drop('emp_length',axis=1,inplace=True)
```

Home Ownership

In [96]:
```
loan_data['home_ownership'].value_counts()
```

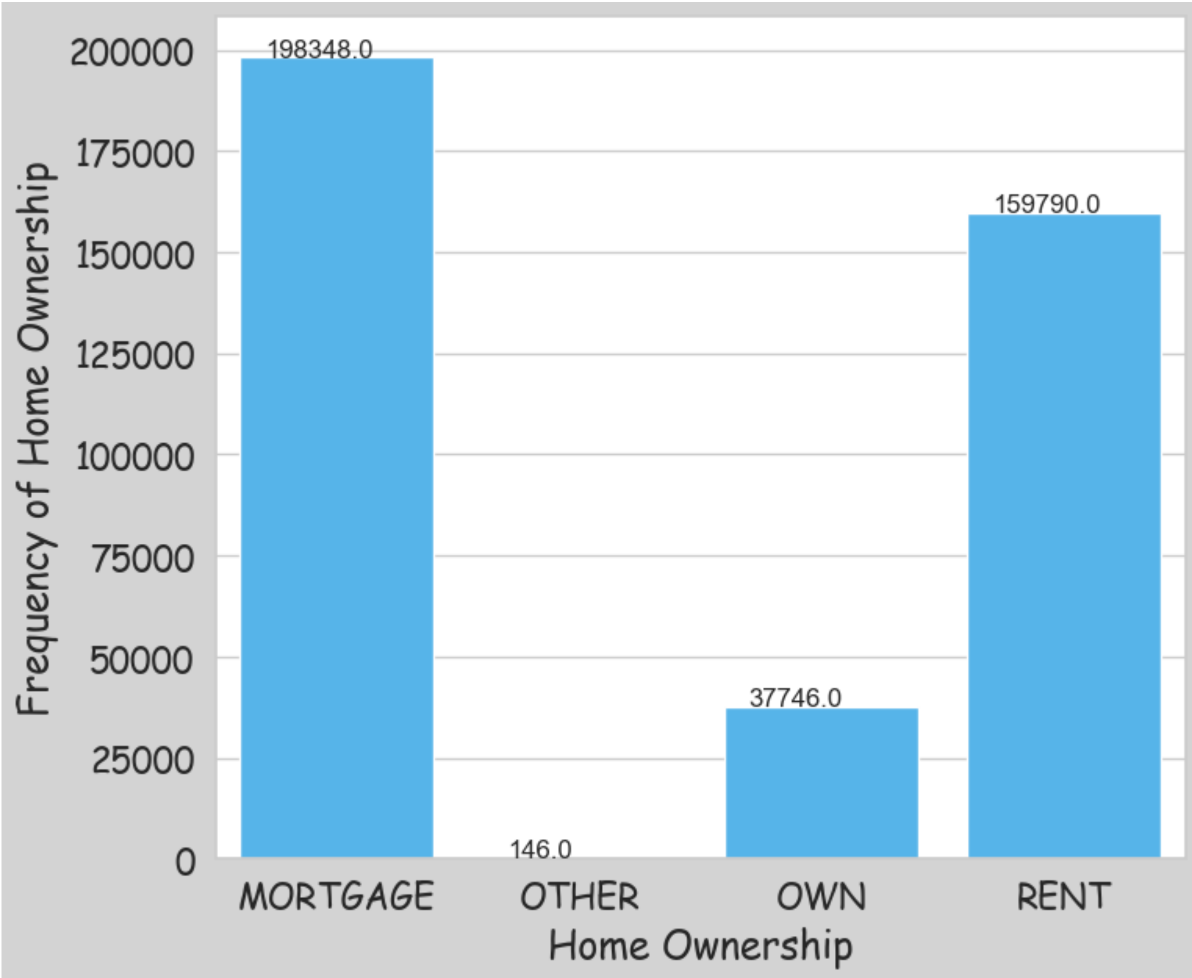
Out[96]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

Inference:

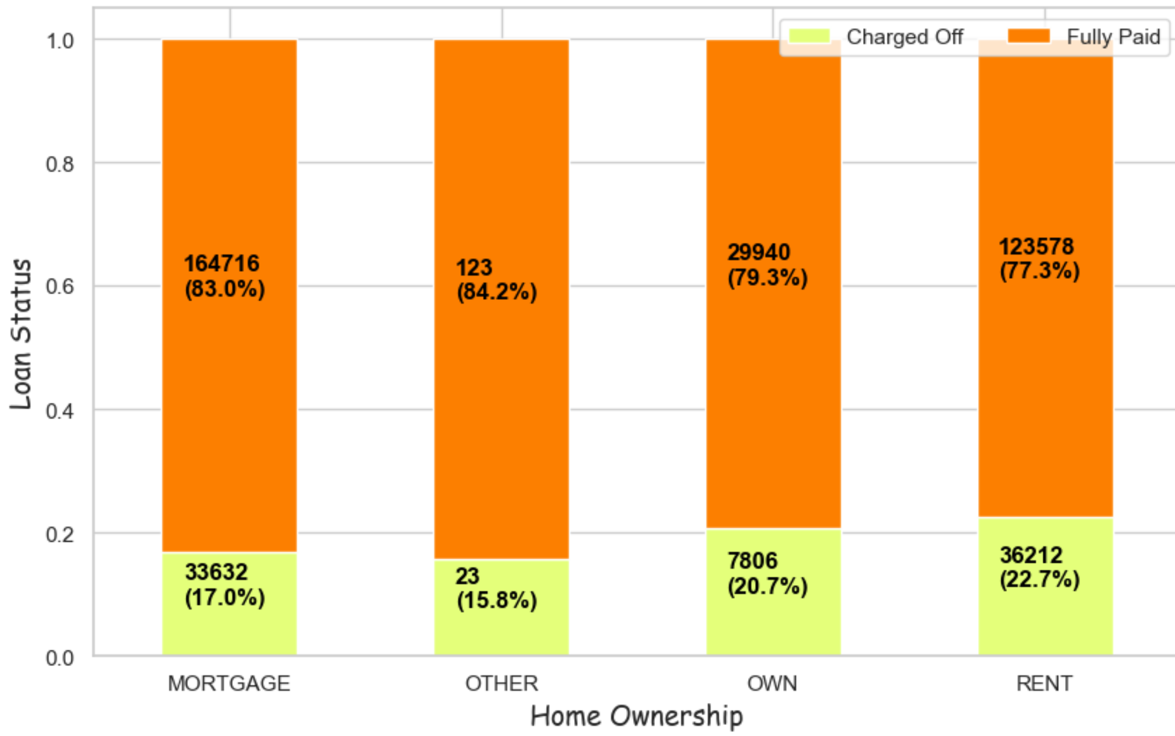
- Home Ownership Category - OTHER will be combined with NONE & ANY

In [97]:
```
loan_data['home_ownership'].replace(['NONE', 'ANY'], 'OTHER', inplace=True)
```

In [98]:
```
count_plt(loan_data,'home_ownership','Home Ownership',width=8,height=7)
```



In [99]:
```
stack_bar(loan_data,'home_ownership','Home Ownership')
```



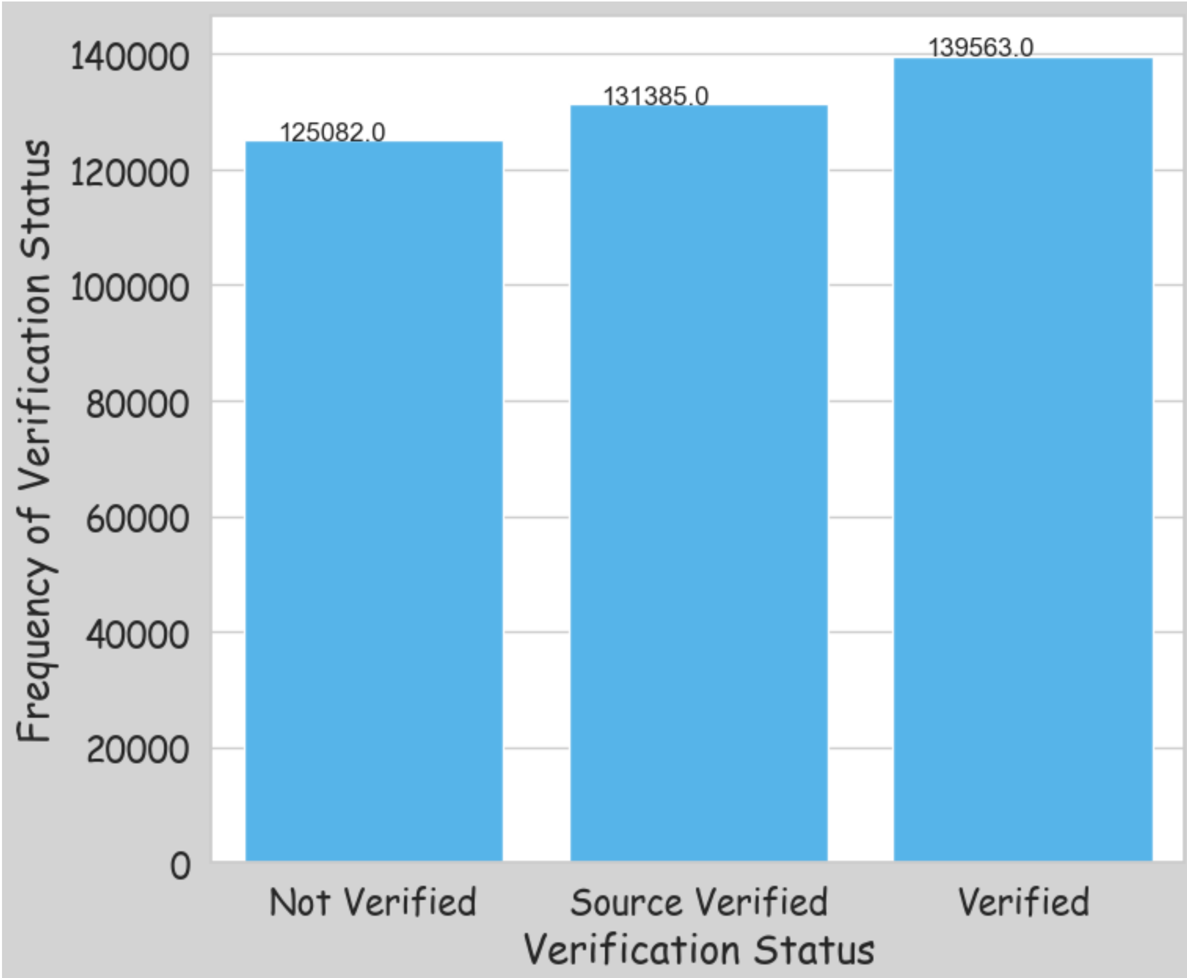
Inference:

- We can see from the above graph that there is a high risk of Charge-off for owners and rented homes

Verification Status

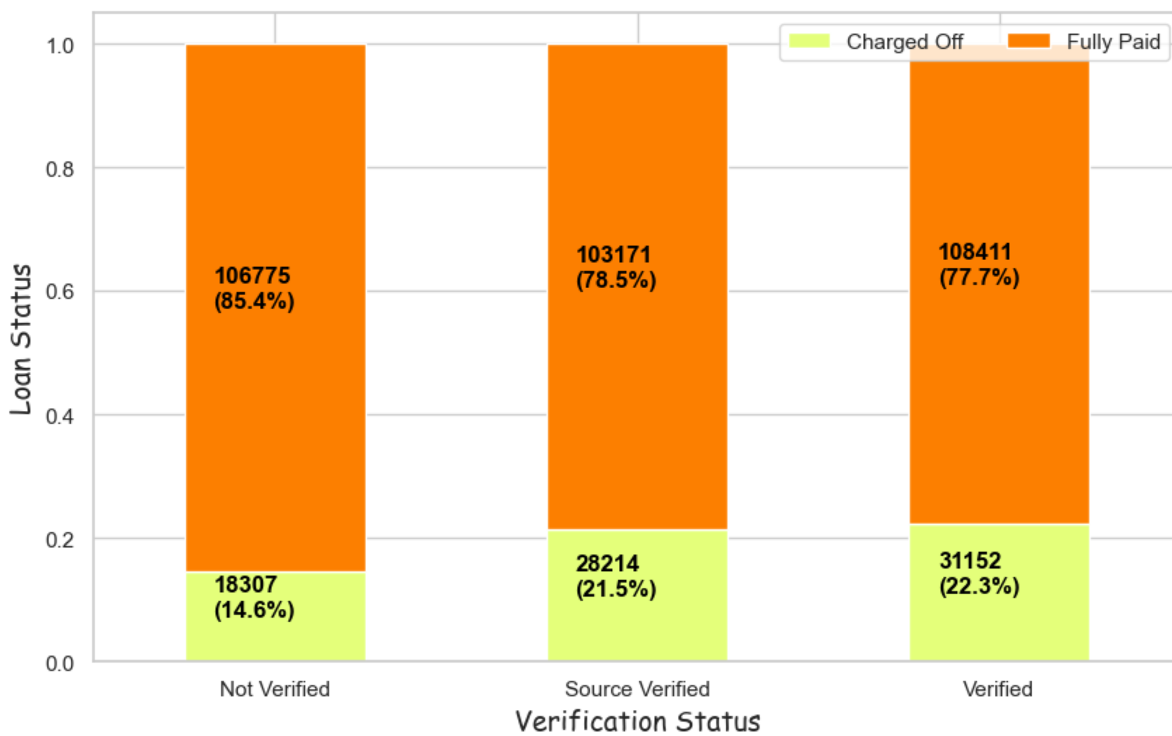
In [100…

```
count_plt(loan_data,'verification_status','Verification Status',width=8,height=7)
```



In [101…

```
stack_bar(loan_data,'verification_status',"Verification Status")
```



Inference:

- Although income is verified, the charge-off rate is higher.

Purpose of the loan

In [102…

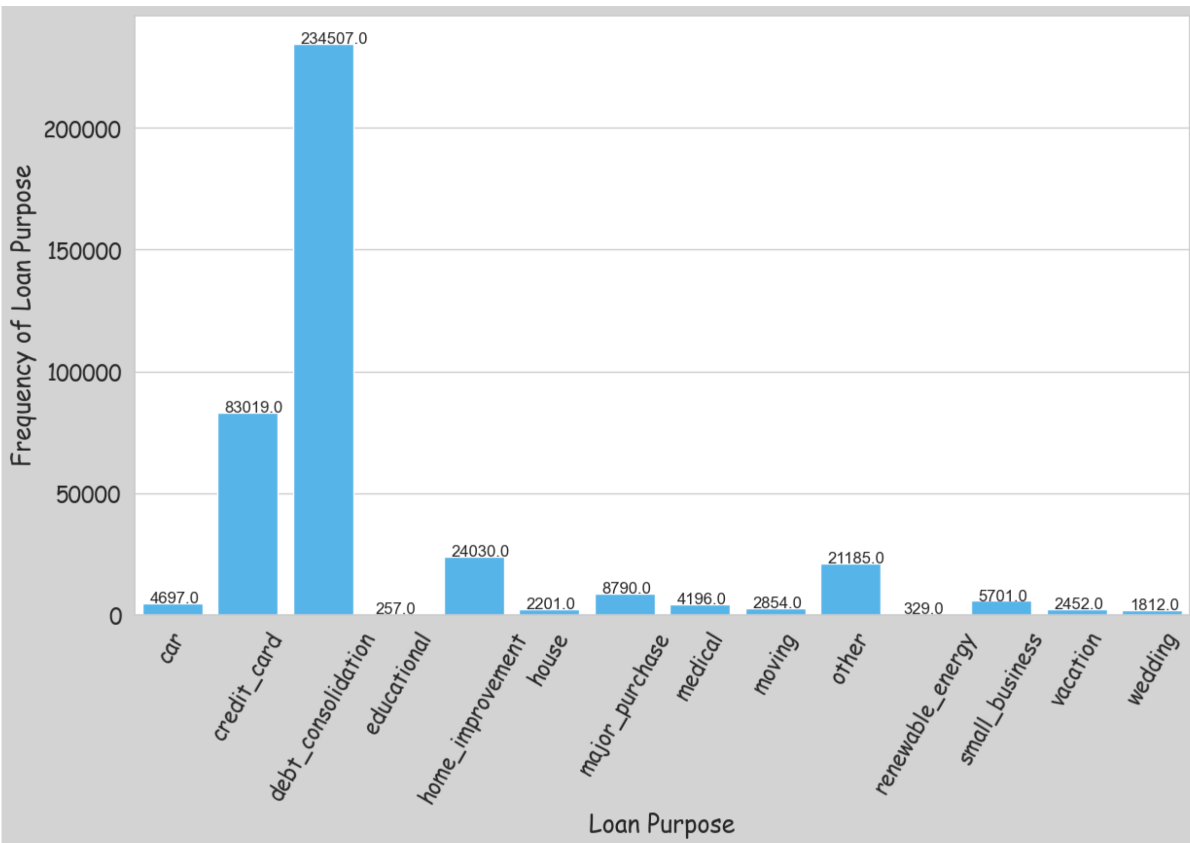
```
loan_data['purpose'].value_counts()
```

Out[102]:

```
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
car                     4697
medical                 4196
moving                  2854
vacation                2452
house                   2201
wedding                 1812
renewable_energy         329
educational              257
Name: purpose, dtype: int64
```

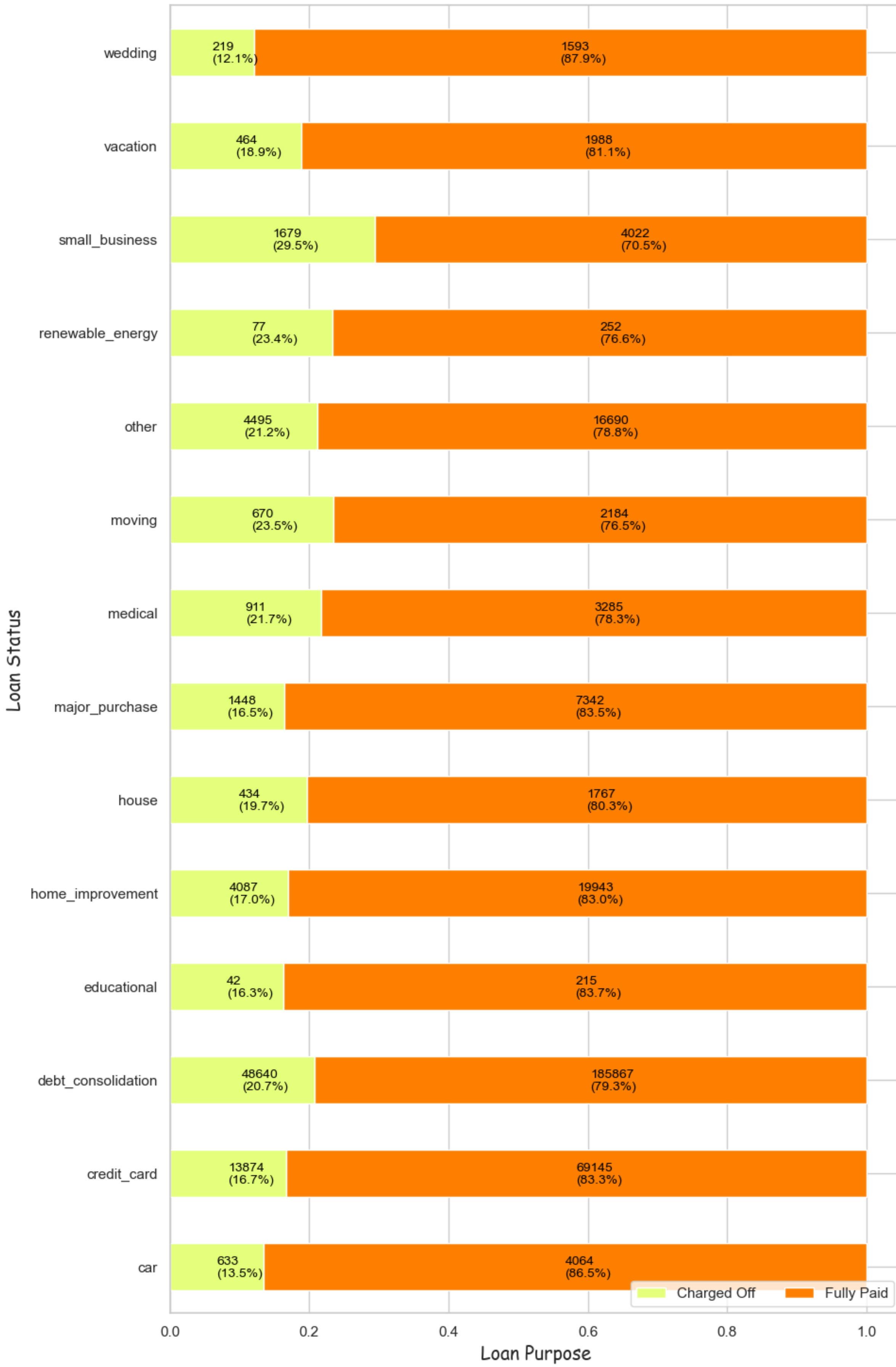
In [103…

```
count_plt(loan_data,'purpose','Loan Purpose',width=14,height=8,rotation=60)
```



In [104…

```
stack_bar_h(loan_data,'purpose',"Loan Purpose")
```


Loan Status
Loan Purpose
wedding
vacation
small_business
renewable_energy
other
moving
medical
major_purchase
house
home_improvement
educational
debt_consolidation
credit_card
car
219 (12.1%)
1593 (87.9%)
464 (18.9%)
1988 (81.1%)
1679 (29.5%)
4022 (70.5%)
77 (23.4%)
252 (76.6%)
4495 (21.2%)
16690 (78.8%)
670 (23.5%)
2184 (76.5%)
911 (21.7%)
3285 (78.3%)
1448 (16.5%)
7342 (83.5%)
434 (19.7%)
1767 (80.3%)
4087 (17.0%)
19943 (83.0%)
42 (16.3%)
215 (83.7%)
48640 (20.7%)
185867 (79.3%)
13874 (16.7%)
69145 (83.3%)
633 (13.5%)
4064 (86.5%)
Charged Off
Fully Paid
0.0
0.2
0.4
0.6
0.8
1.0

Inference

- When the aim of the business is to start or to invest in a small business, there is a 30% chance of getting charged-off

Title:

In [105…
```
loan_data['title'].nunique()
```
Out[105]:
```
48817
```

In [106…
```
loan_data['title'].value_counts().head(5)
```
Out[106]:
```
Debt consolidation         152472
Credit card refinancing     51487
Home improvement            15264
Other                       12930
Debt Consolidation          11608
Name: title, dtype: int64
```

In [107…
```
loan_data['title'].value_counts().head(5)
```
Out[107]:
```
Debt consolidation         152472
Credit card refinancing     51487
Home improvement            15264
Other                       12930
Debt Consolidation          11608
Name: title, dtype: int64
```

In [108…
```
loan_data.drop('title',axis=1,inplace=True)
```

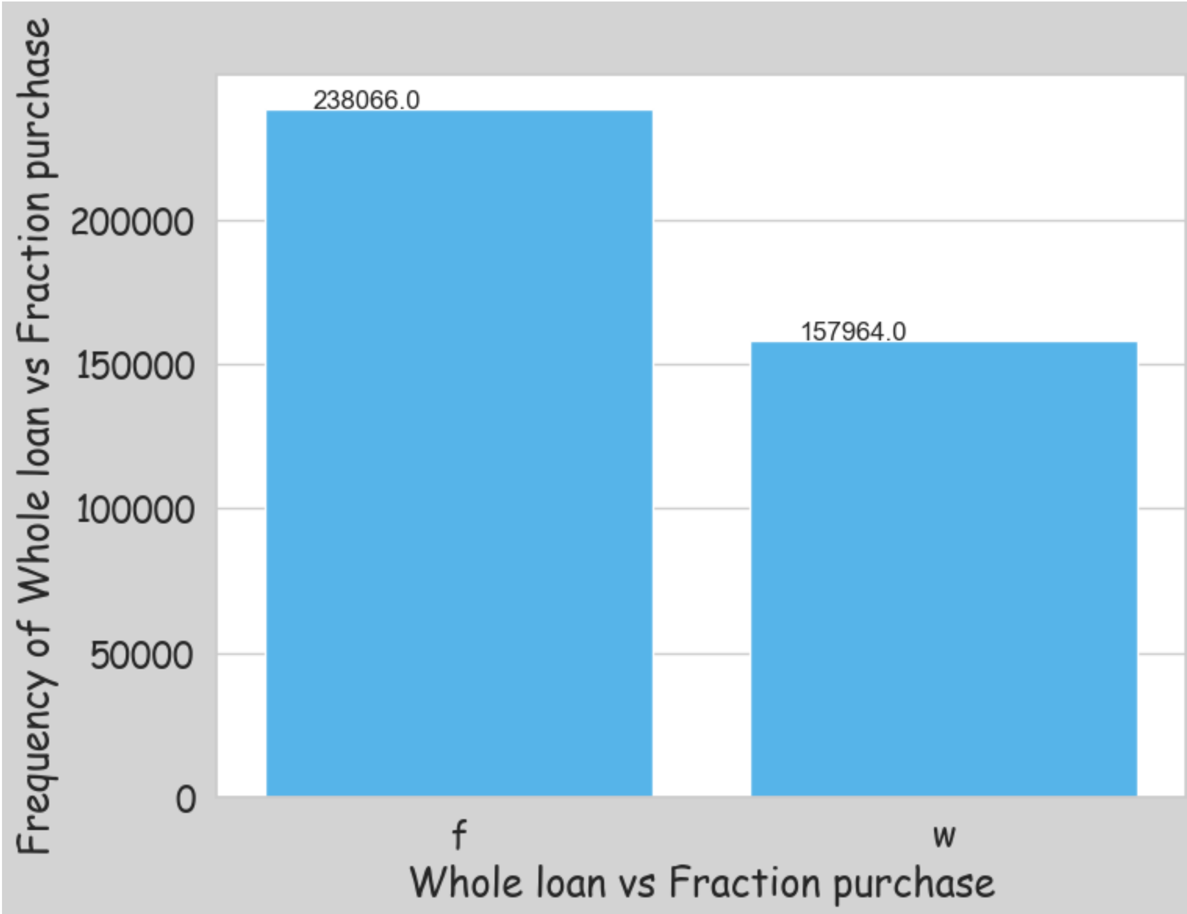
Inferences:

- It appears the title is a subcategory of loan purpose. With 48K+ different sub-purposes and already capturing all the information in the purpose variable, we can remove this variable.

Initial list Status

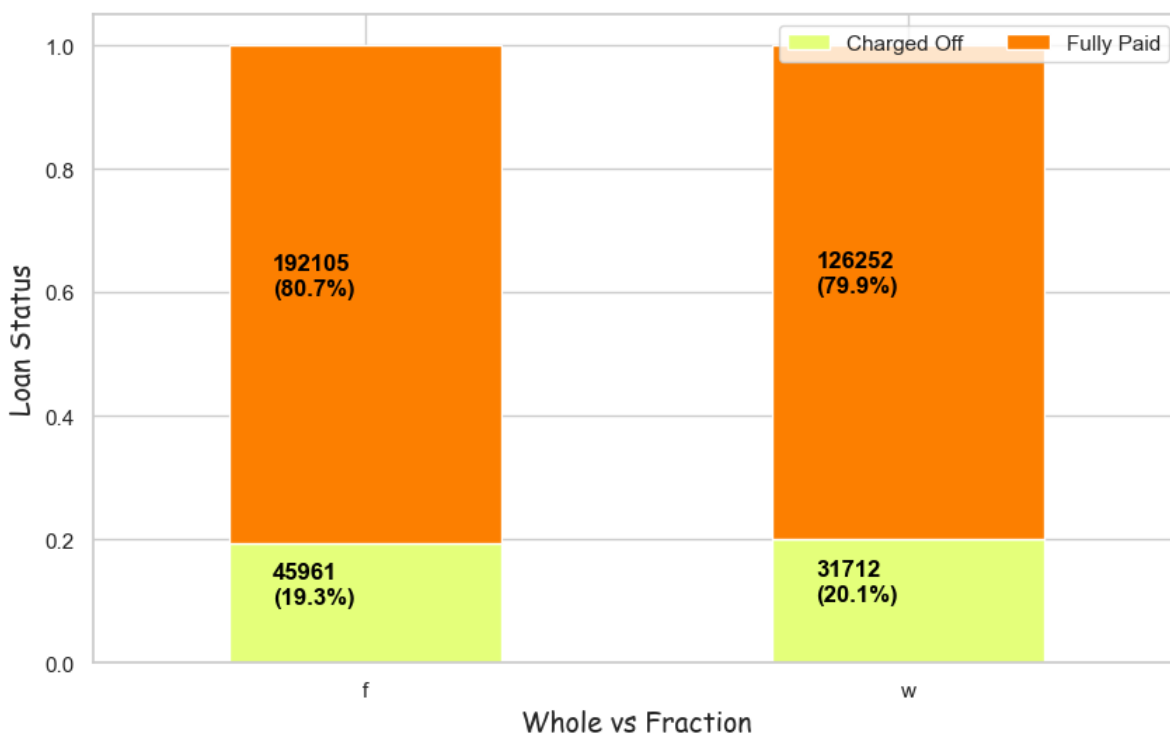
- Whole loan vs Fraction purpose
- Initial list status indicates the initial listing status of the loan. Possible values are W, F. W stands for whole loans, that is, available to investors to be purchased in their entirety (Borrowers benefit from getting 'instant funding').
- Lending club provides a randomized subset of loans by grade available to purchase as a whole loan for a brief period of time (12 hours). The rest are available for fractional purchase.

In [109…
```
count_plt(loan_data,'initial_list_status','Whole loan vs Fraction purchase',width=8
```



In [110…

```
stack_bar(loan_data,'initial_list_status',"Whole vs Fraction")
```

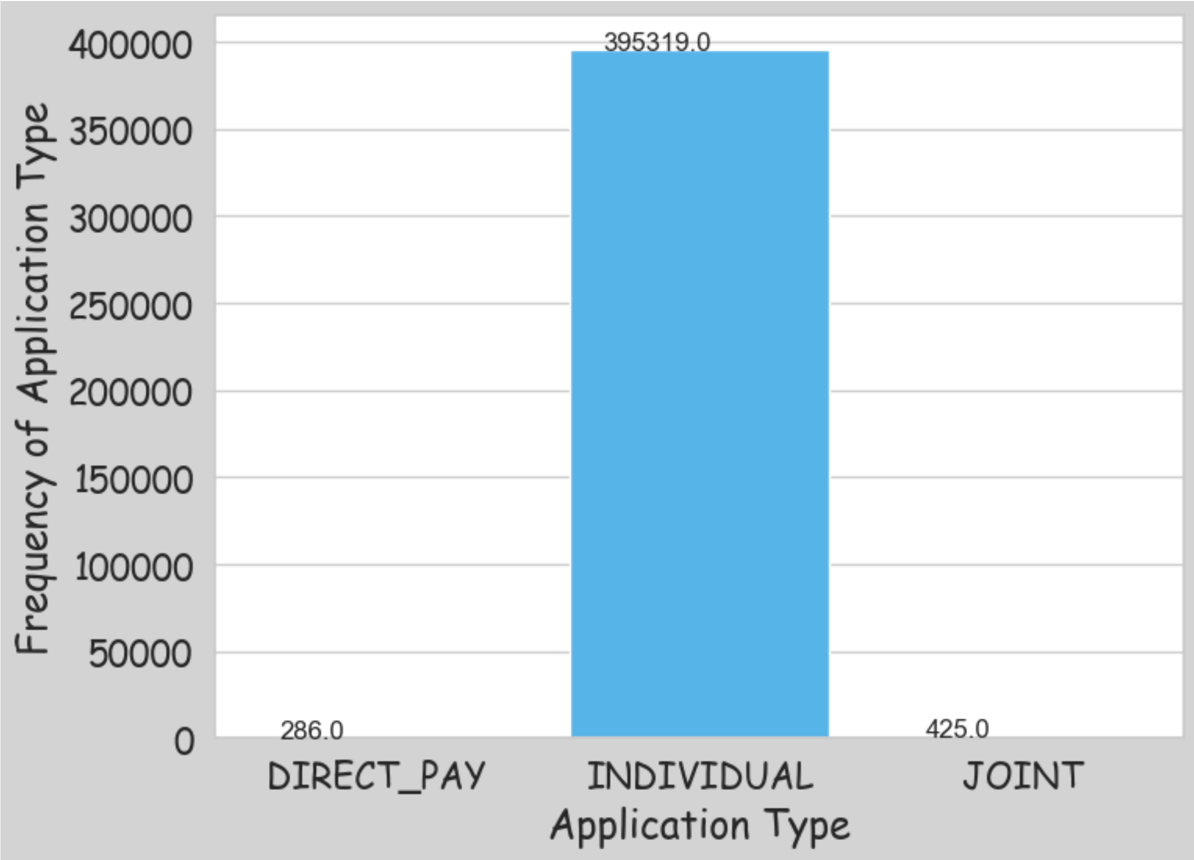


Application Type

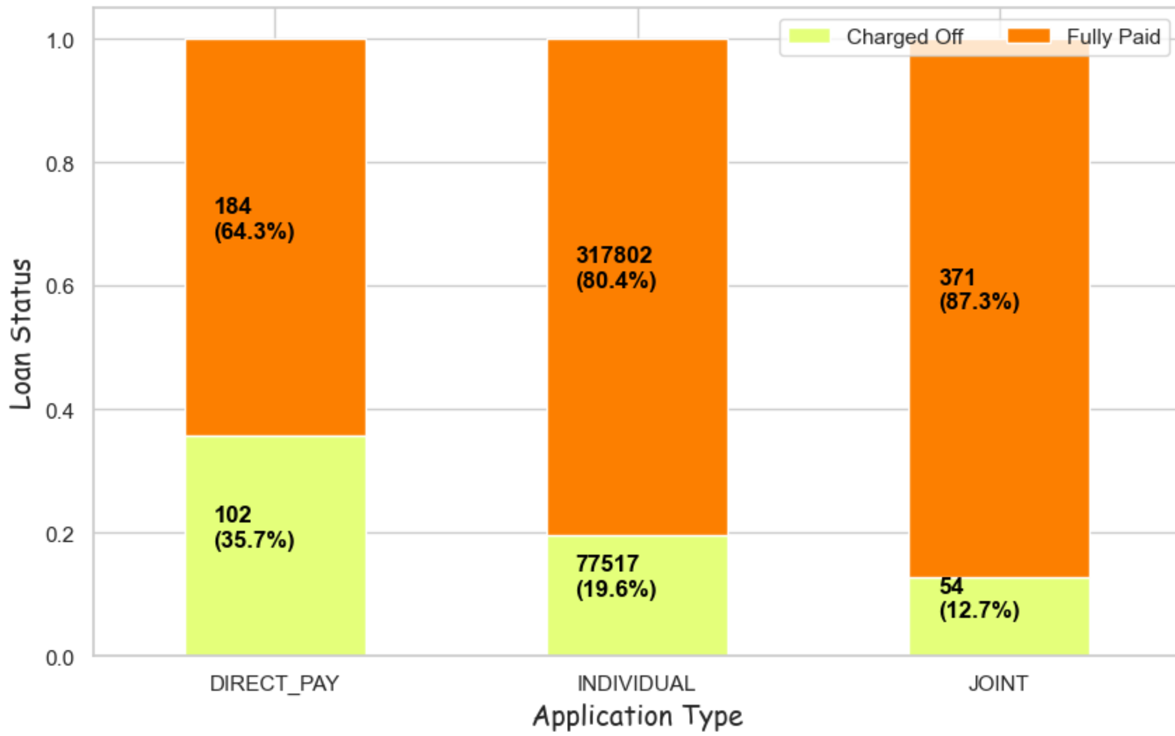
In [111…
```
loan_data['application_type'].value_counts()
```

Out[111]:
```
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: application_type, dtype: int64
```

In [112…
```
count_plt(loan_data,'application_type','Application Type',width=8,height=6)
```



In [113…
```
stack_bar(loan_data,'application_type',"Application Type")
```



Inference:

- The Direct Pay Application Type has a high chance of getting charged-off. Meanwhile, joint pay has a slighty lower chance of being charged off than individual pay

Address:

In [114…
```
loan_data['address'].nunique()
```

Out[114]:
```
393700
```

In [115…
```
(loan_data['address'].nunique()/loan_data.shape[0])*100
```

Out[115]:
```
99.41166073277276
```

Inference

- We can group the data by zipcode, which might provide us with more insights.
- In 99% of cases, the values are different. It would be helpful if the data based on state was provided. Hence Fropping the column

In [116…
```
loan_data.shape
```

Out[116]:
```
(396030, 23)
```

earliest_cr_line

- The month the borrower's earliest reported credit line was opened

In [117…
```
loan_data['earliest_cr_line'].nunique()
```

Out[117]:
```
684
```

In [118…
```
loan_data["earliest_cr_line"] = pd.to_datetime(loan_data['earliest_cr_line'])
```

In [119…
```
loan_data['earliest_cr_line'] = loan_data['earliest_cr_line'].dt.year
```

In [120…
```
loan_data['earliest_cr_line'].value_counts()
```

Out[120]:
```
2000    29366
2001    29083
1999    26491
2002    25901
2003    23657
        ...
1951        3
1950        3
1953        2
1944        1
1948        1
Name: earliest_cr_line, Length: 65, dtype: int64
```

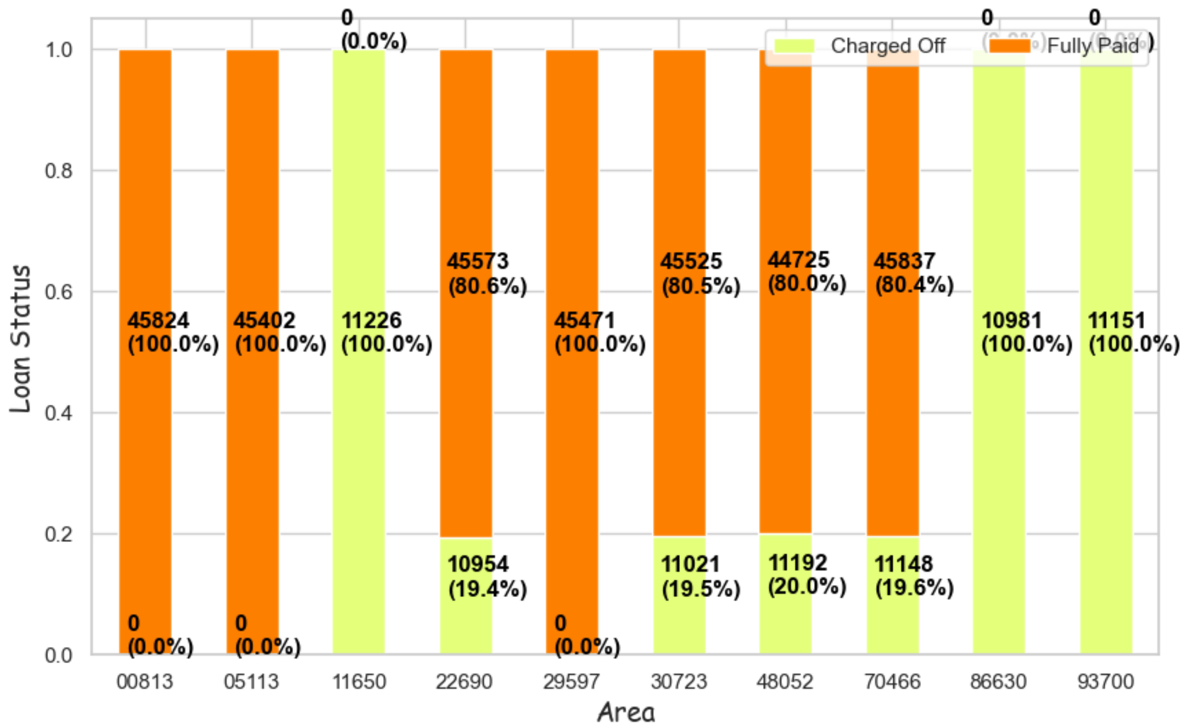
Feature Engineering

address

- Extracting the Zipcode from the address

In [121…
```
loan_data['zipcode'] = loan_data['address'].apply(lambda address:address[-5:])
```

In [122…
```
stack_bar(loan_data,'zipcode',"Area")
```



Inference

- Based on the above graph, we can see that zip codes 11650,86630, and 93700 have a 100% probability of getting charged-off.

In [123…

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

Inference

- Important information is already captured as part of zipcode. Hence dropping the column

dti

- According to our previous analysis, dti greater than 50 has 35% of the loan to be charged-off, whereas dti less than 10 has only 13% of the loan to be charged-off.
- Lets divide the dti value into bins to understand the impact on the loan_status

In [124…

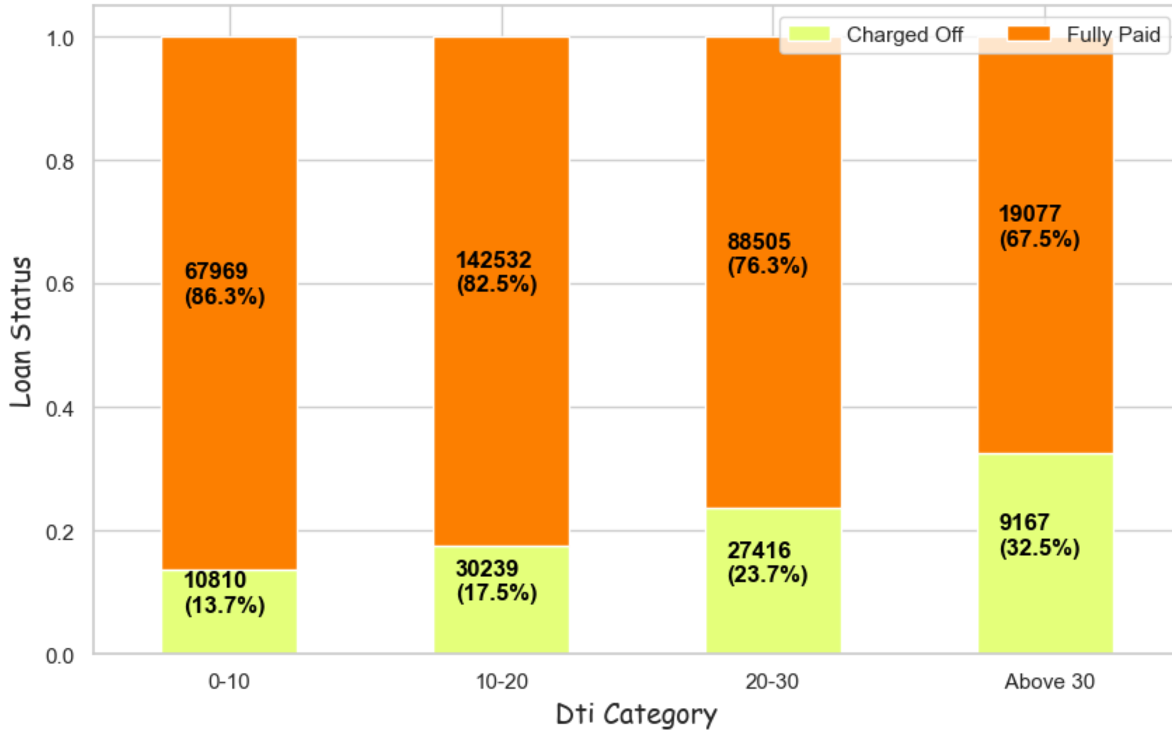
```
bins = [0,10,20,30,1000]
labels =["0-10","10-20","20-30",  "  Above 30"]
loan_data['dti_cat'] = pd.cut(loan_data['dti'], bins,labels=labels)
```

In [125…

```
loan_data['dti_cat'].head()
```

Out[125]:
```
0          20-30
1          20-30
2          10-20
3           0-10
4       Above 30
Name: dti_cat, dtype: category
Categories (4, object): ['0-10' < '10-20' < '20-30' < '  Above 30']
```

In [126...
```
stack_bar(loan_data,'dti_cat',"Dti Category")
```



In [127...
```
loan_data.drop('dti',axis=1,inplace=True)
```

Inferences:

- It is clear that as the dti value increases, so does the probability of being charged off.

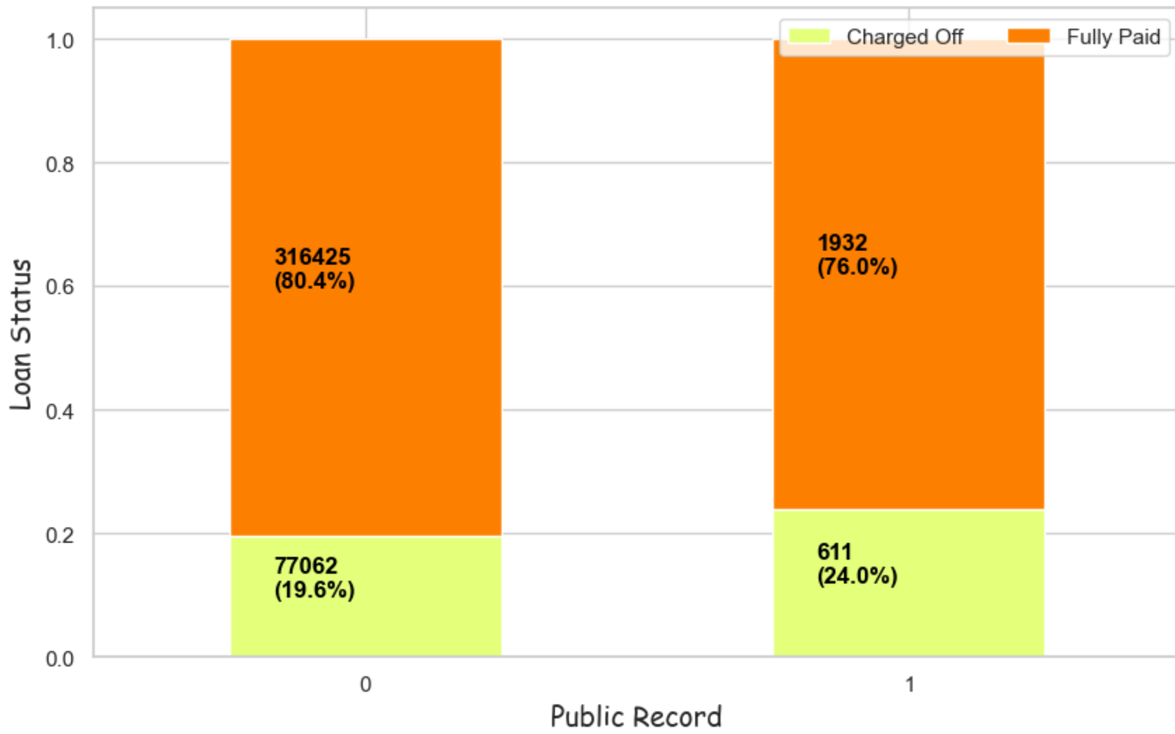
pub_rec

In [128...
```
def pub_rec(num):
    if num <= 2:
        return 0
    elif num >= 0:
        return 1
    else:
        return num
```

In [129...
```
loan_data['pub_rec_cat'] = loan_data.pub_rec.apply(pub_rec)
```

In [130...
```
loan_data["pub_rec_cat"] = loan_data["pub_rec_cat"].astype("category")
```

In [131...
```
stack_bar(loan_data,'pub_rec_cat',"Public Record")
```



Inference:

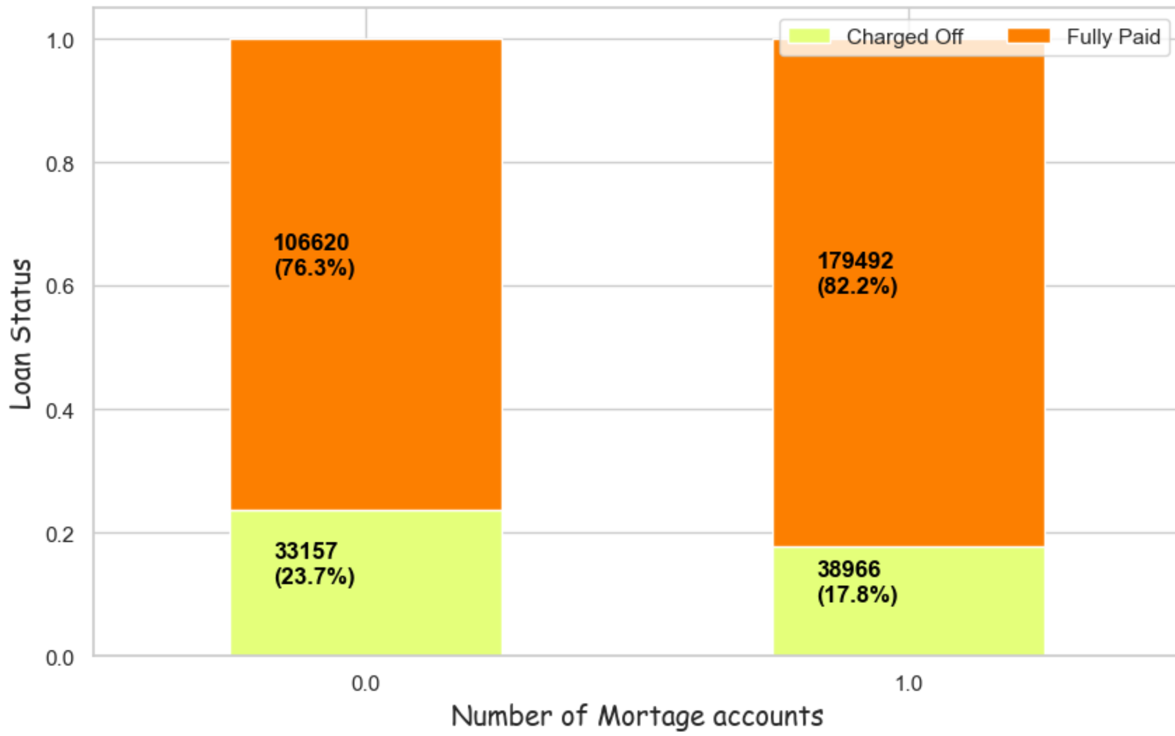
- If Public record having derogatory value more than 2 then we can see loan getting charged-off by 24%

mort_acc

In [132...
```
def mort_acc(num):
    if num == 0.0:
        return 0
    elif num >= 1.0:
        return 1
    else:
        return num
```

In [133...
```
loan_data['mort_acc_cat'] = loan_data.mort_acc.apply(mort_acc)
loan_data["mort_acc_cat"] = loan_data["mort_acc_cat"].astype("category")
```

In [134...
```
stack_bar(loan_data,'mort_acc_cat',"Number of Mortage accounts")
```



In [135...

```
loan_data.drop('mort_acc',axis=1,inplace=True)
```

Inference

- The probability of the loan getting charged off is 24% if the borrower does not have a mortgage account.

pub_rec_bankruptcies

In [136...

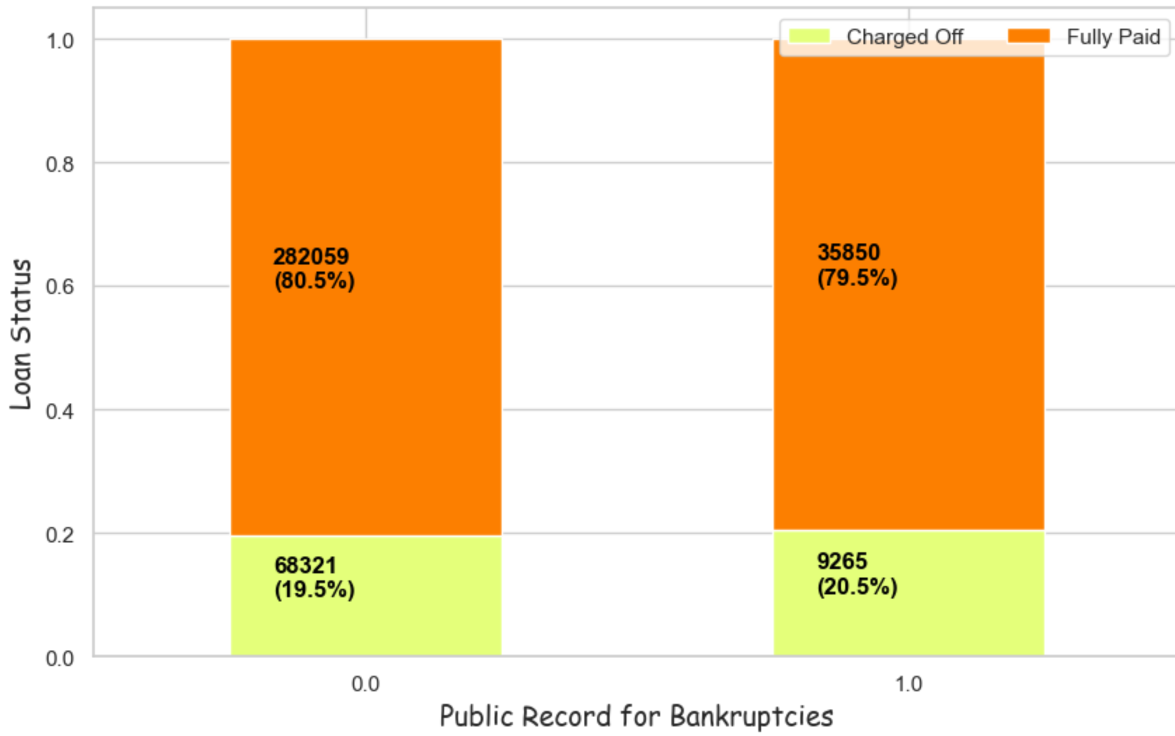
```
def pub_rec_bankruptcies(num):
    if num == 0.0:
        return 0
    elif num >= 1.0:
        return 1
    else:
        return num
```

In [137...

```
loan_data['pub_rec_bankruptcies_cat'] = loan_data.pub_rec_bankruptcies.apply(pub_re
loan_data["pub_rec_bankruptcies_cat"] = loan_data["pub_rec_bankruptcies_cat"].astyp
```

In [138...

```
stack_bar(loan_data,'pub_rec_bankruptcies_cat',"Public Record for Bankruptcies")
```



In [139…
```
loan_data.drop('pub_rec_bankruptcies',axis=1,inplace=True)
```

Inference

- If there are more bankruptcies on public records than 1 then we can see the loan getting charged off by 20%

loan_stats (Target Variable)

In [140…
```
loan_data['loan_status'].unique()
```

Out[140]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [141…
```
def loan_status(str_):
    if str_ == 'Charged Off':
        return 1
    else:
        return 0
```

In [142…
```
loan_data['loan_status'] = loan_data.loan_status.apply(loan_status)
```

In [143…
```
loan_data['loan_status'].unique()
```

Out[143]:
```
array([0, 1], dtype=int64)
```

In [144…
```
loan_data.shape
```

Out[144]:
```
(396030, 24)
```

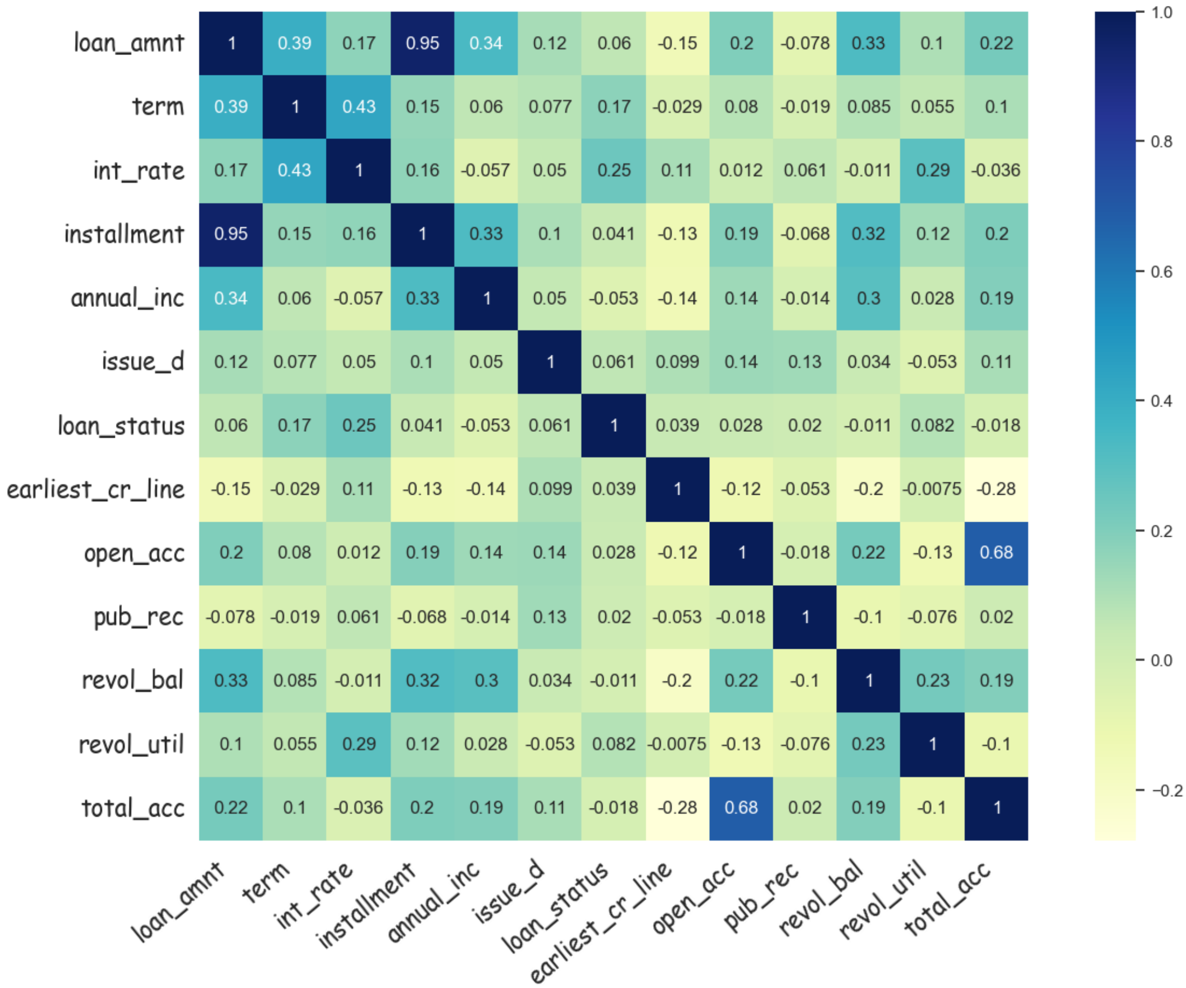
Inferences:

- Overall we have 23 features which shows some relations w.r.t. target variable.
- After EDA we have removed few features
    - emp_length
    - emp_title
    - grade
    - title
- Few new features are derived from existing features
    - pub_rec_bankruptcies_cat
    - dti_cat
    - zipcode
    - mort_acc_cat
    - pub_rec_cat

Checking Correlation

In [145…

```
plt.figure(figsize = (16, 10))
ax = sns.heatmap(loan_data.corr(),
            annot=True,cmap='YlGnBu',square=True)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=40,fontsize=16,family = "Comic Sans MS",
    horizontalalignment='right')

ax.set_yticklabels(
    ax.get_yticklabels(),
    rotation=0,fontsize=16,family = "Comic Sans MS",
    horizontalalignment='right')

plt.show()
```



Inferences:

- Loan Amount ad installment is highly corelated with 95%.
- Not much correlation between other variables can be observed. open_acc and total_acc are most co-related features with 68%

Handling Categorical Variables

- Categorical to Numerical - Our training data more useful and expressive, and it can be rescaled easily. By using numeric values, we more easily determine a probability for our values. In particular, one hot encoding is used for our output values, since it provides more nuanced predictions than single labels

- One Hot Encoding

We use this categorical data encoding technique when the features are nominal(do not have any order). In one hot encoding, for each level of a categorical feature, we create a new variable. Each category is mapped with a binary variable containing either 0 or 1. Here, 0 represents the absence, and 1 represents the presence of that category.

In [146…

```
loan_data.columns
```

Out[146]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'sub_grade',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'purpose', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'application_type', 'zipcode', 'dti_cat', 'pub_rec_cat', 'mort_acc_cat',
       'pub_rec_bankruptcies_cat'],
      dtype='object')
```

In [147…

```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 24 columns):
 #   Column                    Non-Null Count   Dtype   
---  ------                    --------------   -----   
 0   loan_amnt                 396030 non-null  float64 
 1   term                      396030 non-null  int8    
 2   int_rate                  396030 non-null  float64 
 3   installment               396030 non-null  float64 
 4   sub_grade                 396030 non-null  object  
 5   home_ownership            396030 non-null  object  
 6   annual_inc                396030 non-null  float64 
 7   verification_status       396030 non-null  object  
 8   issue_d                   396030 non-null  int64   
 9   loan_status               396030 non-null  int64   
 10  purpose                   396030 non-null  object  
 11  earliest_cr_line          396030 non-null  int64   
 12  open_acc                  396030 non-null  float64 
 13  pub_rec                   396030 non-null  float64 
 14  revol_bal                 396030 non-null  float64 
 15  revol_util                395754 non-null  float64 
 16  total_acc                 396030 non-null  float64 
 17  initial_list_status       396030 non-null  object  
 18  application_type          396030 non-null  object  
 19  zipcode                   396030 non-null  object  
 20  dti_cat                   395715 non-null  category
 21  pub_rec_cat               396030 non-null  category
 22  mort_acc_cat              358235 non-null  category
 23  pub_rec_bankruptcies_cat  395495 non-null  category
dtypes: category(4), float64(9), int64(3), int8(1), object(7)
memory usage: 59.3+ MB
```

In [148…

```
loan_data.shape
```

Out[148]:

```
(396030, 24)
```

In [149…

```
cat_columns = ['sub_grade', 'home_ownership','verification_status', 'issue_d',
               'purpose',  'initial_list_status', 'application_type','zipcode',
       'dti_cat', 'pub_rec_cat', 'mort_acc_cat', 'pub_rec_bankruptcies_cat']
```

In [150…

```
dummyVar = pd.get_dummies(loan_data[cat_columns],drop_first=True)
dummyVar.shape
```

Out[150]:

```
(396030, 71)
```

In [151…

```
dummyVar.head()
```

Out[151]:

	issue_d	sub_grade_A2	sub_grade_A3	sub_grade_A4	sub_grade_A5	sub_grade_B1	sub_grade_B:
0	2015	0	0	0	0	0	(
1	2015	0	0	0	0	0	(
2	2015	0	0	0	0	0	(
3	2014	1	0	0	0	0	(
4	2013	0	0	0	0	0	(

5 rows × 71 columns

In [152…

```
# Merging the dummy variable to significant variable dataframe.
loan_data_encoded = pd.concat([loan_data,dummyVar],axis=1)
loan_data_encoded.shape
```

Out[152]:

```
(396030, 95)
```

In [153…

```
# Dropping origincal Categorical variables as no need. Already added them as numeri
loan_data_encoded.drop(cat_columns,axis=1,inplace=True)
loan_data_encoded.shape
```

Out[153]:

```
(396030, 82)
```

In [154…

```
loan_data_encoded.columns
```

Out[154]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc',
       'loan_status', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'sub_grade_A2', 'sub_grade_A3',
       'sub_grade_A4', 'sub_grade_A5', 'sub_grade_B1', 'sub_grade_B2',
       'sub_grade_B3', 'sub_grade_B4', 'sub_grade_B5', 'sub_grade_C1',
       'sub_grade_C2', 'sub_grade_C3', 'sub_grade_C4', 'sub_grade_C5',
       'sub_grade_D1', 'sub_grade_D2', 'sub_grade_D3', 'sub_grade_D4',
       'sub_grade_D5', 'sub_grade_E1', 'sub_grade_E2', 'sub_grade_E3',
       'sub_grade_E4', 'sub_grade_E5', 'sub_grade_F1', 'sub_grade_F2',
       'sub_grade_F3', 'sub_grade_F4', 'sub_grade_F5', 'sub_grade_G1',
       'sub_grade_G2', 'sub_grade_G3', 'sub_grade_G4', 'sub_grade_G5',
       'home_ownership_OTHER', 'home_ownership_OWN', 'home_ownership_RENT',
       'verification_status_Source Verified', 'verification_status_Verified',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_business',
       'purpose_vacation', 'purpose_wedding', 'initial_list_status_w',
       'application_type_INDIVIDUAL', 'application_type_JOINT',
       'zipcode_05113', 'zipcode_11650', 'zipcode_22690', 'zipcode_29597',
       'zipcode_30723', 'zipcode_48052', 'zipcode_70466', 'zipcode_86630',
       'zipcode_93700', 'dti_cat_10-20', 'dti_cat_20-30', 'dti_cat_  Above 30',
       'pub_rec_cat_1', 'mort_acc_cat_1.0', 'pub_rec_bankruptcies_cat_1.0'],
      dtype='object')
```

Train, Validation & Test split

In [155…
```
# Train & Test data split
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
```

In [156…
```
#putting features variables in X
X = loan_data_encoded.drop(['loan_status'], axis=1)

#putting response variables in Y
y = loan_data_encoded['loan_status']

# Splitting the data into train and test
X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X,y, train_size=0.8,test_size=0
```

Train and Cross-Validation Split

In [157…
```
# Splitting the Training Data into Train and Validation Sets:
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv,y_tr_cv,test_size=0.25,ra
```

Libraries used for model selection

In [158…

```
# For imputation to NAN values.
from sklearn.impute import SimpleImputer

# For rescaling we are using Standarad scaler
from sklearn.preprocessing import StandardScaler

# For logistic regression model
from sklearn.linear_model import LogisticRegression

# For feature selection
from sklearn.feature_selection import RFE

# For pipeline creation
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline

# For collecting different metrics.
from sklearn.metrics import f1_score
from sklearn import metrics
```

Utility Function Draw ROC curve

- True Positve rate vs False Positive rate

In [159…

```
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(6, 6))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return fpr, tpr, thresholds
```

Handling Missing Values

- Data is not complete without handling missing values and many machine learning algorithms do not allow missing values.
- It is essential to address any missing data before feeding it to your model.
- In the case study we are using SimpleImputer with median.

In [160…

```
imputer = SimpleImputer(strategy='median', missing_values=np.nan)
```

- Rescaling the Features

As per above table, features are varying in different ranges. This will be problem. It is important that we rescale the feature such that thay have a comparable scales. This can lead us time consuming during model evaluation.

So it is advices to Standardization and normalization so that units of coefficients obtained are in same scale. Two common ways of rescaling are

Standardization (mean-0, sigma-1) Min-Max scaling (Normization) In this case we are using Standardizationscaling

In [161…
```
scaler = StandardScaler()
```

Build Pipeline

- Imputation
- Rescaling
- Building the model

1. Basic Model creation

In [162…
```
pl_basic_logreg = Pipeline(steps=[('imputer',imputer),
                                  ('scaler',scaler),
                                  ('logistic_model',LogisticRegression())
                                 ])
```

In [163…
```
# Model Training:
pl_basic_logreg.fit(X_train,y_train)
```

Out[163]:
```
Pipeline(steps=[('imputer', SimpleImputer(strategy='median')),
                ('scaler', StandardScaler()),
                ('logistic_model', LogisticRegression())])
```

In [164…
```
# Training Data Prediction and F1 Score Calculation:
train_y_pred = pl_basic_logreg.predict(X_train)
train_score = f1_score(y_train, train_y_pred)
```

In [165…
```
print(train_y_pred)
```

```
[0 0 0 ... 0 0 0]
```

In [166…
```
# Printing the Training F1 Score:
print(" F1 Score for Basic Model (Train) ", train_score)
```

```
 F1 Score for Basic Model (Train)  0.6160009255777631
```

In [167…
```
# Handling Missing Data in Test Data:
X_test['revol_util'] = X_test['revol_util'].fillna(X_test['revol_util'].median())
```

In [168…
```
# Test Data Prediction and F1 Score Calculation:
y_pred_test = pl_basic_logreg.predict(X_test)
test_score = f1_score(y_test, y_pred_test)

# Printing the Test F1 Score:
print("F1 Score for Basic Model (Test) ",test_score)
```

```
F1 Score for Basic Model (Test)  0.6228797015313868
```

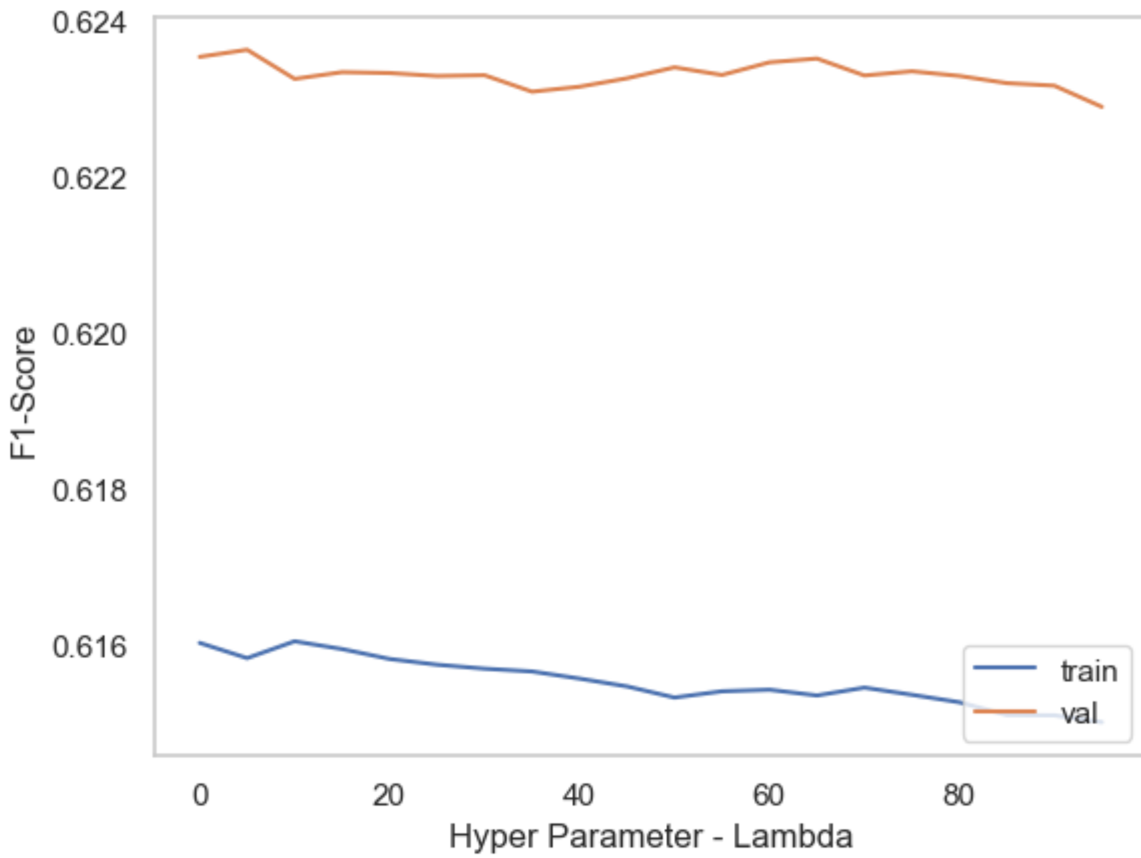
In [169…
```
print(y_pred_test)
```

```
[1 0 0 ... 0 0 1]
```

2. Using Hyper-parmeter Optimization

In [170…
```
train_scores = []
val_scores = []

la_low = 0.01
la_upp = 100
la_diff = 5

for lambda_ in np.arange(la_low,la_upp,la_diff):
    hp_logreg = Pipeline(steps=[('imputer',imputer),
                              ('scaler',scaler),
                              ('logistic_model',LogisticRegression(C=1/lambda_))
                             ])
    hp_logreg.fit(X_train, y_train)
    train_y_pred = hp_logreg.predict(X_train)
    val_y_pred = hp_logreg.predict(X_val)
    train_score = f1_score(y_train, train_y_pred)
    val_score = f1_score(y_val, val_y_pred)
    train_scores.append(train_score)
    val_scores.append(val_score)
```

In [171…
```
plt.figure()
plt.plot(list(np.arange(la_low,la_upp,la_diff)), train_scores, label="train")
plt.plot(list(np.arange(la_low,la_upp,la_diff)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("Hyper Parameter - Lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```



In [172...

```
# Model with lambda_best
best_hp_model = np.argmax(val_scores)
print(val_scores[best_hp_model])
```

```
0.6236187845303868
```

In [173...

```
l_best = la_low+la_diff*best_hp_model
best_hp_logreg = Pipeline(steps=[('imputer',imputer),
                              ('scaler',scaler),
                              ('logistic_model',LogisticRegression(C=1/l_best))
                             ])
best_hp_logreg.fit(X_train, y_train)

y_pred_hp_test = best_hp_logreg.predict(X_test)
test_score = f1_score(y_test, y_pred_hp_test)

print('F1 Score for Best Hyper-Parmeter Model (Test) ',test_score)
```

```
F1 Score for Best Hyper-Parmeter Model (Test)  0.6228839308967793
```

In [174...

```
print(f"Accuracy : {metrics.accuracy_score(y_test, y_pred_hp_test)*100}%")
print(f"recall_score : {metrics.recall_score(y_test, y_pred_hp_test)*100}%")
print(f"precision_score : {metrics.precision_score(y_test, y_pred_hp_test)*100}%")
print(f"f1_score : {metrics.f1_score(y_test, y_pred_hp_test)*100}%")
print(f"AUC score : {metrics.roc_auc_score( y_test, y_pred_hp_test)*100}%")
print(f"confusion_matrix :")
print(metrics.confusion_matrix(y_test, y_pred_hp_test))
```

```
Accuracy : 89.03113400499963%
recall_score : 46.3052597612133%
precision_score : 95.12130452074771%
f1_score : 62.28839308967793%
AUC score : 72.86382574945189%
confusion_matrix :
[[63343   368]
 [ 8320  7175]]
```

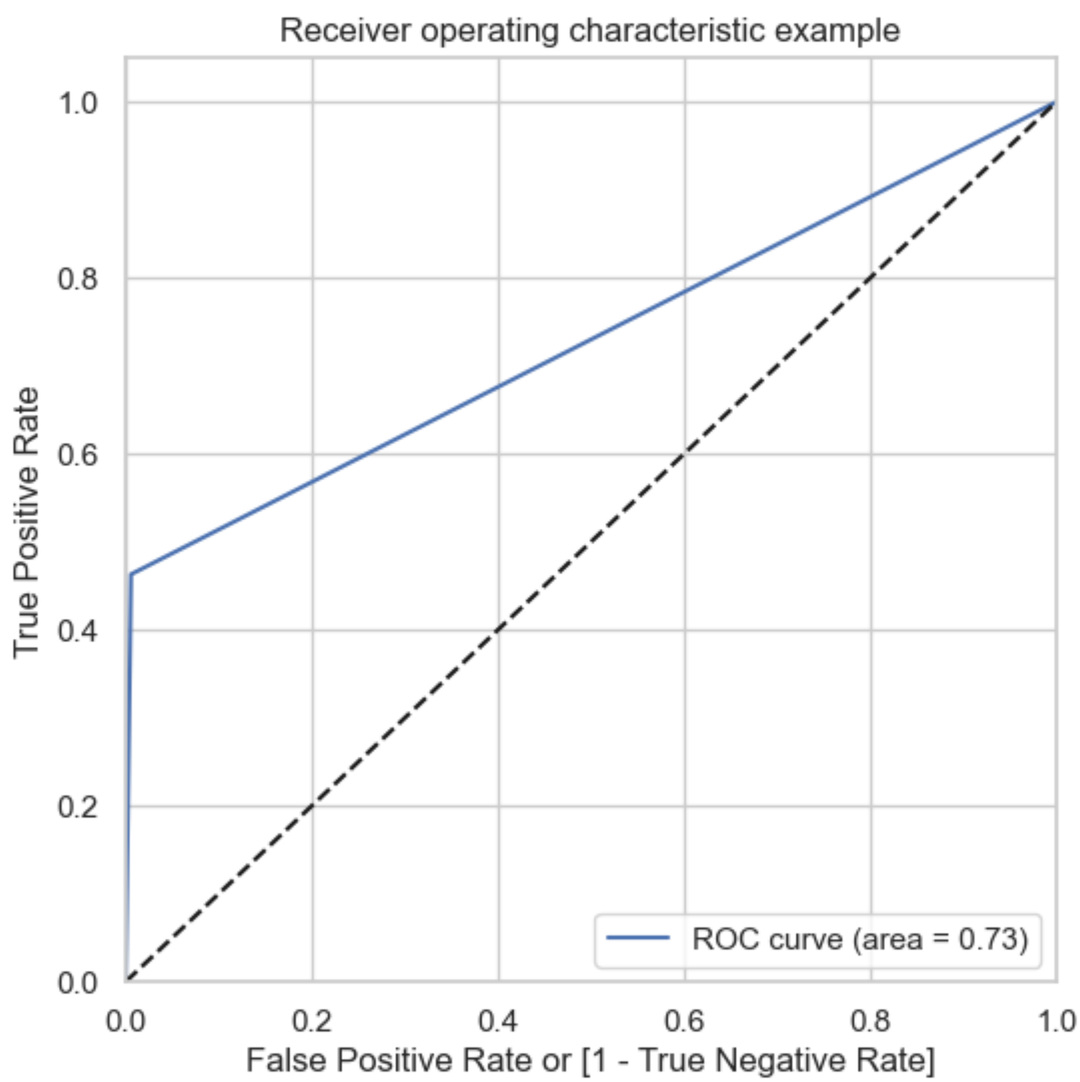
In [175…

```
print(metrics.classification_report(y_test,y_pred_hp_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.94     63711
           1       0.95      0.46      0.62     15495

    accuracy                           0.89     79206
   macro avg       0.92      0.73      0.78     79206
weighted avg       0.90      0.89      0.87     79206
```

In [176…

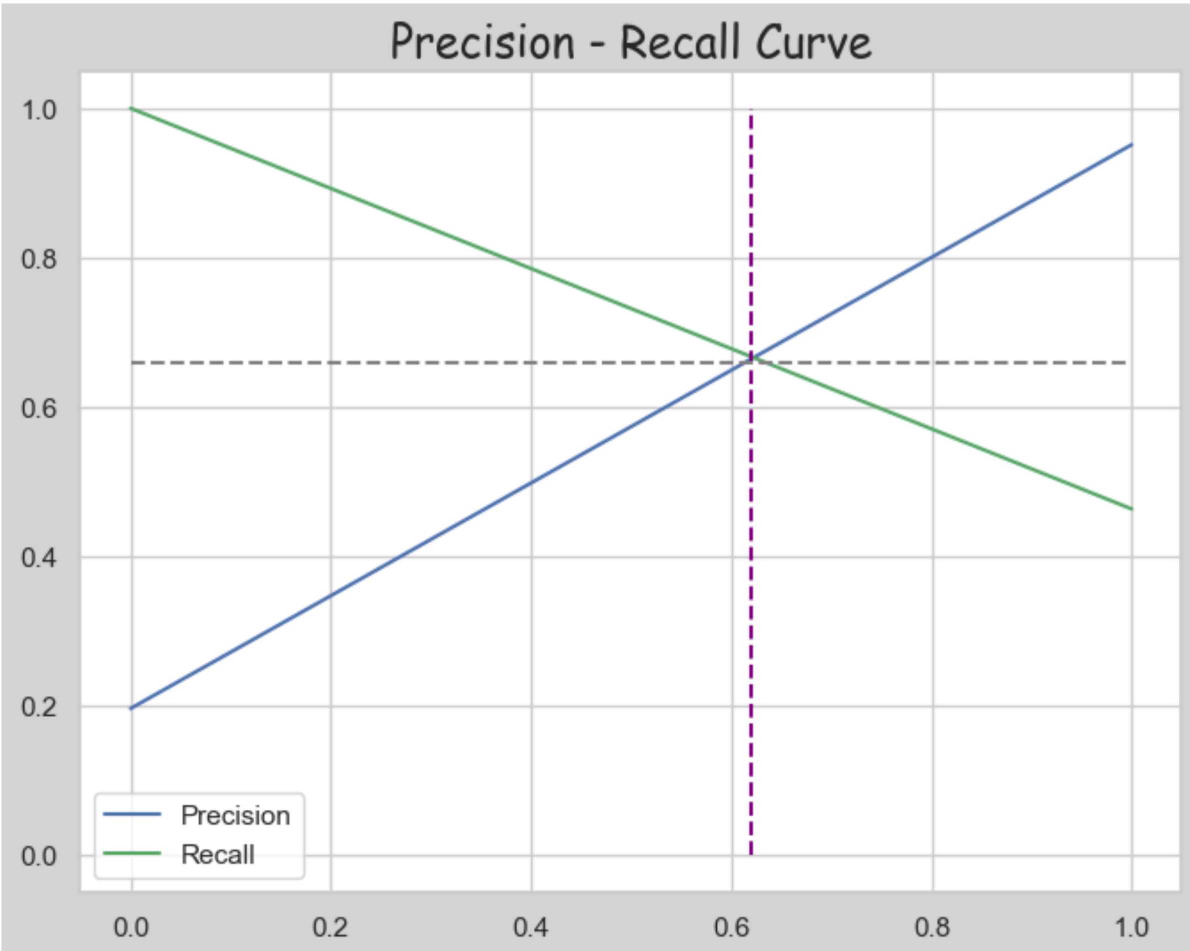
```
draw_roc(y_test, y_pred_hp_test)
```



Out[176]:
```
(array([0.        , 0.00577608, 1.        ]),
 array([0.       , 0.4630526, 1.       ]),
 array([2, 1, 0], dtype=int64))
```

Recall vs Precision

In [177…
```
fig = plt.figure(figsize = (8,6))
fig.set_facecolor("lightgrey")

# Precision Recall Curve
precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pred_hp_te
plt.plot(thresholds, precision[:-1], "b",label='Precision')
plt.plot(thresholds, recall[:-1], "g",label='Recall')
plt.vlines(x=0.62,ymax=1,ymin=0.0,color="purple",linestyles="--")
plt.hlines(y=0.66,xmax=1,xmin=0.0,color="grey",linestyles="--")
plt.title('Precision - Recall Curve',fontsize=18,family = "Comic Sans MS")
plt.legend()
plt.show()
```



In [ ]:

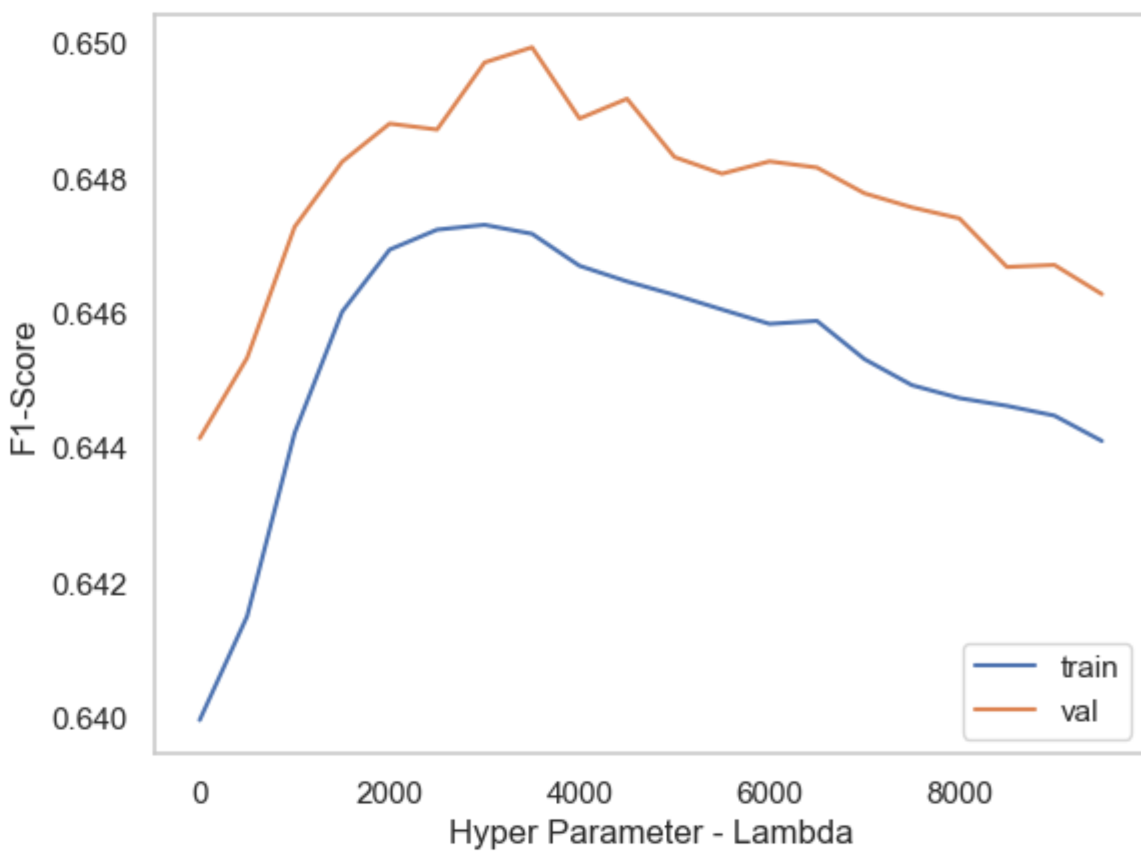
3. Advanced Model with Hyper-parameter, and balancing the data using class weights

In [178…

```
train_scores = []
val_scores = []

la_low = 0.01
la_upp = 10000
la_diff = 500

for lambda_ in np.arange(la_low,la_upp,la_diff):
    hp__clwg_logreg = Pipeline(steps=[('imputer',imputer),
                              ('scaler',scaler),
                              ('logistic_model',LogisticRegression(C=1/lambda_,clas
    hp__clwg_logreg.fit(X_train, y_train)
    train_y_pred = hp__clwg_logreg.predict(X_train)
    val_y_pred = hp__clwg_logreg.predict(X_val)
    train_score = f1_score(y_train, train_y_pred)
    val_score = f1_score(y_val, val_y_pred)
    train_scores.append(train_score)
    val_scores.append(val_score)
```

In [179…

```
plt.figure()
plt.plot(list(np.arange(la_low,la_upp,la_diff)), train_scores, label="train")
plt.plot(list(np.arange(la_low,la_upp,la_diff)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("Hyper Parameter - Lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```



In [180…

```
# Model with lambda_best
best_hp_clwg_model = np.argmax(val_scores)
print(val_scores[best_hp_clwg_model])
```

```
0.6499347071376246
```

In [181…

```
l_best = la_low+la_diff*best_hp_clwg_model
best_hp_clwg_logreg = Pipeline(steps=[('imputer',imputer),
                              ('scaler',scaler),
                              ('logistic_model',LogisticRegression(C=1/l_best,class
                             ])

# Training the Model with Best Hyperparameters:
best_hp_clwg_logreg.fit(X_train, y_train)

# Evaluating the Model on the Test Data:
y_pred_test = best_hp_clwg_logreg.predict(X_test)

# Calculating Various Evaluation Metrics:
test_score = f1_score(y_test, y_pred_test)

print('F1 Score for Best Hyper-Parmeter with class weight Model (Test) ',test_score
```

```
F1 Score for Best Hyper-Parmeter with class weight Model (Test)  0.649209296369806
```

In [182…

```
print(f"Accuracy : {metrics.accuracy_score(y_test, y_pred_test)*100}%")
print(f"recall_score : {metrics.recall_score(y_test, y_pred_test)*100}%")
print(f"precision_score : {metrics.precision_score(y_test, y_pred_test)*100}%")
print(f"f1_score : {metrics.f1_score(y_test, y_pred_test)*100}%")
print(f"AUC score : {metrics.roc_auc_score( y_test, y_pred_test)*100}%")
print(f"confusion_matrix :")
print(metrics.confusion_matrix(y_test, y_pred_test))
```

```
Accuracy : 86.16518950584552%
recall_score : 65.44046466602128%
precision_score : 64.40957886044592%
f1_score : 64.92092963698059%
AUC score : 78.32303247741271%
confusion_matrix :
[[58108  5603]
 [ 5355 10140]]
```

- Accuracy: 86% - In this case, the model is approximately 84.07% accurate in classifying charged offs correctly. - Recall (Sensitivity or True Positive Rate): 65% - A recall of 73.07% means that the model correctly identified about 73.07% of all charged offs. - Precision: 64% - A precision of 57.27% means that out of all the charged offs predicted by the model, about 57.27% were actually correct. - F1 Score: 64% - An F1 score of 64.21% indicates a good balance between precision and recall. - AUC Score (Area Under the Receiver Operating Characteristic Curve): 78% - the model has an AUC score of approximately 79.91%, which suggests that it performs reasonably well in distinguishing between charged offs and full payments.

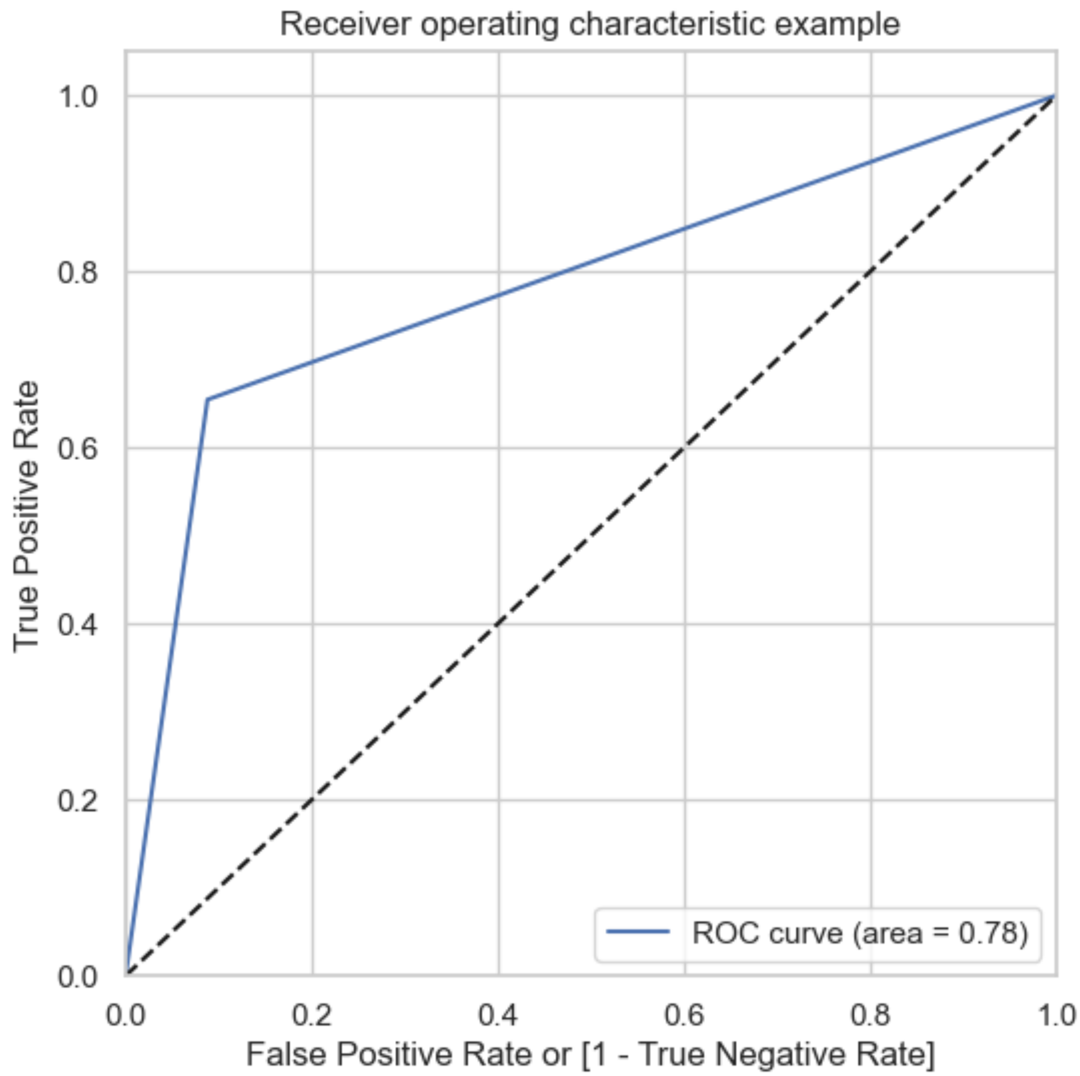
In [183…

```
print(metrics.classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.91      0.91     63711
           1       0.64      0.65      0.65     15495

    accuracy                           0.86     79206
   macro avg       0.78      0.78      0.78     79206
weighted avg       0.86      0.86      0.86     79206
```

In [184…

```
draw_roc(y_test, y_pred_test)
```



Out[184]:

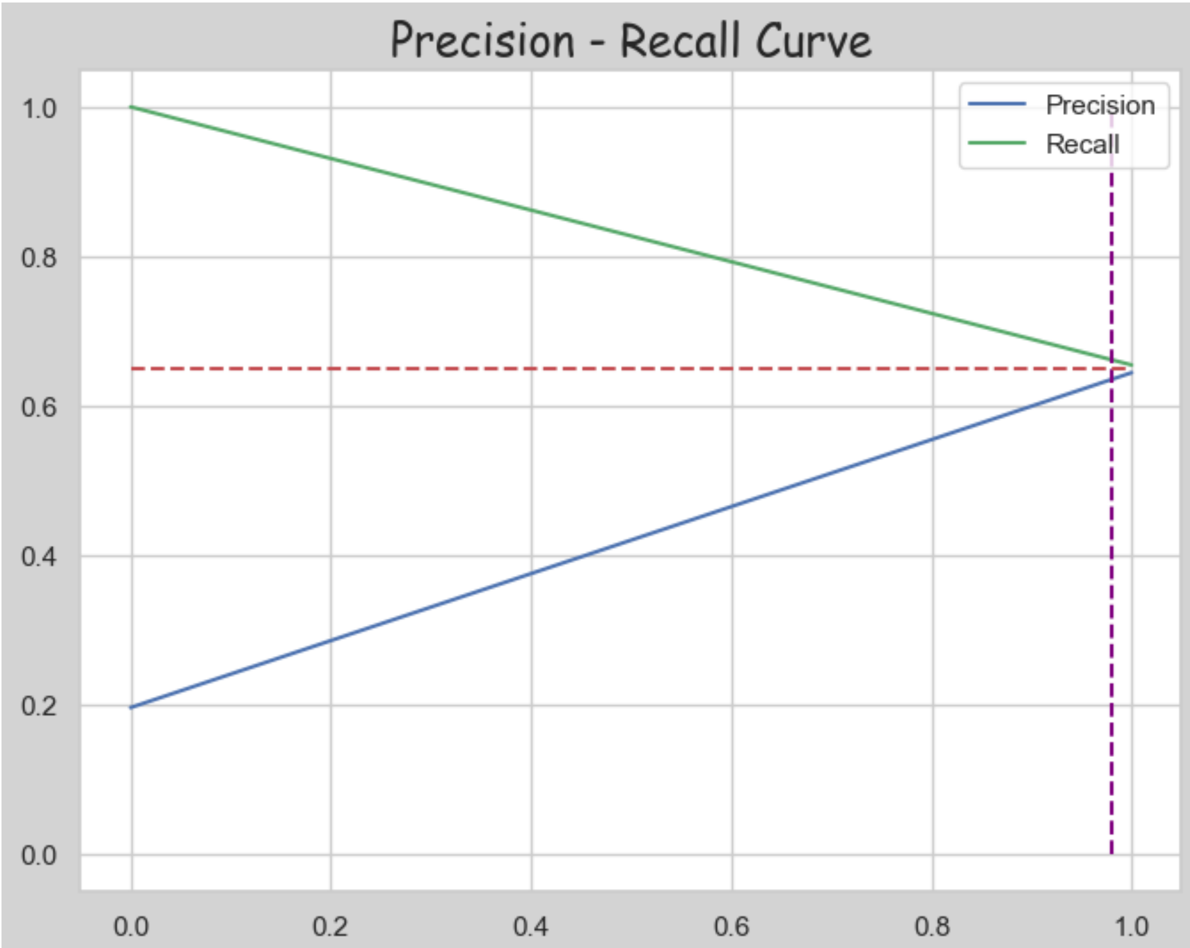
```
(array([0.       , 0.087944, 1.       ]),
 array([0.        , 0.65440465, 1.        ]),
 array([2, 1, 0], dtype=int64))
```

Array 1: False Positive Rate (FPR) Values [0.0, 0.087944, 1.0] Array 2: True Positive Rate (TPR) or Recall Values [0.0, 0.65440465, 1.0] Array 3: Thresholds [2, 1, 0] These values represent the threshold levels used for classification. The first value, 2, corresponds to the lowest threshold (where almost everything is classified as class 0). The second value, 1, corresponds to the threshold where there's a balance between true positives and false positives. The third value, 0, corresponds to the highest threshold (where almost everything is classified as class 1).the ROC curve and its associated data help you assess the trade-off between true positive rate and false positive rate for different classification thresholds, allowing you to choose a threshold that suits your specific application and balance between sensitivity and specificity.

Recall vs Precision

In [185…

```
fig = plt.figure(figsize = (8,6))
fig.set_facecolor("lightgrey")

# Precision Recall Curve
precision, recall, thresholds = metrics.precision_recall_curve(y_test, y_pred_test)
plt.plot(thresholds, precision[:-1], "b",label='Precision')
plt.plot(thresholds, recall[:-1], "g",label='Recall')
plt.vlines(x=0.98,ymax=1,ymin=0.0,color="purple",linestyles="--")
plt.hlines(y=0.65,xmax=1,xmin=0.0,color="r",linestyles="--")
plt.title('Precision - Recall Curve',fontsize=18,family = "Comic Sans MS")
plt.legend()
plt.show()
```



Top 5 features that played key role in getting charged-off or not

- Used RFE technique

In [186…

```
X_train['revol_util'] = X_train['revol_util'].fillna(X_train['revol_util'].median()
```

In [187…

```
rfe = RFE(best_hp_clwg_logreg['logistic_model'], n_features_to_select=15)
rfe = rfe.fit(X_train, y_train)
```

In [188…

```
cols=X_train.columns[rfe.support_]
cols
```

Out[188]:

```
Index(['int_rate', 'home_ownership_RENT',
       'verification_status_Source Verified', 'verification_status_Verified',
       'purpose_debt_consolidation', 'initial_list_status_w', 'zipcode_05113',
       'zipcode_11650', 'zipcode_29597', 'zipcode_86630', 'zipcode_93700',
       'dti_cat_10-20', 'dti_cat_20-30', 'dti_cat_  Above 30',
       'mort_acc_cat_1.0'],
      dtype='object')
```

In [189…

```
#Function to fit the logistic regression model from the statmodel package
def fit_LogRegModel(X_train):
    # Adding a constant variable
    X_train = sm.add_constant(X_train)
    lm = sm.GLM(y_train,X_train,family = sm.families.Binomial()).fit()
    print(lm.summary())
    return lm
```

In [190…

```
# Calculate the VIFs for the new model
def getVIF(X_train):
    vif = pd.DataFrame()
    X = X_train
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return(vif)
```

Assessing the Model using StatsModels

In [191…

```
# Creating X_test dataframe with RFE selected variables
X_train_GM = X_train[cols]
lm = fit_LogRegModel(X_train_GM)
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            loan_status   No. Observations:               237618
Model:                            GLM   Df Residuals:                   237602
Model Family:                Binomial   Df Model:                           15
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -67592.
Date:                Sat, 30 Sep 2023   Deviance:                   1.3518e+05
Time:                        20:18:50   Pearson chi2:                 1.60e+05
No. Iterations:                    30   Pseudo R-squ. (CS):             0.3438
Covariance Type:            nonrobust
==================================================================================
                                         coef    std err          z      P>|z|
----------------------------------------------------------------------------------
const                                 -4.0360      0.033   -120.495      0.000
int_rate                               0.1284      0.002     80.457      0.000
home_ownership_RENT                    0.1863      0.017     10.987      0.000
verification_status_Source Verified    0.2830      0.018     15.520      0.000
verification_status_Verified           0.1637      0.018      8.888      0.000
purpose_debt_consolidation             0.0711      0.014      4.937      0.000
initial_list_status_w                  0.1095      0.014      7.646      0.000
zipcode_05113                        -29.8620   2.53e+04     -0.001      0.999     -
zipcode_11650                         33.1020   5.09e+04      0.001      0.999     -
zipcode_29597                        -29.8701   2.53e+04     -0.001      0.999     -
zipcode_86630                         33.1015   5.21e+04      0.001      0.999     -
zipcode_93700                         33.1083   5.16e+04      0.001      0.999     -
dti_cat_10-20                          0.2155      0.021     10.355      0.000
dti_cat_20-30                          0.4802      0.022     22.322      0.000
dti_cat_  Above 30                     0.7280      0.028     25.640      0.000
mort_acc_cat_1.0                      -0.1498      0.017     -8.800      0.000
==================================================================================
```

In [192…

```
X_train_GM = X_train_GM.drop(['zipcode_05113','zipcode_86630','zipcode_93700','zipc
```

In [193…

```
lm = fit_LogRegModel(X_train_GM)
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            loan_status   No. Observations:               237618
Model:                            GLM   Df Residuals:                   237607
Model Family:                Binomial   Df Model:                           10
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:            -1.0886e+05
Date:                Sat, 30 Sep 2023   Deviance:                   2.1772e+05
Time:                        20:18:51   Pearson chi2:                 2.34e+05
No. Iterations:                     5   Pseudo R-squ. (CS):            0.07131
Covariance Type:            nonrobust
===============================================================================
                                          coef    std err          z      P>|z|
-------------------------------------------------------------------------------
const                                  -3.8157      0.026   -146.838      0.000
int_rate                                0.1289      0.001    103.034      0.000
home_ownership_RENT                     0.1894      0.013     14.406      0.000
verification_status_Source Verified     0.2704      0.014     19.138      0.000
verification_status_Verified            0.1642      0.014     11.544      0.000
purpose_debt_consolidation              0.0494      0.011      4.427      0.000
initial_list_status_w                   0.1036      0.011      9.306      0.000
dti_cat_10-20                           0.2145      0.016     13.309      0.000
dti_cat_20-30                           0.5038      0.017     30.282      0.000
dti_cat_  Above 30                      0.7677      0.022     34.836      0.000
mort_acc_cat_1.0                       -0.1518      0.013    -11.502      0.000
===============================================================================
```

In [194…

```python
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train_GM)), family = sm.families.Binomial
res = logm1.fit()
res.summary()
```

Out[194]:

Generalized Linear Model Regression Results

Dep. Variable:	loan_status	**No. Observations:**	237618
Model:	GLM	**Df Residuals:**	237607
Model Family:	Binomial	**Df Model:**	10
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-1.0886e+05
Date:	Sat, 30 Sep 2023	**Deviance:**	2.1772e+05
Time:	20:18:52	**Pearson chi2:**	2.34e+05
No. Iterations:	5	**Pseudo R-squ. (CS):**	0.07131
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.8157	0.026	-146.838	0.000	-3.867	-3.765
int_rate	0.1289	0.001	103.034	0.000	0.126	0.131
home_ownership_RENT	0.1894	0.013	14.406	0.000	0.164	0.215
verification_status_Source Verified	0.2704	0.014	19.138	0.000	0.243	0.298
verification_status_Verified	0.1642	0.014	11.544	0.000	0.136	0.192
purpose_debt_consolidation	0.0494	0.011	4.427	0.000	0.028	0.071
initial_list_status_w	0.1036	0.011	9.306	0.000	0.082	0.125
dti_cat_10-20	0.2145	0.016	13.309	0.000	0.183	0.246
dti_cat_20-30	0.5038	0.017	30.282	0.000	0.471	0.536
dti_cat_ Above 30	0.7677	0.022	34.836	0.000	0.725	0.811
mort_acc_cat_1.0	-0.1518	0.013	-11.502	0.000	-0.178	-0.126

In [195…

```
# Make a VIF dataframe for all the variables present
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['Features'] = X_train_GM.columns
vif['VIF'] = [variance_inflation_factor(X_train_GM.values, i) for i in range(X_trai
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[195]:

	Features	VIF
0	int_rate	7.16
9	mort_acc_cat_1.0	2.87
6	dti_cat_10-20	2.79
4	purpose_debt_consolidation	2.43
7	dti_cat_20-30	2.31
3	verification_status_Verified	2.26
1	home_ownership_RENT	2.17
2	verification_status_Source Verified	2.06
5	initial_list_status_w	1.68
8	dti_cat_ Above 30	1.39

Inferences:

- Key features that heavily affected the outcome are -
  - dti, mort_acc, verification_status, sub_grade & int_rate

Confusion Metrics w.r.t. Lending Club Loan

	Fully Paid (0)	**Charged off (1)**
Charged off (1)	TN	FP
Fully Paid (0)	FN	TP

Which metrics we should select for our model will depend on the Business use case

Case 1 - When the bank does not want to lose the money as well as the customers. 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 supply chains and earn interest on it.

- In case of low recall, Lending Club loan might lose money. Low precision means even if the borrower is not a defaulter or charged off, he will not be approved for a loan. That means lost business for the banks. It is important to have a balance between recall and precision, so a good F1-score will make sure that balance is maintained.

Case 2 - The bank does not want to lose the money but can grow slowly with genuine customers. 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 with NPA.

- In this case, when predicting whether or not a loan will default - it would be better to have a high recall because the banks don't want to lose money, so it would be a good idea to alert the bank even if there is a slight doubt about the borrower.Low precision, in this case, might be okay.

Case 3: When a bank wants to grow faster and get more customers at the expense of losing some money in some cases.

- In this case, it would be ok to have a slight higher precision compare the recall.

Comparison between Model 3 & Model 2

	Model 3 (Hyper-param & Balanced Data)	**Model 2 (Hyper-param)**
Accuracy	86	89
Recall	65	46
Precision	64	94
F1 Score	65	62
AUC Score	78	72

Inferences

- From the above metrics it is clearly shows Model 3 is much better than Model 2 as balance between recall and precision is maintained.
- A low recall or precision (one or both inputs) makes the F1-score more sensitive, which is great if you want to balance the two.The higher the F1-score the better the model for case 1
- Model 3 has F1-score as 65 where as Model 2 has F-score as 62only.
- Moreover, we can clearly see that recall is very high for models with balanced data. In our case it it Model 3.

Inferences and Recommendations

Inferences Based on EDA:

- Eighty-five percent of loan balances are fully paid, while 19 percent have been charged off
- There is a strong correlation between loan amount and installment (with 0.95)
- Mortgages are the most common form of home ownership
- 94% of people who have grades 'A' pay their loans on time.
- The two top job titles that take most loans are teacher and manager.
- zip codes 11650,86630, and 93700 have a 100% probability of getting charged-off. Location plays imprtant role for loan getting charged-off.

Inferences based on the modlel:

- From the above metrics it is clearly shows Model 3 is much better than Model 2 as balance between recall and precision is maintained.
- A low recall or precision (one or both inputs) makes the F1-score more sensitive, which is great if you want to balance the two.The higher the F1-score the better the model for case 1
- Model 3 has F1-score as 65 where as Model 2 has F-score as 62only.
- Moreover, we can clearly see that recall is very high for models with balanced data. In our case it it Model 3.

Recommendations:

- Model 3 is recommended as it can detect real defaulters and ensure that the bank will not lose the opportunity to finance more supply chains and earn interest.
- One way to make sure we have fewer defaulters is to get customers with high grades.
- zip codes 11650,86630, and 93700 have a 100% probability of getting charged-off. Banks should refrain from lending to these areas until they understand why. As well, setup a team to analyze, as this is a common trend for getting charged-off at those locations.
- Key features that heavily affected the outcome are - dti, mort_acc, verification_status, sub_grade & int_rate

In [ ]: