Business Case - Jai Kisan Neo Bank - Logistic Regression

Problem Statement

- Help **Jai Kisan Neo Bank** to determine whether to **extend a credit line to a business** based on the attributes for individuals or borrowers.
- If so, **what should be the repayment terms?**

About Jai Kisan Neo Bank

- Jai Kisan Neo Bank is a rural-focused fintech that aims to bridge the **credit gap in the rural market.** Currently, 80% of rural individuals and businesses find it difficult to access formal credit.
- Jai Kisan Neo bank is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.

Dataset - Jai kisan Neo Bank

Column Profiling

- loan_amnt : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
- term : The number of payments on the loan. Values are in months and can be either 36 or 60.
- int_rate : Interest Rate on the loan
- installment : The monthly payment owed by the borrower if the loan originates.
- grade : JaiKisan assigned loan grade
- sub_grade : JaiKisan assigned loan subgrade
- emp_title :The job title supplied by the Borrower when applying for the loan.*
- emp_length : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.
- annual_inc : The self-reported annual income provided by the borrower during registration. verification_status : Indicates if income was verified by JaiKisan, not verified, or if the income source was verified
- issue_d : The month which the loan was funded
- loan_status : Current status of the loan - Target Variable
- purpose : A category provided by the borrower for the loan request.
- title : The loan title provided by the borrower
- dti : A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested JaiKisan loan, divided by the borrower’s self-reported monthly income.
- earliest_cr_line :The month the borrower's earliest reported credit line was opened
- open_acc : The number of open credit lines in the borrower's credit file.

- pub_rec : Number of derogatory public records
- revol_bal : Total credit revolving balance
- revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- total_acc : The total number of credit lines currently in the borrower's credit file
- initial_list_status : The initial listing status of the loan. Possible values are – W, F
- application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
- mort_acc : Number of mortgage accounts.
- pub_rec_bankruptcies : Number of public record bankruptcies
- Address- Address of the borrower

In [889…

```
#Importing packages
import numpy as np
import pandas as pd

# Importing matplotlib and seaborn for graphs
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid')

import warnings
warnings.filterwarnings('ignore')

from scipy import stats
```

Utility Functions - Used during Analysis

Missing Value - Calculator

In [890…

```
def missingValue(df):
    #Identifying Missing data. Already verified above. To be sure again checking.
    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 Pe
    return md
```

Numerical Variable Analysis

- box plot
- distplot

In [891…

```
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 [892…

```
# 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='#56B4

    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 [893…

```
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()
```

Source - https://towardsdatascience.com/100-stacked-charts-in-python-6ca3e1962d2b

In [894…

```
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()
```

Loading and inspecting the Dataset

Loading the csv file

In [895…

```
loan_data = pd.read_csv("./jai_kisan_logistic_regression.csv")
```

In [896…

```
loan_data.head()
```

Out[896…

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ow
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	MO
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	MO

5 rows × 27 columns

Validating Duplicate Records

In [897…

```
loan_data.shape
```

Out[897…

```
(396030, 27)
```

In [898…

```
loan_data.columns
```

Out[898…

```
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 [899…

```
loan_data.duplicated().sum()
```

Out[899…

```
0
```

In [900…

```
missingValue(loan_data).head(7)
```

```
Total records =  396030
```

Out[900…

	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 [901…

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

Out[901…

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

In [902…

```
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
```

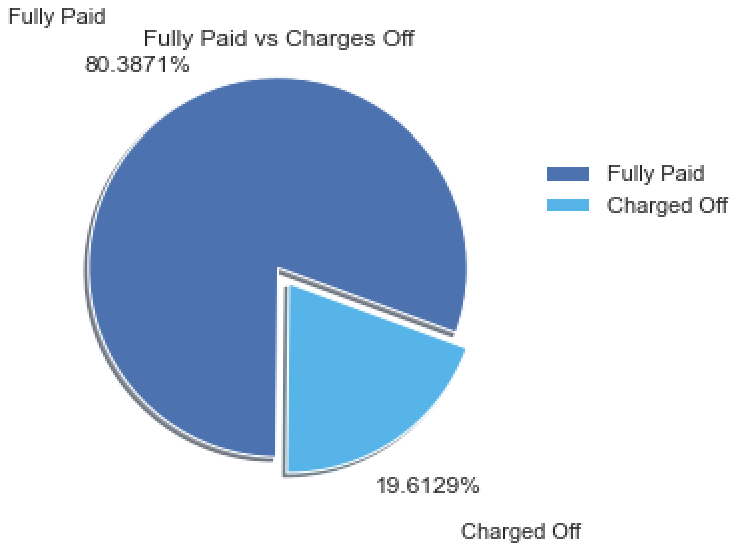
Target variable Analysis

In [903…

```
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[903…

```
<matplotlib.legend.Legend at 0x29423448860>
```



Inference

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

Pre-Processing & EDA

Numerical Variables

loan_amnt

In [904…

```
loan_data[['loan_amnt']].describe().T
```

Out[904…

	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 [905…

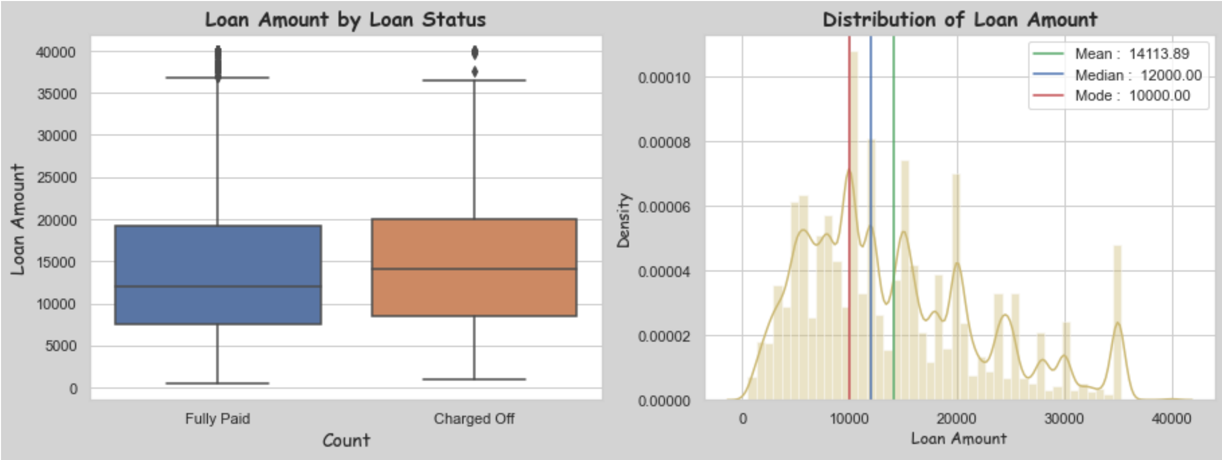
```
loan_data.groupby(['loan_status'])['loan_amnt'].describe()
```

Out[905…

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

In [906…

```
plot_num_var(loan_data,'loan_amnt','Loan Amount')
```



Inference

- Medain Loan Amount is 14113
- Charged-offs have a higher loan amount than fully paid with a mean loan amount of 13866 & 15126, respectively.

Interest Rate

In [907…

```
loan_data[['int_rate']].describe().T
```

Out[907…

	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 [908…

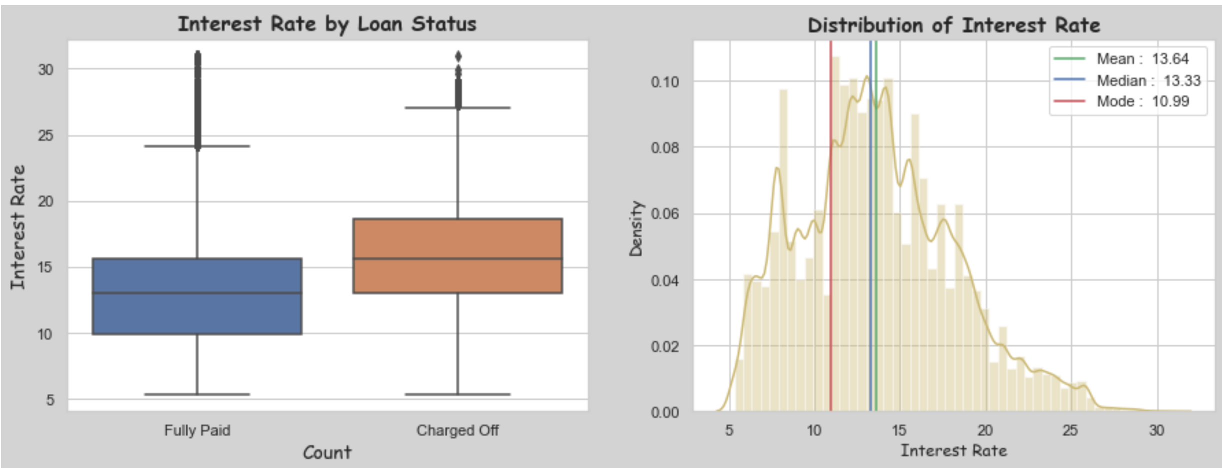
```
loan_data.groupby(['loan_status'])['int_rate'].describe()
```

Out[908…

	count	mean	std	min	25%	50%	75%	max
loan_status								
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 [909…

```
plot_num_var(loan_data,'int_rate','Interest Rate')
```



Inference

- Medain 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.

Installement

In [910…

```
loan_data[['installment']].describe().T
```

Out[910…

	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 [911…

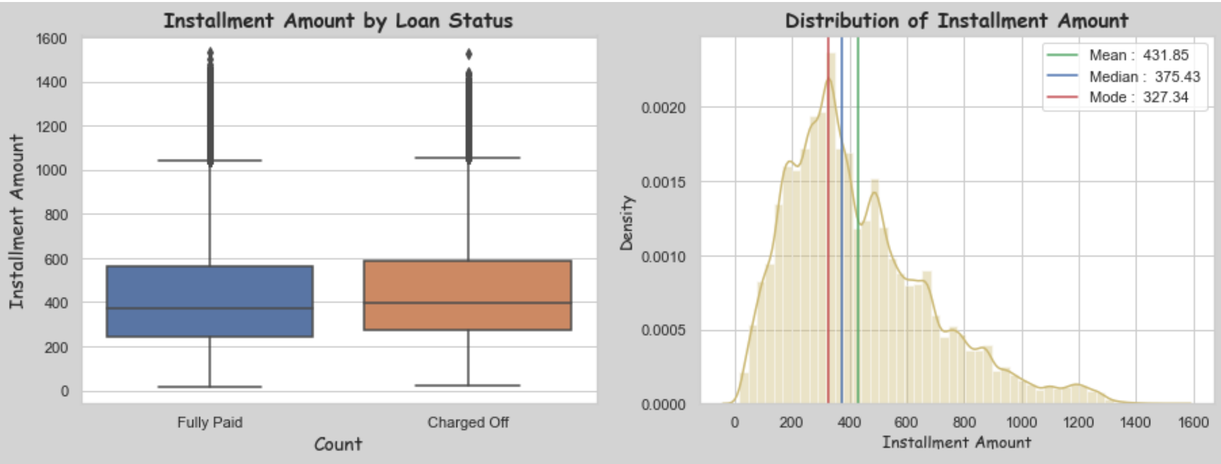
```
loan_data.groupby(['loan_status'])['installment'].describe()
```

Out[911…

	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 [912…

```
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 [913…

```
loan_data[['annual_inc']].describe().T
```

Out[913…

	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 [914…

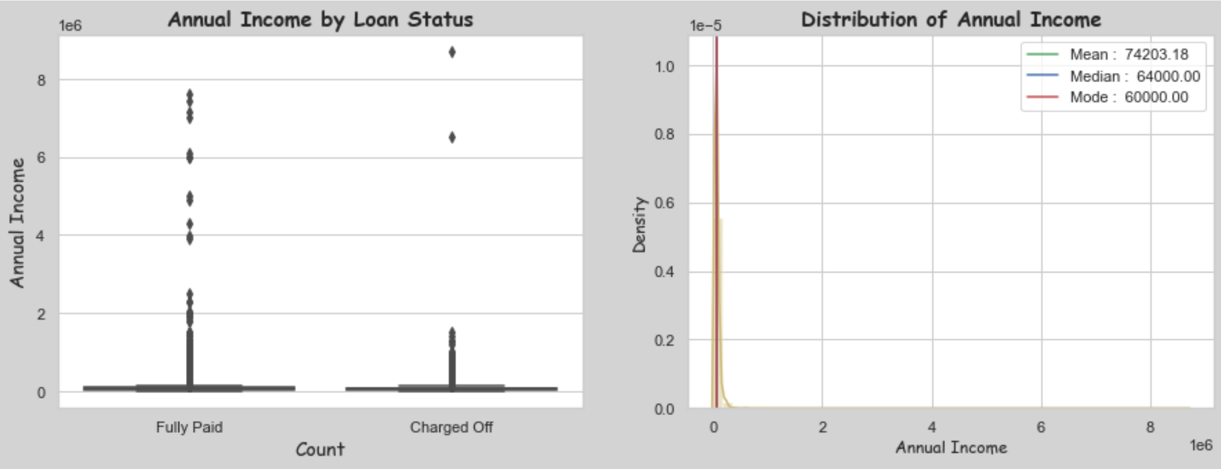
```
loan_data.groupby(['loan_status'])['annual_inc'].describe()
```

Out[914...

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 [915...

```
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 [916...

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

In [917...

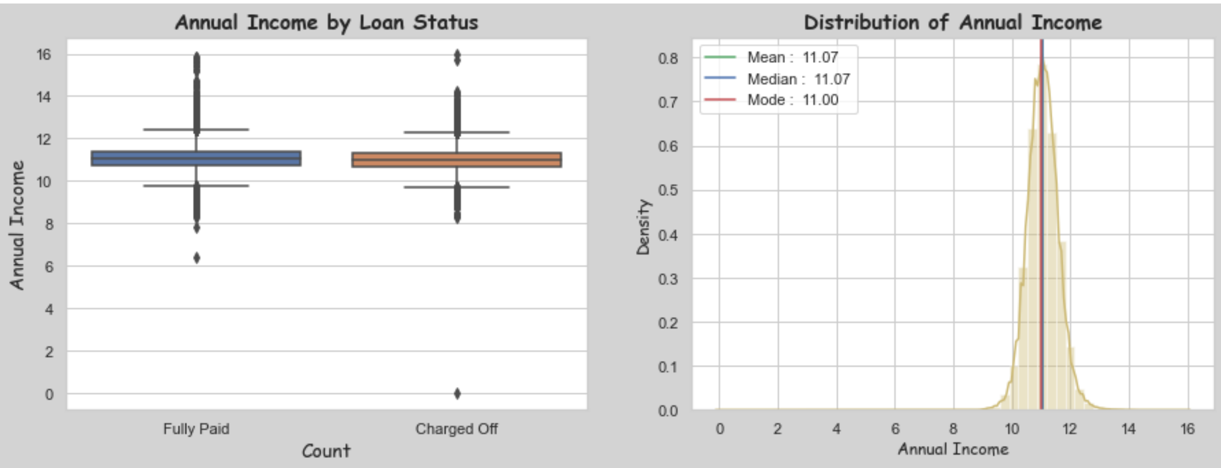
```
loan_data[['annual_inc_ln']].describe().T
```

Out[917...

	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 [918...

```
plot_num_var(loan_data,'annual_inc_ln','Annual Income')
```



In [919…
```
loan_data.groupby(['loan_status'])['annual_inc_ln'].describe()
```
Out[919…

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 [920…
```
77673/loan_data.shape[0]
```
Out[920…
```
0.1961290811302174
```
In [921…
```
318357/loan_data.shape[0]
```
Out[921…
```
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 [922…
```
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 JaiKisan loan, divided by the borrower's self-reported monthly income.

In [923…
```
loan_data[['dti']].describe().T
```
Out[923…

	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

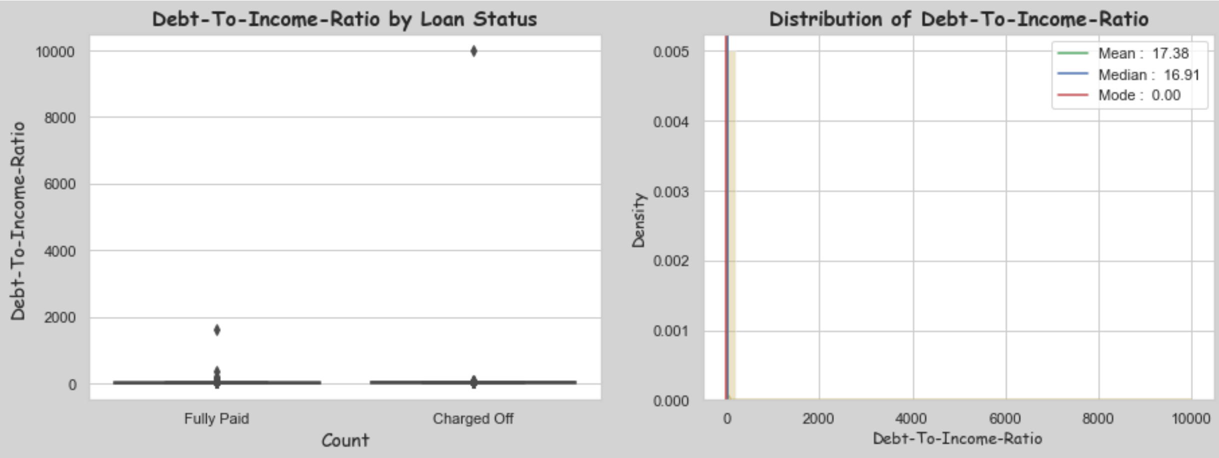
In [924…
```
loan_data.groupby(['loan_status'])['dti'].describe()
```
Out[924…

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	19.656346	36.781068	0.0	13.33	19.34	25.55	9999.0

	count	mean	std	min	25%	50%	75%	max
loan_status								
Fully Paid	318357.0	16.824010	8.500979	0.0	10.87	16.34	22.29	1622.0

In [925...

```
plot_num_var(loan_data,'dti','Debt-To-Income-Ratio')
```



In [926...

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

Out[926...

```
Fully Paid     26
Charged Off     9
Name: loan_status, dtype: int64
```

In [927...

```
9/35
```

Out[927...

```
0.2571428571428571
```

In [928...

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

Out[928...

```
Fully Paid     68242
Charged Off    10850
Name: loan_status, dtype: int64
```

In [929...

```
10850/(68242+10850)
```

Out[929...

```
0.13718201588024073
```

Inferences

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

Open Credit lines

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

In [930...

```
loan_data[['open_acc']].describe().T
```

Out[930...

	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 [931…

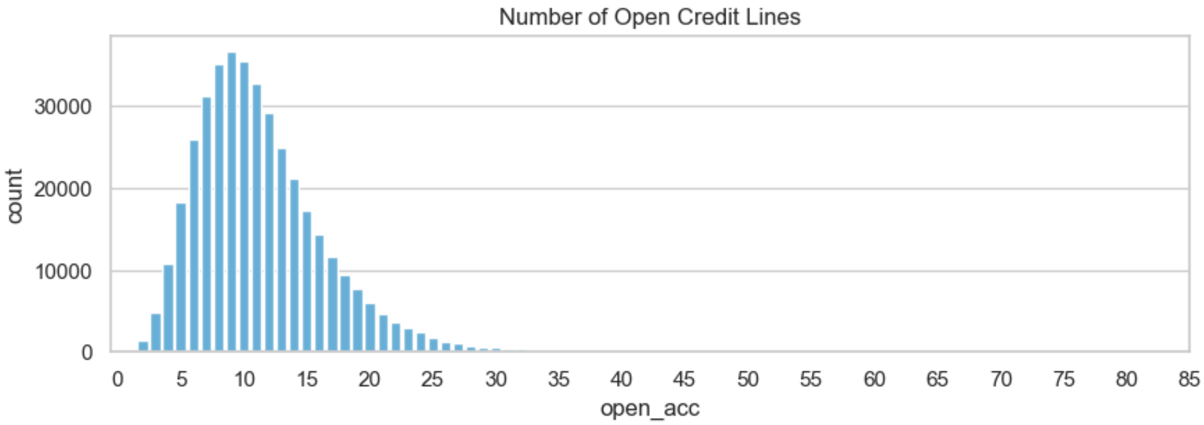
```
loan_data['open_acc'].nunique()
```

Out[931…

```
61
```

In [932…

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



Public record (pub_rec)

- Number of derogatory public records

In [933…

```
loan_data[['pub_rec']].describe().T
```

Out[933…

	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 [934…

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

Out[934…

```
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 [935…

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

Out[935…

```
Fully Paid     45424
Charged Off    12334
Name: loan_status, dtype: int64
```

In [936…

```
12334/(12334+45424)
```

Out[936…

```
0.21354617542158663
```

In [937…

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

Out[937…

```
Fully Paid     1932
Charged Off     611
Name: loan_status, dtype: int64
```

In [938…

```
611/(611+1932)
```

Out[938…

```
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 [939…

```
loan_data['revol_bal'].nunique()
```

Out[939…

```
55622
```

In [940…

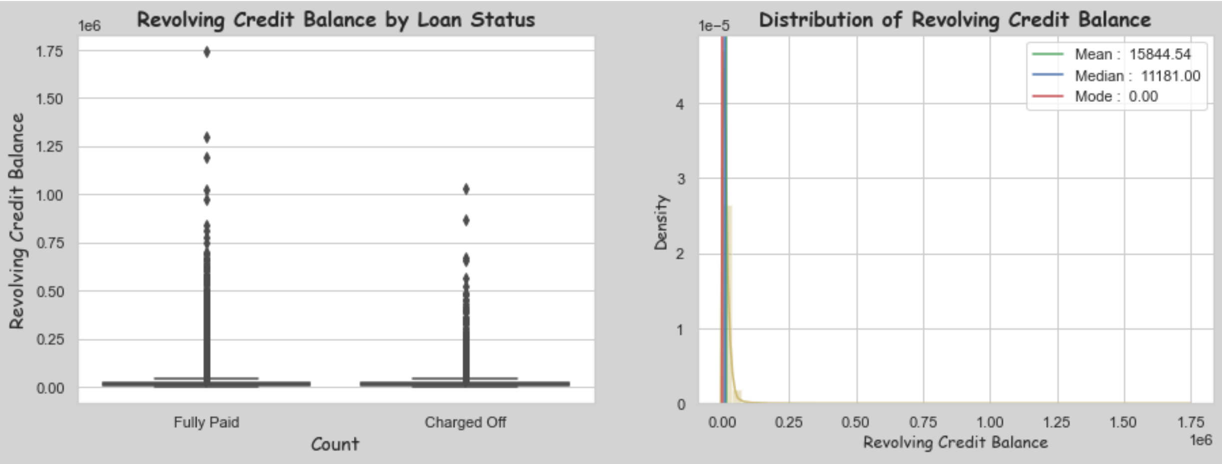
```
loan_data[['revol_bal']].describe().T
```

Out[940…

	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 [941…

```
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.

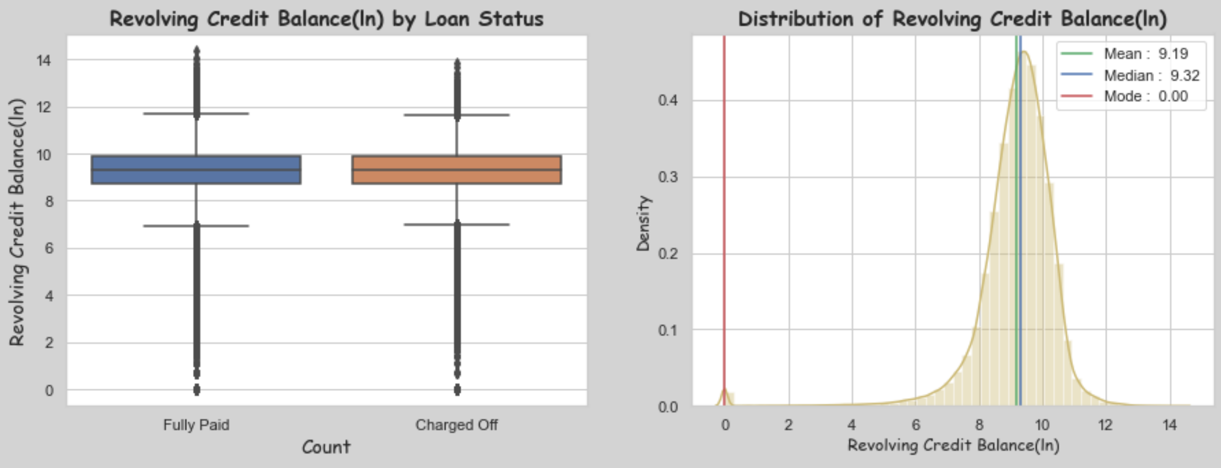
In [942…

```
## trainsforming target variable using numpy.log1p,
```

```
loan_data["revol_bal_ln"] = np.log1p(loan_data["revol_bal"])
```

In [943...

```
plot_num_var(loan_data,'revol_bal_ln','Revolving Credit Balance(ln)')
```



In [944...

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

Out[944...

	count	mean	std	min	25%	50%	75%	max
loan_status								
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 [945...

```
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 [946...

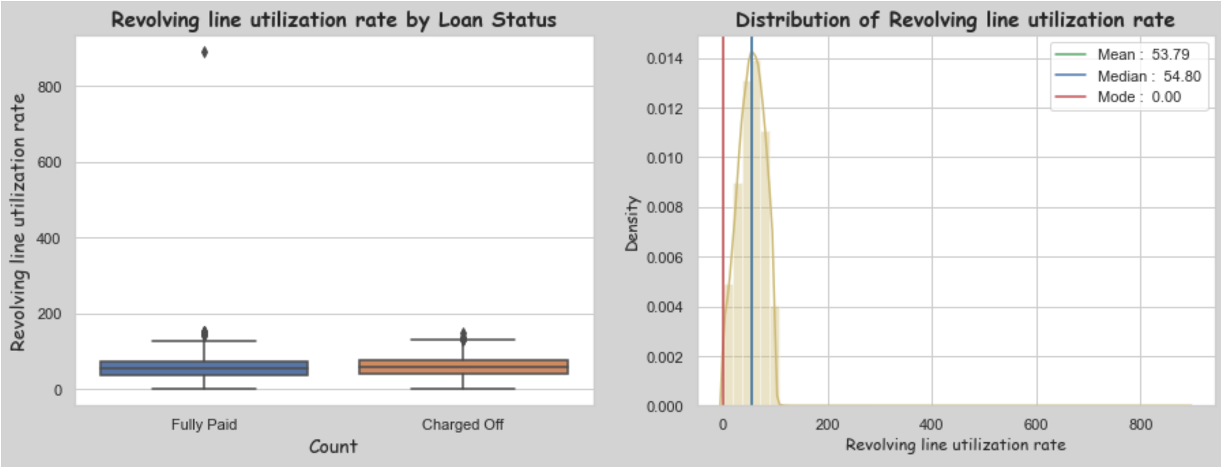
```
loan_data[['revol_util']].describe().T
```

Out[946...

	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 [947...

```
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 [948…

```
loan_data[['total_acc']].describe().T
```

Out[948…

	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 [949…

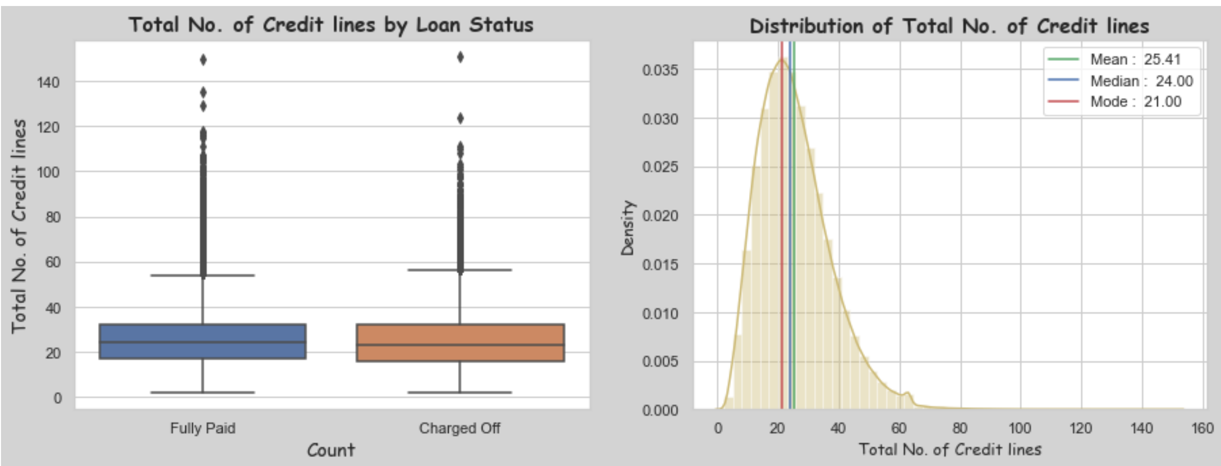
```
loan_data.groupby(['loan_status'])['total_acc'].describe()
```

Out[949…

	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 [950…

```
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 [951...

```
loan_data[['mort_acc']].describe().T
```

Out[951...

	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 [952...

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

Out[952...

	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 [953...

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

Out[953...

```
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 [954...

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

Out[954...

```
Fully Paid     1797
Charged Off     269
Name: loan_status, dtype: int64
```

In [955...

```
269/(1797+269)
```

Out[955...

```
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 [956…
```
loan_data['pub_rec_bankruptcies'].value_counts().sort_index()
```
Out[956…
```
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 [957…
```
loan_data.loc[loan_data['pub_rec_bankruptcies']>=1, 'loan_status'].value_counts()
```
Out[957…
```
Fully Paid     35850
Charged Off     9265
Name: loan_status, dtype: int64
```
In [958…
```
9265/(9265+35850)
```
Out[958…
```
0.20536406959991133
```

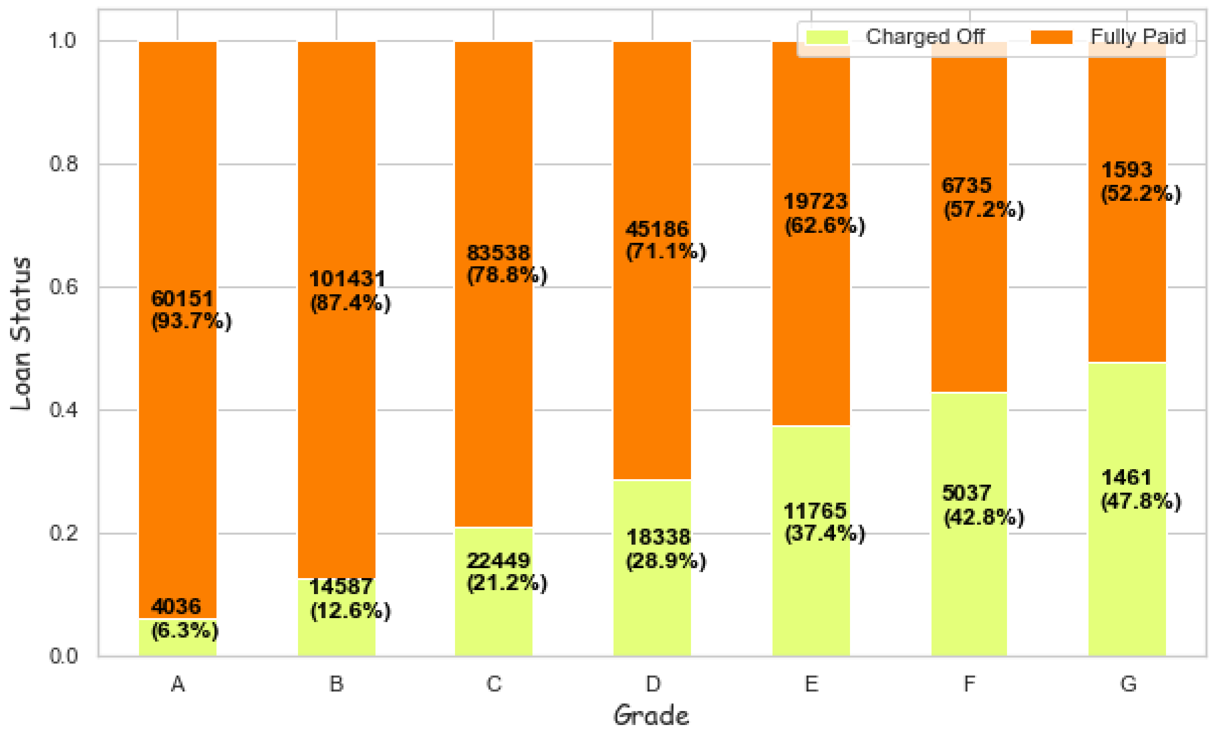
Inferences

- 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 [959…
```
print(sorted(loan_data['grade'].unique()))
```
```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```
In [960…
```
stack_bar(loan_data,'grade',"Grade")
```



In [961…

```
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', 'F3',
'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5']
```

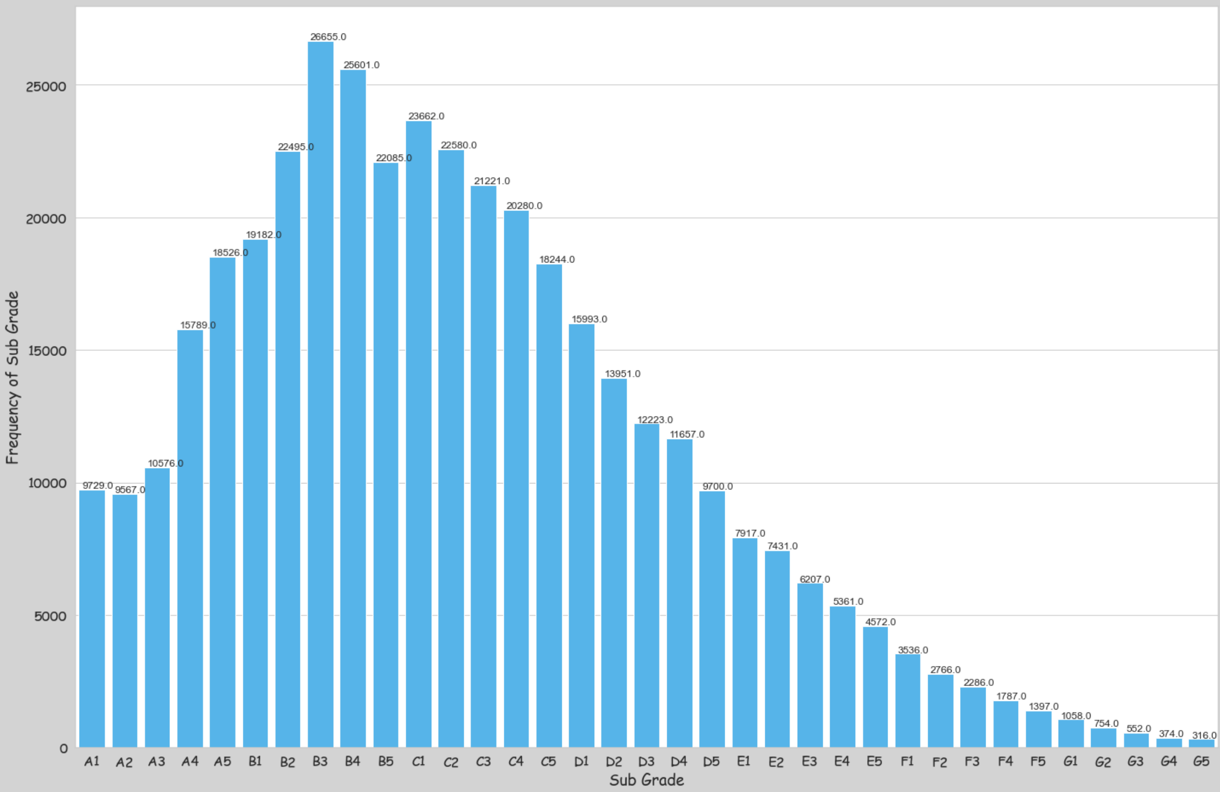
In [962…

```
stack_bar_h(loan_data,'sub_grade',"Sub Grade")
```


Loan Status
Sub Grade
Charged Off
Fully Paid
0.0
0.2
0.4
0.6
0.8
1.0
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%)
A1 279 (2.9%) 9450 (97.1%)

In [963...

```
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 [964...

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

Term

In [965...

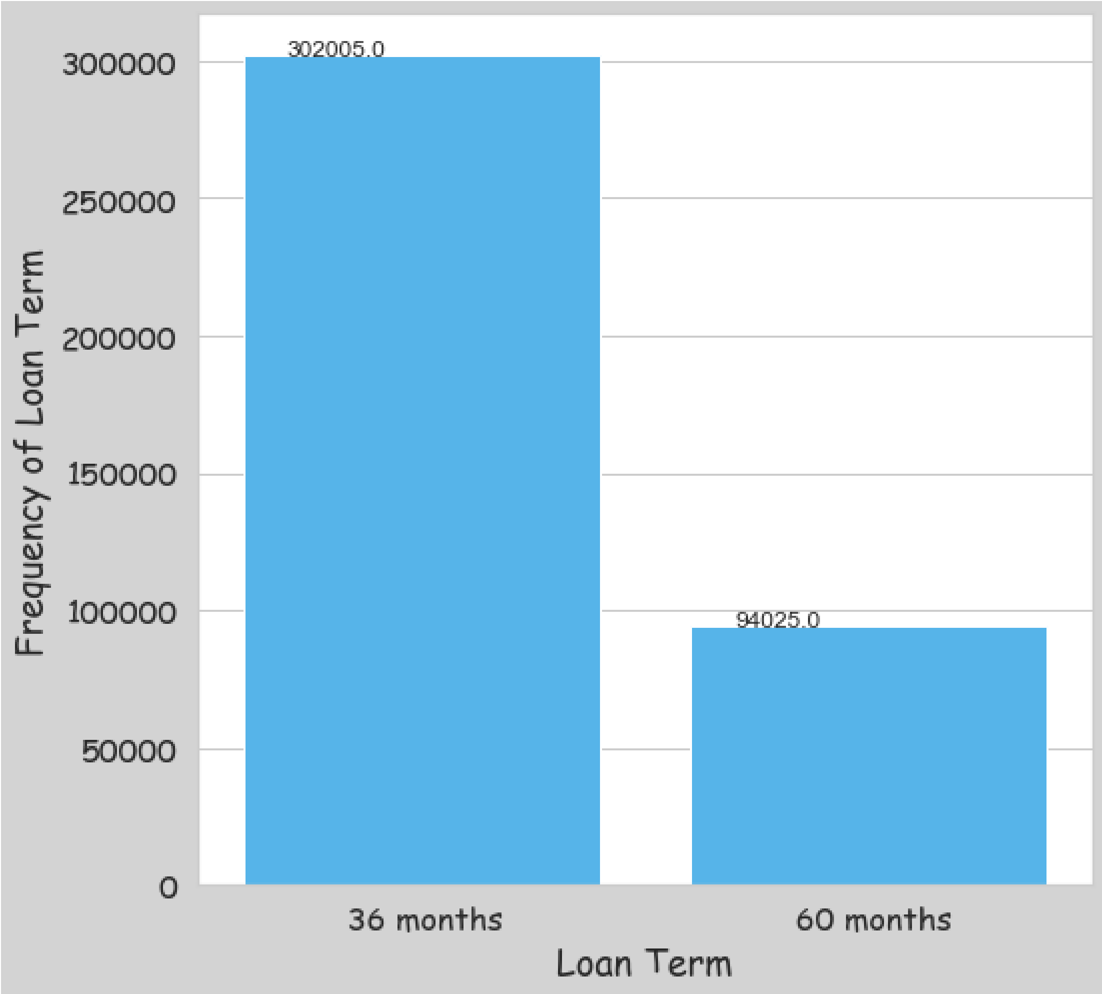
```
loan_data['term'].value_counts()
```

Out[965...

```
 36 months    302005
 60 months     94025
Name: term, dtype: int64
```

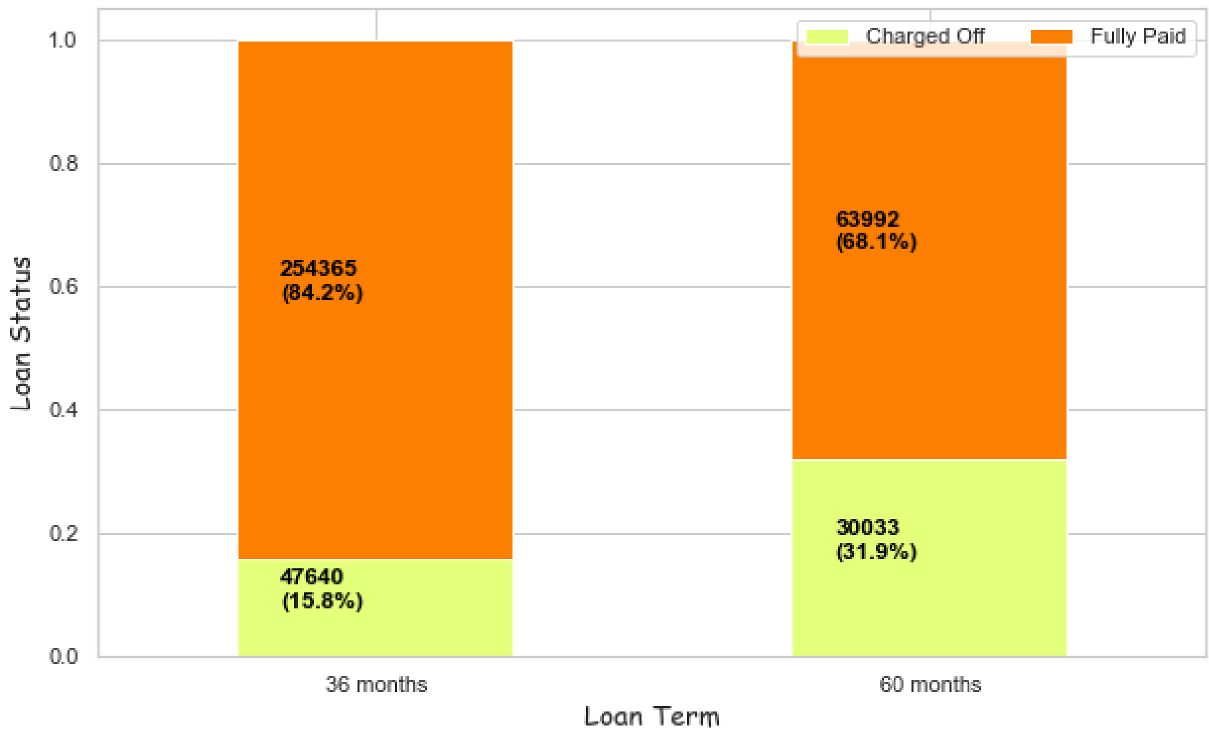
In [966...

```
count_plt(loan_data,'term','Loan Term',width=8,height=8)
```



In [967…

```
stack_bar(loan_data,'term',"Loan Term")
```



Converting to integrer value

In [968…

```
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.

Employee Title

In [969…

```
loan_data['emp_title'].nunique()
```

Out[969…

```
173105
```

In [970…

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

Out[970…

```
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 [971…

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

Out[971…

```
Fully Paid     3321
Charged Off     929
Name: loan_status, dtype: int64
```

In [972…

```
929/(3321+929)
```

Out[972…

```
0.21858823529411764
```

In [973…

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

Out[973…

```
Charged Off    6
Fully Paid     1
Name: loan_status, dtype: int64
```

In [974…

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

Out[974…

```
43.710072469257376
```

Inferences

- 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 [975…
```
loan_data.drop('emp_title',axis=1,inplace=True)
```

issue_d

In [976…
```
loan_data.columns
```

Out[976…
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'sub_grade',
       'emp_length', 'home_ownership', 'annual_inc', 'verification_status',
       'issue_d', 'loan_status', 'purpose', 'title', 'dti', 'earliest_cr_line',
       'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [977…
```
loan_data['issue_d'].value_counts(dropna=False)
```

Out[977…
```
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
```

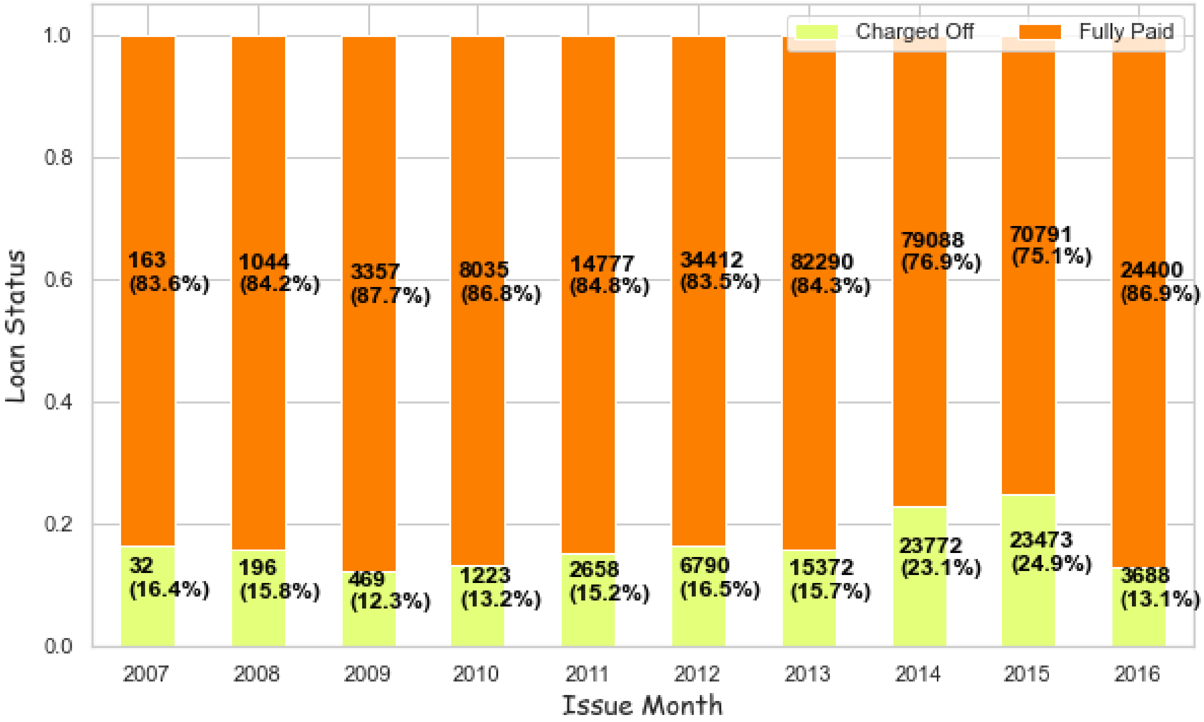
In [978…
```
loan_data["issue_d"] = pd.to_datetime(loan_data['issue_d'])
```

In [979…
```
loan_data['issue_d'] = loan_data['issue_d'].dt.year
```

In [980…
```
loan_data['issue_d'].value_counts(dropna=False)
```

Out[980…
```
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 [981…
```
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.

Employee Length

In [982…

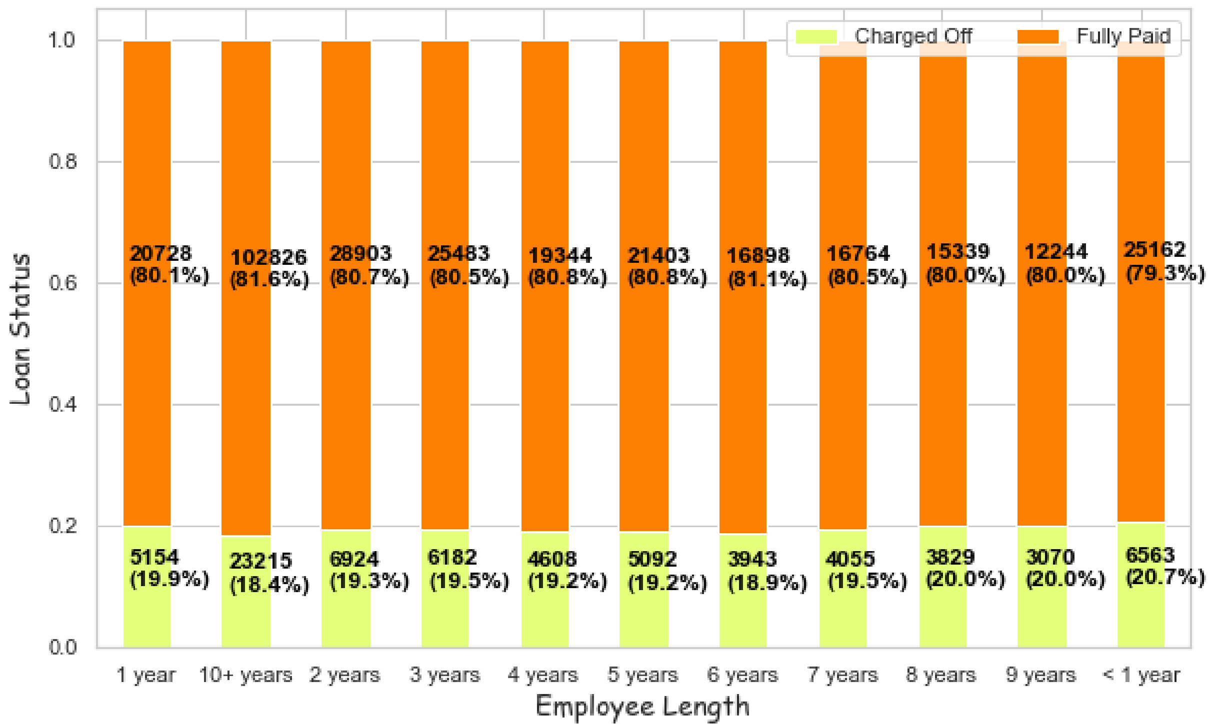
```
loan_data['emp_length'].value_counts(dropna=False)
```

Out[982…

```
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 [983…

```
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 [984…

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

Home Ownership

In [985…

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

Out[985…

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

Inferences

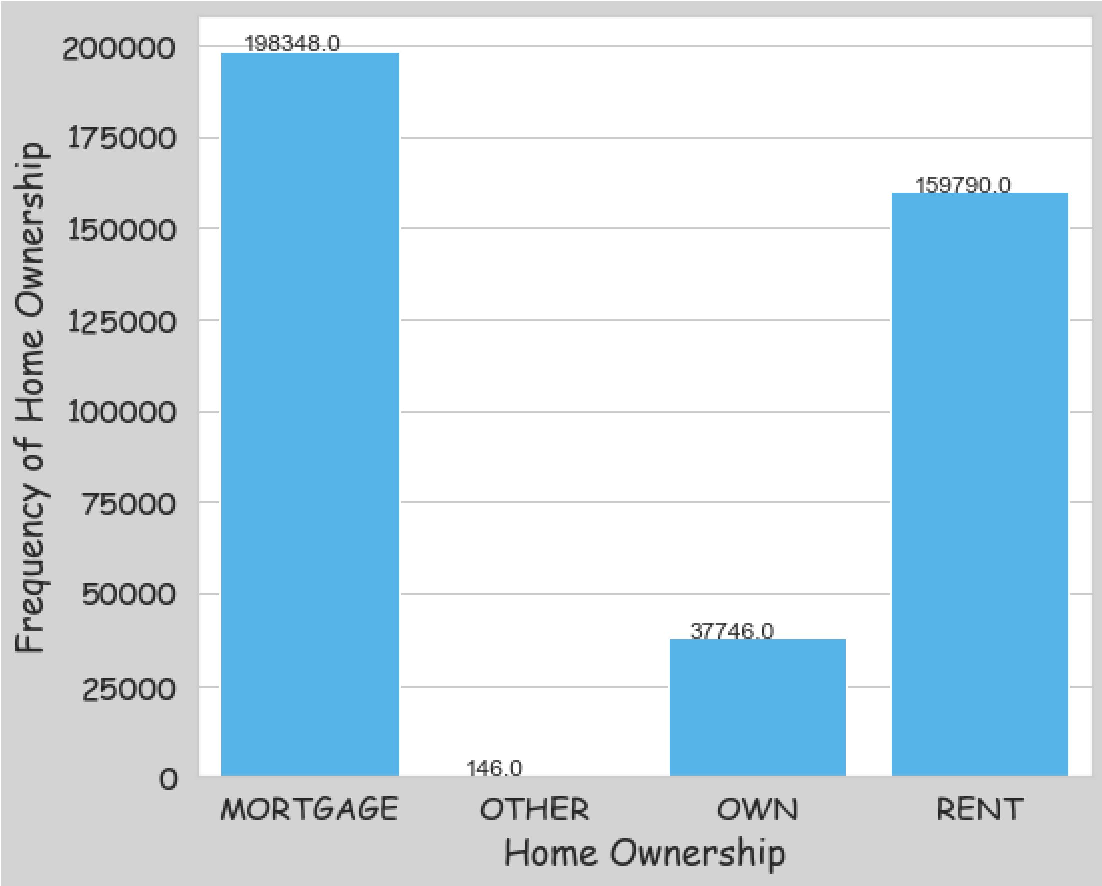
- Home Ownership Category - OTHER will be combined with NONE & ANY

In [986…

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

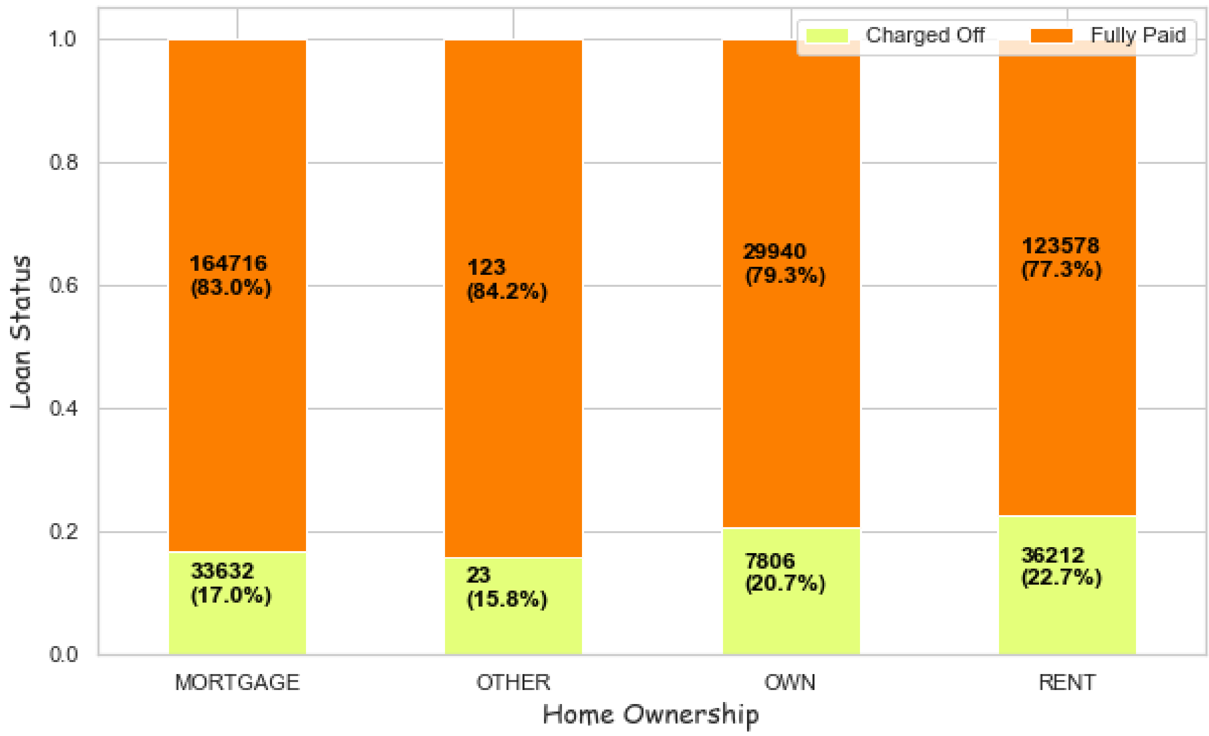
In [987…

```
count_plt(loan_data,'home_ownership','Home Ownership',width=8,height=7)
```



In [988…

```
stack_bar(loan_data,'home_ownership','Home Ownership')
```



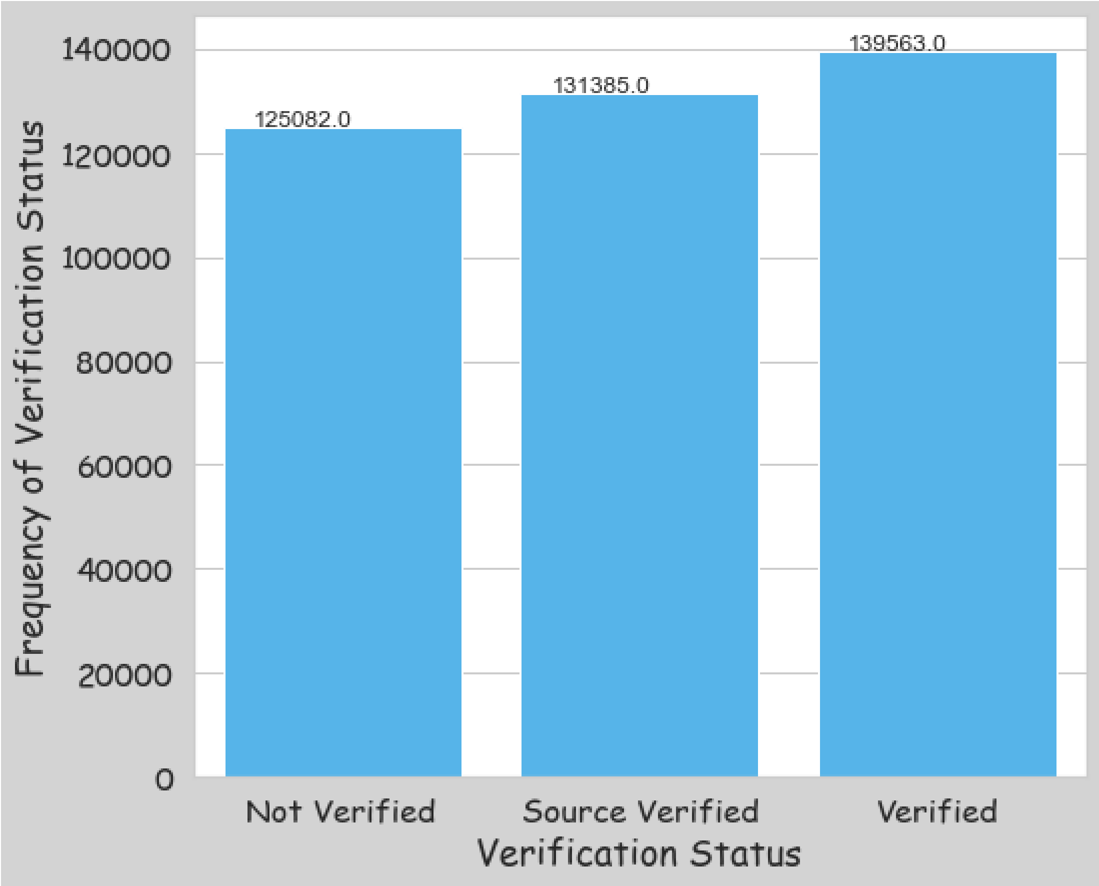
Inferences

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

Verfication Status

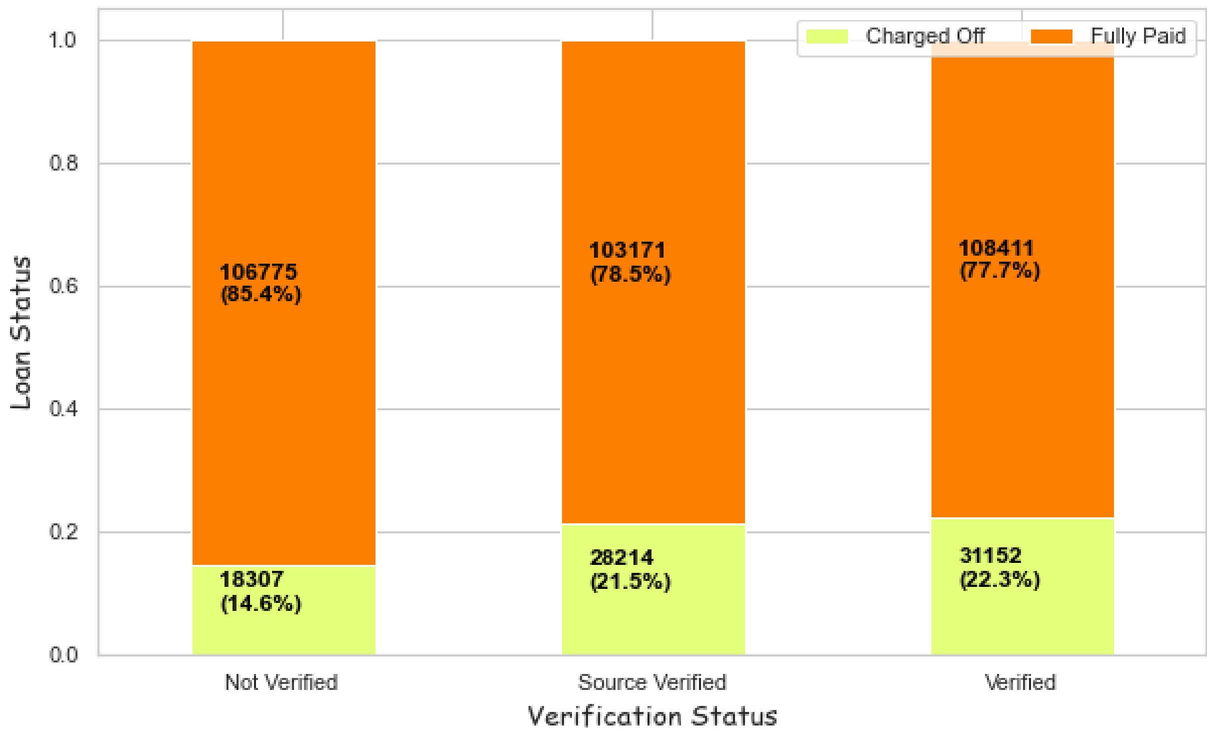
In [989…

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



In [990…

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



Inferences

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

Purpose of the loan

In [991…

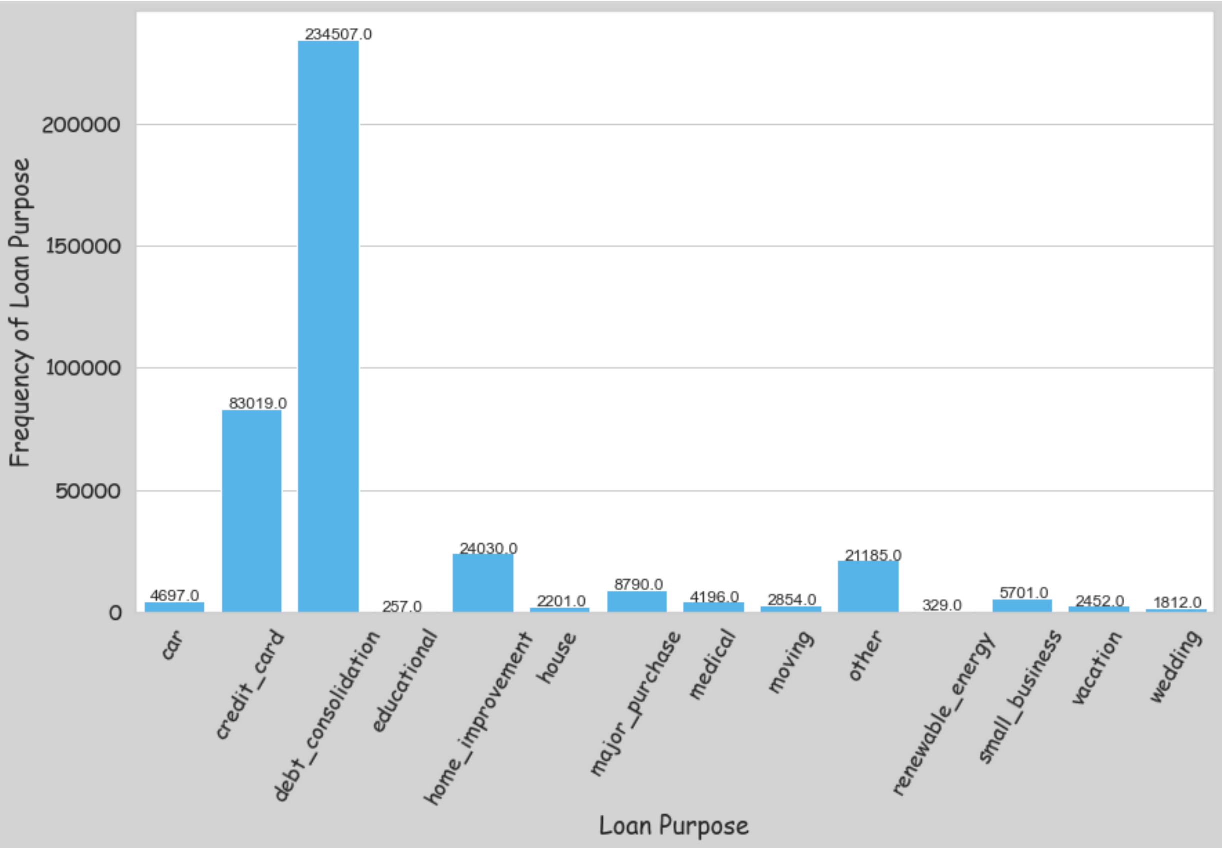
```
loan_data['purpose'].value_counts()
```

Out[991…

```
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
```

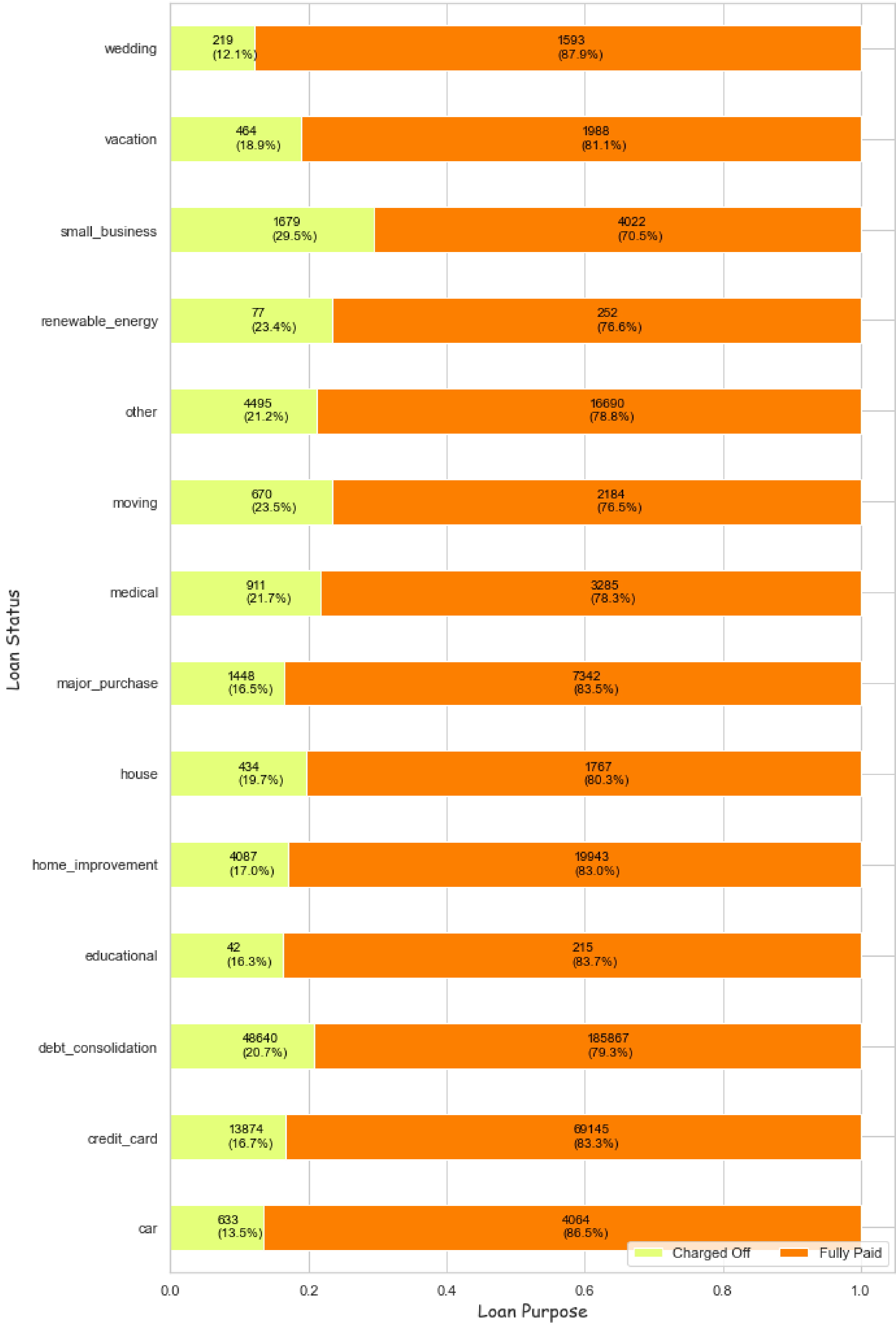
In [992…

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



In [993…

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



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 [994…
```
loan_data['title'].nunique()
```
Out[994…
```
48817
```
In [995…
```
loan_data['title'].value_counts().head(5)
```
Out[995…
```
Debt consolidation         152472
Credit card refinancing     51487
Home improvement            15264
Other                       12930
Debt Consolidation          11608
Name: title, dtype: int64
```
In [996…
```
loan_data['title'].value_counts().head(5)
```
Out[996…
```
Debt consolidation         152472
Credit card refinancing     51487
Home improvement            15264
Other                       12930
Debt Consolidation          11608
Name: title, dtype: int64
```
In [997…
```
loan_data.drop('title',axis=1,inplace=True)
```

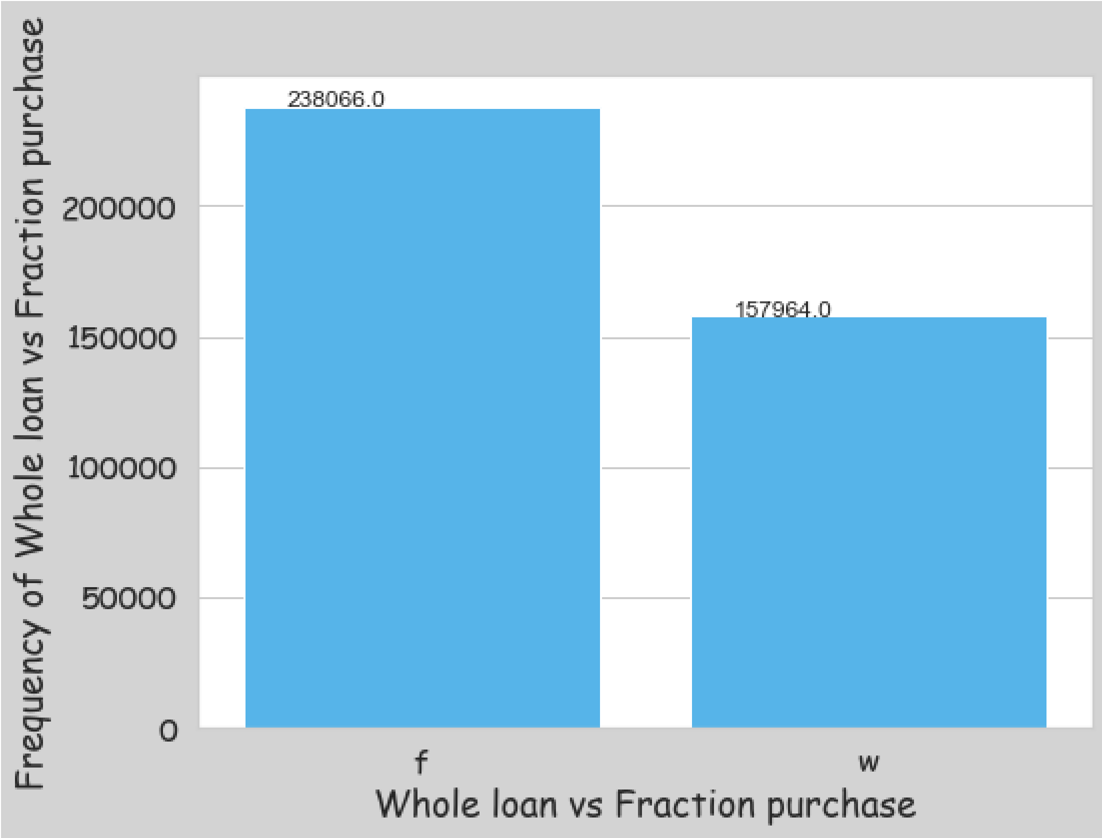
Inferences

- 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

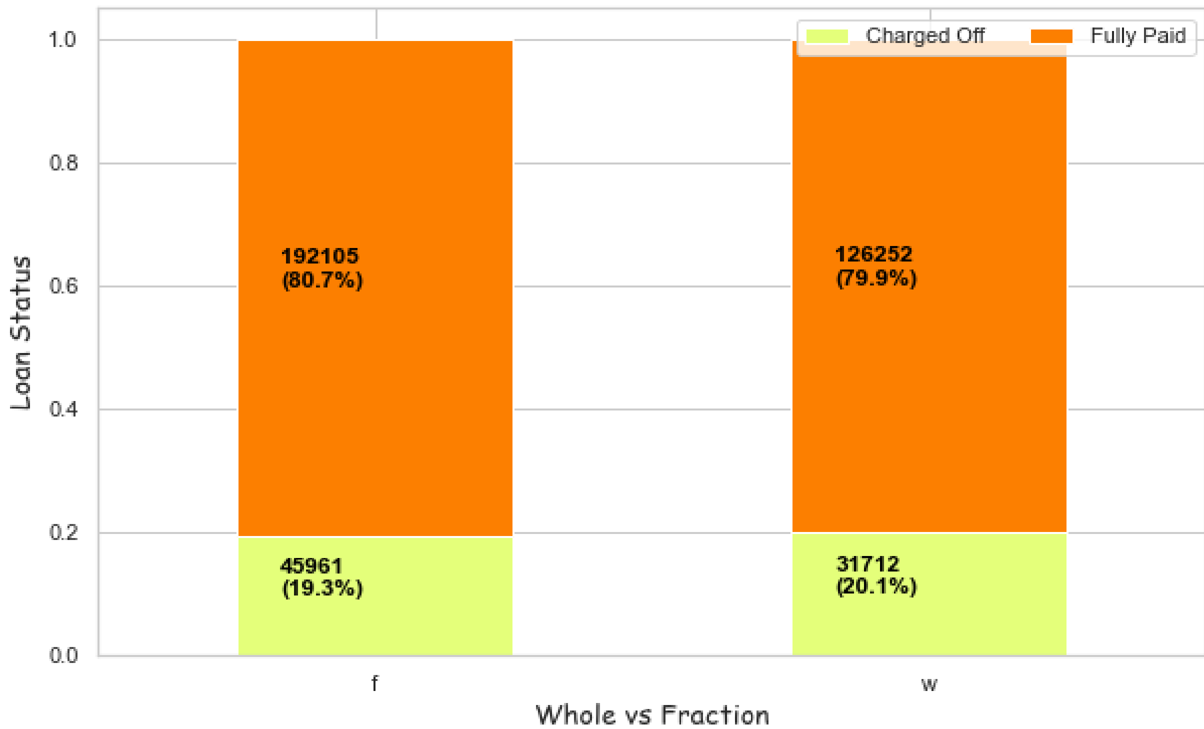
- Whole loan vs Fraction purpose
- Initial list status indicates the initial listing status of the loan. Possible values are W, F. W stands for whole loans, that is, available to investors to be purchased in their entirety (Borrowers benefit from getting 'instant funding').
- Neo bank provides a randomized 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 [998…
```
count_plt(loan_data,'initial_list_status','Whole loan vs Fraction purchase',width=8,
```



In [999…

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



Application Type

In [100…

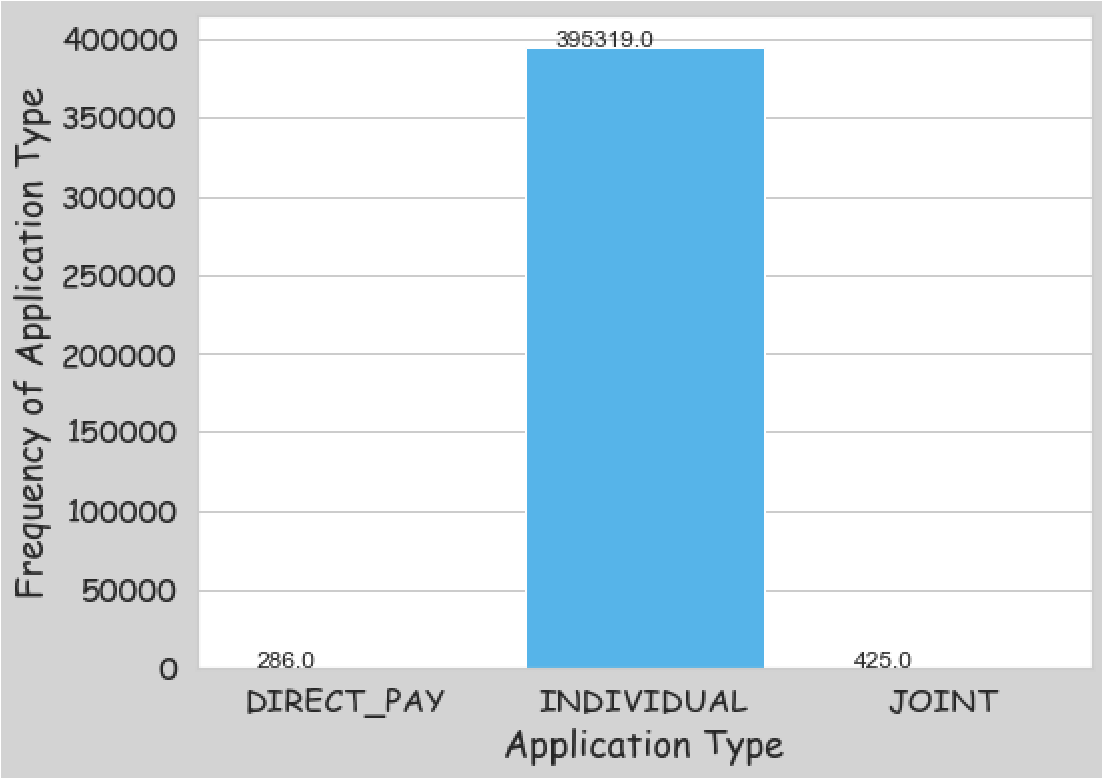
```
loan_data['application_type'].value_counts()
```

Out[100…

```
INDIVIDUAL     395319
JOINT             425
DIRECT_PAY        286
Name: application_type, dtype: int64
```

In [100…

```
count_plt(loan_data,'application_type','Application Type',width=8,height=6)
```



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 [100…

```
loan_data['address'].nunique()
```

Out[100…

```
393700
```

In [100…

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

Out[100…

```
99.41166073277276
```

Inference

- We can group the data by zipcode, which might provide us with more insights.

Inferences

- 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 [100…

```
loan_data.shape
```

Out[100…

```
(396030, 23)
```

earliest_cr_line

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

In [100…
```
loan_data['earliest_cr_line'].nunique()
```

Out[100…
```
684
```

In [100…
```
loan_data["earliest_cr_line"] = pd.to_datetime(loan_data['earliest_cr_line'])
```

In [100…
```
loan_data['earliest_cr_line'] = loan_data['earliest_cr_line'].dt.year
```

In [100…
```
loan_data['earliest_cr_line'].value_counts()
```

Out[100…
```
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
```

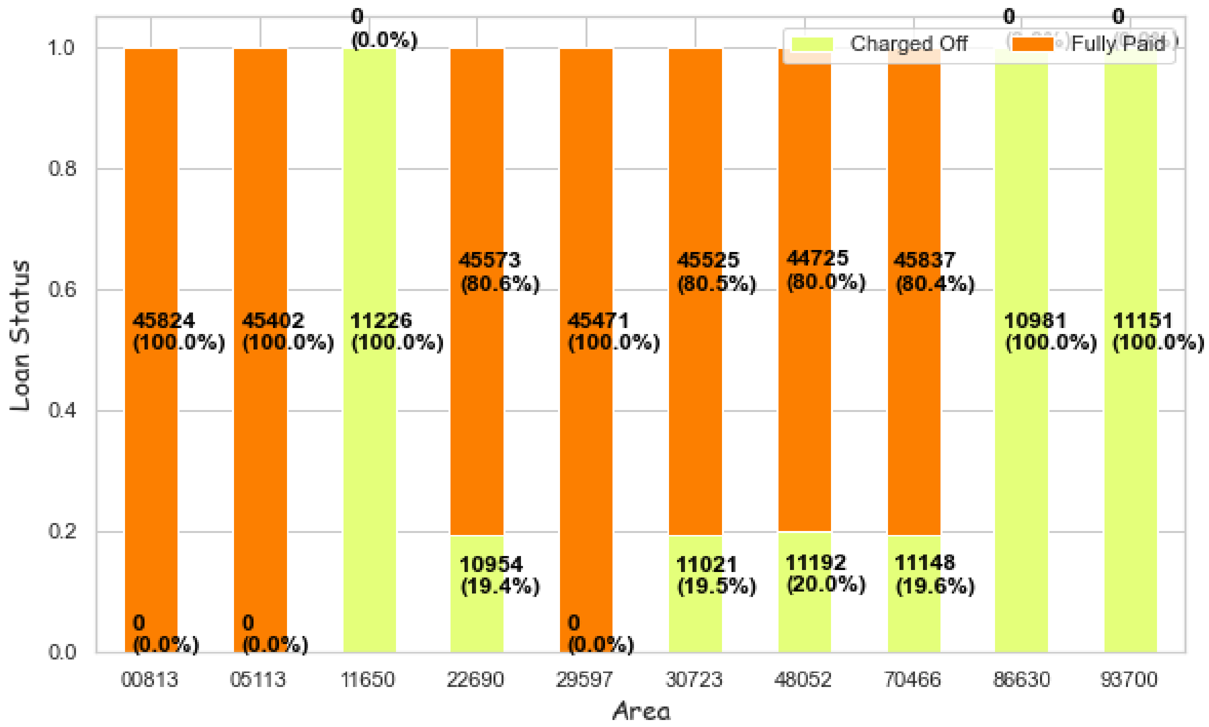
Feature Engineering

address

- Extracting the Zipcode from the address

In [100…
```
loan_data['zipcode'] = loan_data['address'].apply(lambda address:address[-5:])
```

In [101…
```
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 [101…

```
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 [101…

```
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 [101…

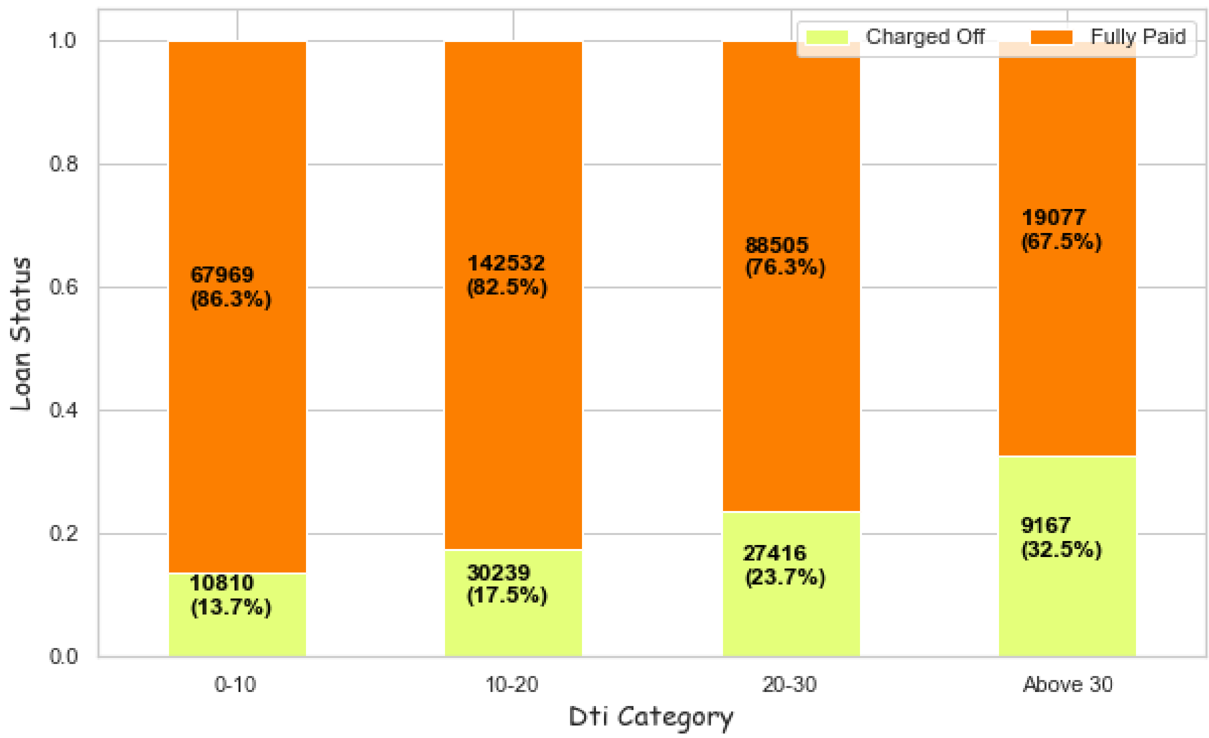
```
loan_data['dti_cat'].head()
```

Out[101…

```
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 [101…

```
stack_bar(loan_data,'dti_cat',"Dti Category")
```



In [101…

```
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 [101…

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

In [101…

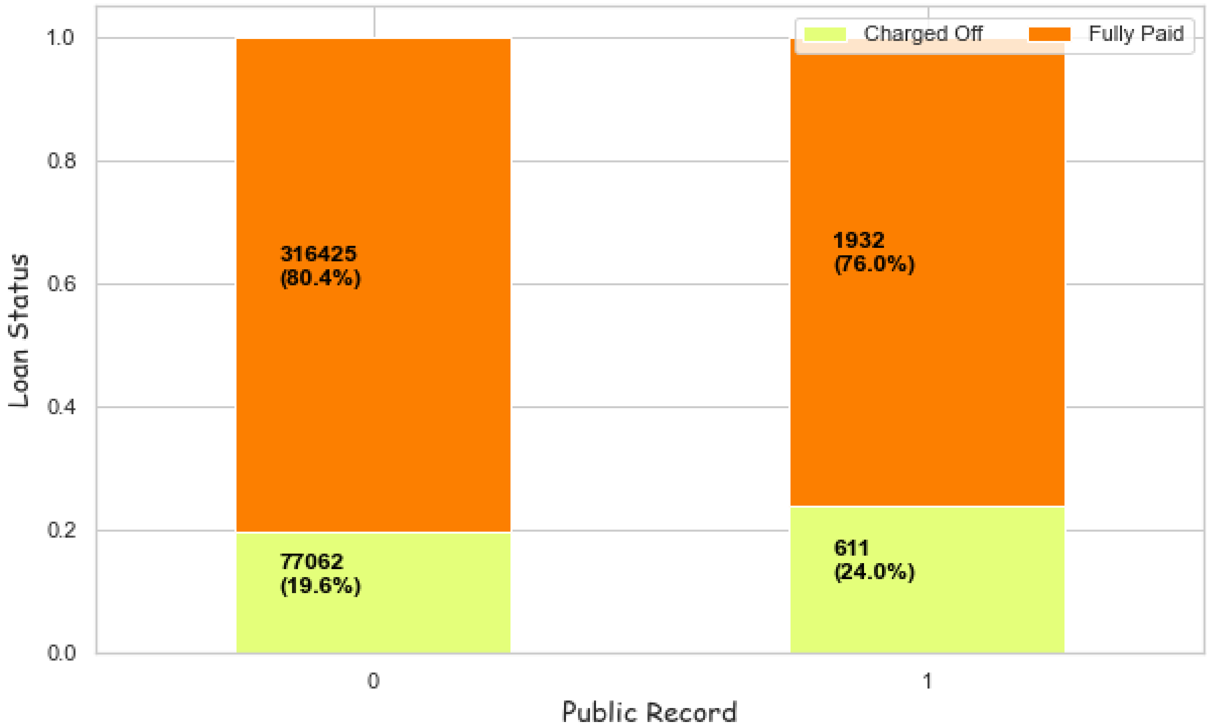
```
loan_data['pub_rec_cat'] = loan_data.pub_rec.apply(pub_rec)
```

In [101…

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

In [101…

```
stack_bar(loan_data,'pub_rec_cat',"Public Record")
```



In [102...
```
loan_data.drop('pub_rec',axis=1,inplace=True)
```

Inference

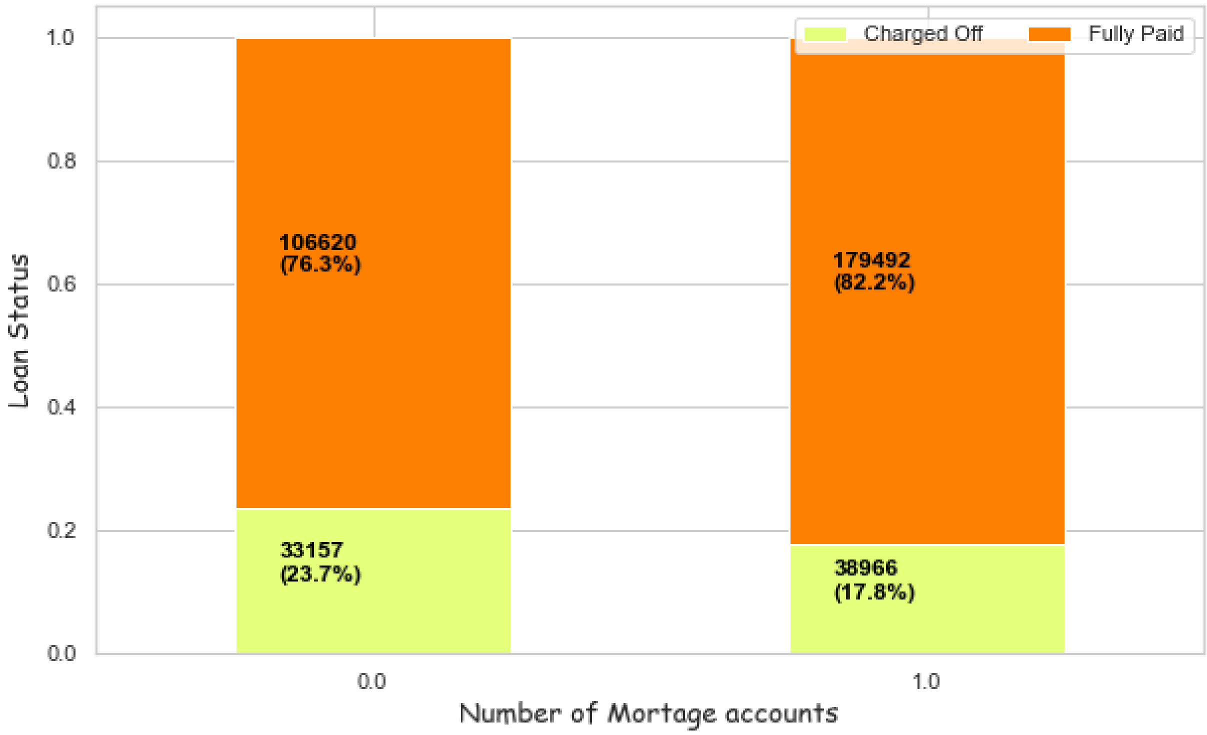
- If Public record having derogatory value more than 2 then we can see loan getting charged-off by 24%

mort_acc

In [102...
```
def mort_acc(num):
    if num == 0.0:
        return 0
    elif num >= 1.0:
        return 1
    else:
        return num
```

In [102...
```
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 [102...
```
stack_bar(loan_data,'mort_acc_cat',"Number of Mortage accounts")
```



In [102…

```
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 [102…

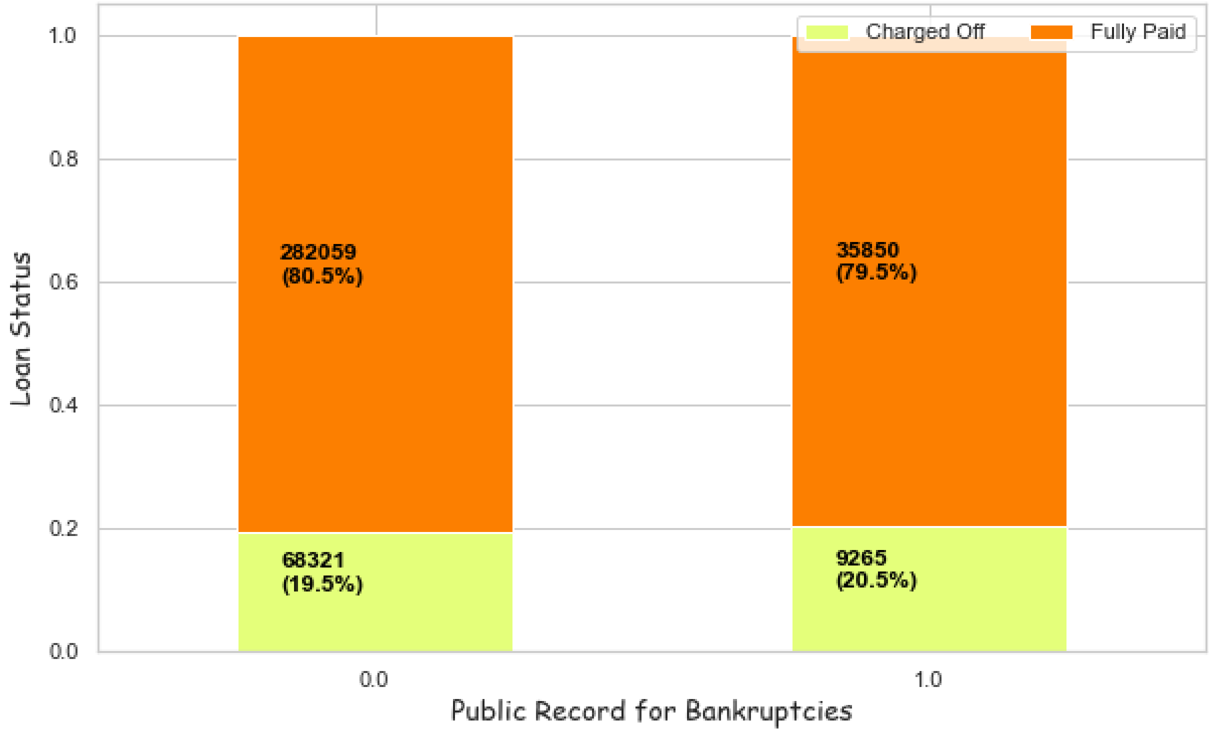
```
def pub_rec_bankruptcies(num):
    if num == 0.0:
        return 0
    elif num >= 1.0:
        return 1
    else:
        return num
```

In [102…

```
loan_data['pub_rec_bankruptcies_cat'] = loan_data.pub_rec_bankruptcies.apply(pub_rec
loan_data["pub_rec_bankruptcies_cat"] = loan_data["pub_rec_bankruptcies_cat"].astype
```

In [102…

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



In [102…
```
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_status

In [102…
```
loan_data['loan_status'].unique()
```

Out[102…
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [103…
```
def loan_status(str_):
    if str_ == 'Charged Off':
        return 1
    else:
        return 0
```

In [103…
```
loan_data['loan_status'] = loan_data.loan_status.apply(loan_status)
```

In [103…
```
loan_data['loan_status'].unique()
```

Out[103…
```
array([0, 1], dtype=int64)
```

In [103…
```
loan_data.shape
```

Out[103…
```
(396030, 23)
```

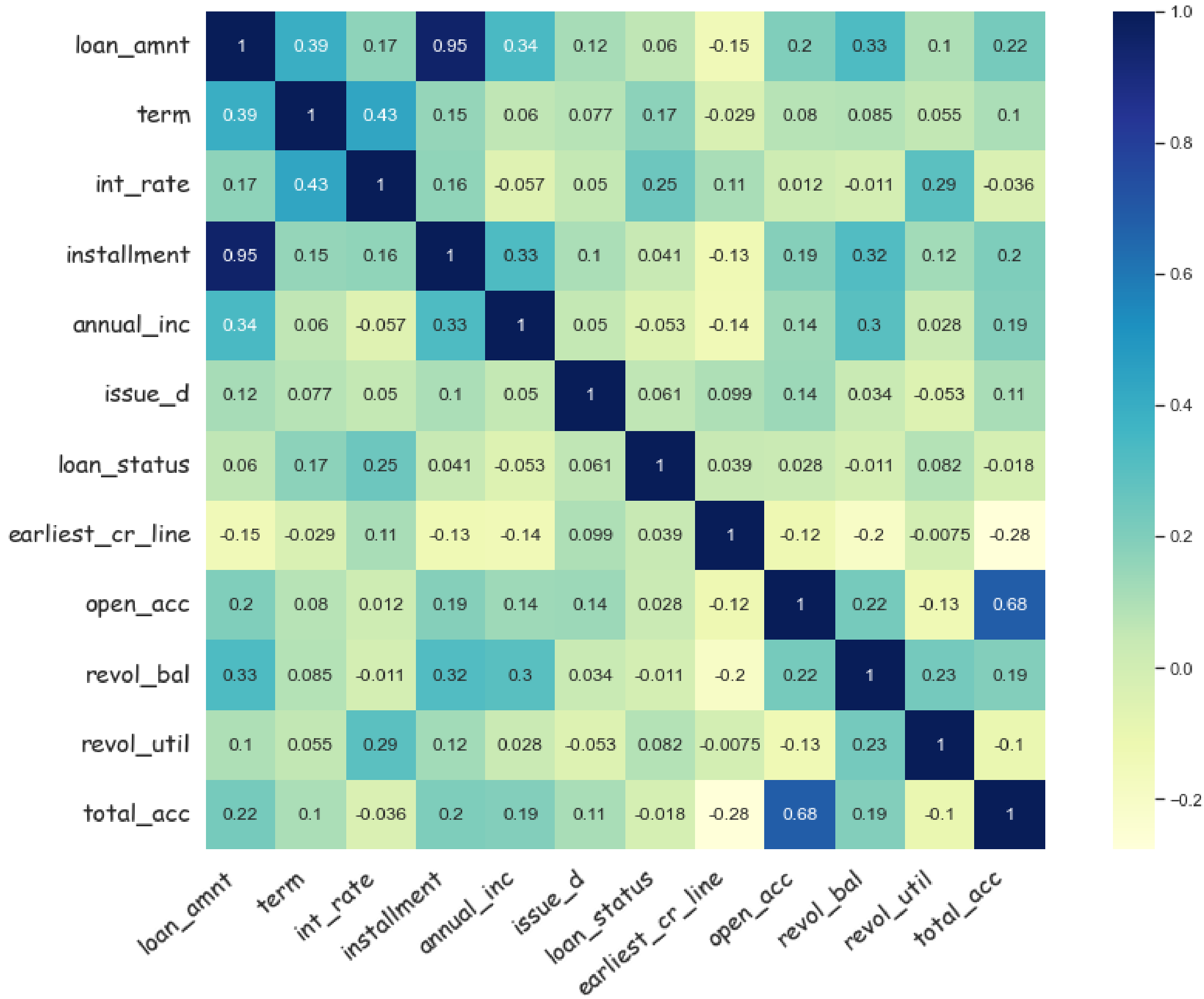
Inferences

- 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 [103…

```
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 variable

- **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 [103…

```
loan_data.columns
```

Out[103…

```
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', '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 [103…

```
loan_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 23 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  revol_bal                 396030 non-null  float64
 14  revol_util                395754 non-null  float64
 15  total_acc                 396030 non-null  float64
 16  initial_list_status       396030 non-null  object
 17  application_type          396030 non-null  object
 18  zipcode                   396030 non-null  object
 19  dti_cat                   395715 non-null  category
 20  pub_rec_cat               396030 non-null  category
 21  mort_acc_cat              358235 non-null  category
 22  pub_rec_bankruptcies_cat  395495 non-null  category
dtypes: category(4), float64(8), int64(3), int8(1), object(7)
memory usage: 56.3+ MB
```

In [103…

```
loan_data.shape
```

Out[103…

```
(396030, 23)
```

In [103…

```
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 [103…

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

Out[103…

```
(396030, 71)
```

In [104…

```
dummyVar.head()
```

Out[104…

	issue_d	sub_grade_A2	sub_grade_A3	sub_grade_A4	sub_grade_A5	sub_grade_B1	sub_grade_B2
0	2015	0	0	0	0	0	0

	issue_d	sub_grade_A2	sub_grade_A3	sub_grade_A4	sub_grade_A5	sub_grade_B1	sub_grade_B2
1	2015	0	0	0	0	0	0
2	2015	0	0	0	0	0	0
3	2014	1	0	0	0	0	0
4	2013	0	0	0	0	0	0

5 rows × 71 columns

In [104…

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

Out[104…

```
(396030, 94)
```

In [104…

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

Out[104…

```
(396030, 81)
```

In [104…

```
loan_data_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 81 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   annual_inc                          396030 non-null  float64
 5   loan_status                         396030 non-null  int64  
 6   earliest_cr_line                    396030 non-null  int64  
 7   open_acc                            396030 non-null  float64
 8   revol_bal                           396030 non-null  float64
 9   revol_util                          395754 non-null  float64
 10  total_acc                           396030 non-null  float64
 11  sub_grade_A2                        396030 non-null  uint8  
 12  sub_grade_A3                        396030 non-null  uint8  
 13  sub_grade_A4                        396030 non-null  uint8  
 14  sub_grade_A5                        396030 non-null  uint8  
 15  sub_grade_B1                        396030 non-null  uint8  
 16  sub_grade_B2                        396030 non-null  uint8  
 17  sub_grade_B3                        396030 non-null  uint8  
 18  sub_grade_B4                        396030 non-null  uint8  
 19  sub_grade_B5                        396030 non-null  uint8  
 20  sub_grade_C1                        396030 non-null  uint8  
 21  sub_grade_C2                        396030 non-null  uint8  
 22  sub_grade_C3                        396030 non-null  uint8  
 23  sub_grade_C4                        396030 non-null  uint8  
 24  sub_grade_C5                        396030 non-null  uint8  
 25  sub_grade_D1                        396030 non-null  uint8  
 26  sub_grade_D2                        396030 non-null  uint8  
```

```
 27  sub_grade_D3                        396030 non-null  uint8
 28  sub_grade_D4                        396030 non-null  uint8
 29  sub_grade_D5                        396030 non-null  uint8
 30  sub_grade_E1                        396030 non-null  uint8
 31  sub_grade_E2                        396030 non-null  uint8
 32  sub_grade_E3                        396030 non-null  uint8
 33  sub_grade_E4                        396030 non-null  uint8
 34  sub_grade_E5                        396030 non-null  uint8
 35  sub_grade_F1                        396030 non-null  uint8
 36  sub_grade_F2                        396030 non-null  uint8
 37  sub_grade_F3                        396030 non-null  uint8
 38  sub_grade_F4                        396030 non-null  uint8
 39  sub_grade_F5                        396030 non-null  uint8
 40  sub_grade_G1                        396030 non-null  uint8
 41  sub_grade_G2                        396030 non-null  uint8
 42  sub_grade_G3                        396030 non-null  uint8
 43  sub_grade_G4                        396030 non-null  uint8
 44  sub_grade_G5                        396030 non-null  uint8
 45  home_ownership_OTHER                396030 non-null  uint8
 46  home_ownership_OWN                  396030 non-null  uint8
 47  home_ownership_RENT                 396030 non-null  uint8
 48  verification_status_Source Verified  396030 non-null  uint8
 49  verification_status_Verified        396030 non-null  uint8
 50  purpose_credit_card                 396030 non-null  uint8
 51  purpose_debt_consolidation          396030 non-null  uint8
 52  purpose_educational                 396030 non-null  uint8
 53  purpose_home_improvement            396030 non-null  uint8
 54  purpose_house                       396030 non-null  uint8
 55  purpose_major_purchase              396030 non-null  uint8
 56  purpose_medical                     396030 non-null  uint8
 57  purpose_moving                      396030 non-null  uint8
 58  purpose_other                       396030 non-null  uint8
 59  purpose_renewable_energy            396030 non-null  uint8
 60  purpose_small_business              396030 non-null  uint8
 61  purpose_vacation                    396030 non-null  uint8
 62  purpose_wedding                     396030 non-null  uint8
 63  initial_list_status_w               396030 non-null  uint8
 64  application_type_INDIVIDUAL         396030 non-null  uint8
 65  application_type_JOINT              396030 non-null  uint8
 66  zipcode_05113                       396030 non-null  uint8
 67  zipcode_11650                       396030 non-null  uint8
 68  zipcode_22690                       396030 non-null  uint8
 69  zipcode_29597                       396030 non-null  uint8
 70  zipcode_30723                       396030 non-null  uint8
 71  zipcode_48052                       396030 non-null  uint8
 72  zipcode_70466                       396030 non-null  uint8
 73  zipcode_86630                       396030 non-null  uint8
 74  zipcode_93700                       396030 non-null  uint8
 75  dti_cat_10-20                       396030 non-null  uint8
 76  dti_cat_20-30                       396030 non-null  uint8
 77  dti_cat_Above 30                    396030 non-null  uint8
 78  pub_rec_cat_1                       396030 non-null  uint8
 79  mort_acc_cat_1.0                    396030 non-null  uint8
 80  pub_rec_bankruptcies_cat_1.0        396030 non-null  uint8
dtypes: float64(8), int64(2), int8(1), uint8(70)
memory usage: 57.0 MB
```

In [104…

```
loan_data_encoded.columns
```

Out[104…

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc',
       'loan_status', 'earliest_cr_line', 'open_acc', '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

- Train 60%
- Cross validation - 20%
- Test validation - 20%

train & test Split

In [104…

```
# Train & Test data split
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
```

In [104…

```
#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 & Cross validation split

In [104…

```
# Splitting the data into train and test
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv,y_tr_cv,test_size=0.25,ran
```

In [ ]:

```

```

Libraries used for Model creation

In [104…

```
# 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
```

Utility function draw ROC curve

- True Positve rate vs False Positive rate

In [104…

```
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
```

Handlng 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 [105…

```
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 [105…
```
scaler = StandardScaler()
```

1. Basic Model creation

Build Pipeline

- Imputation
- Rescaling
- Building the model

In [105…
```
pl_basic_logreg = Pipeline(steps=[('imputer',imputer),
                                  ('scaler',scaler),
                                  ('logistic_model',LogisticRegression())
                                 ])
```

In [105…
```
pl_basic_logreg.fit(X_train,y_train)
```

Out[105…
```
Pipeline(steps=[('imputer', SimpleImputer(strategy='median')),
                ('scaler', StandardScaler()),
                ('logistic_model', LogisticRegression())])
```

In [105…
```
train_y_pred = pl_basic_logreg.predict(X_train)
train_score = f1_score(y_train, train_y_pred)
```

In [105…
```
print(" F1 Score for Basic Model (Train) ", train_score)
```

```
 F1 Score for Basic Model (Train)  0.6158897293857302
```

In [105…
```
X_test['revol_util'] = X_test['revol_util'].fillna(X_test['revol_util'].median())
```

In [105…
```
y_pred_test = pl_basic_logreg.predict(X_test)
test_score = f1_score(y_test, y_pred_test)

print("F1 Score for Basic Model (Test) ",test_score)
```

```
F1 Score for Basic Model (Test)  0.6230048577376821
```

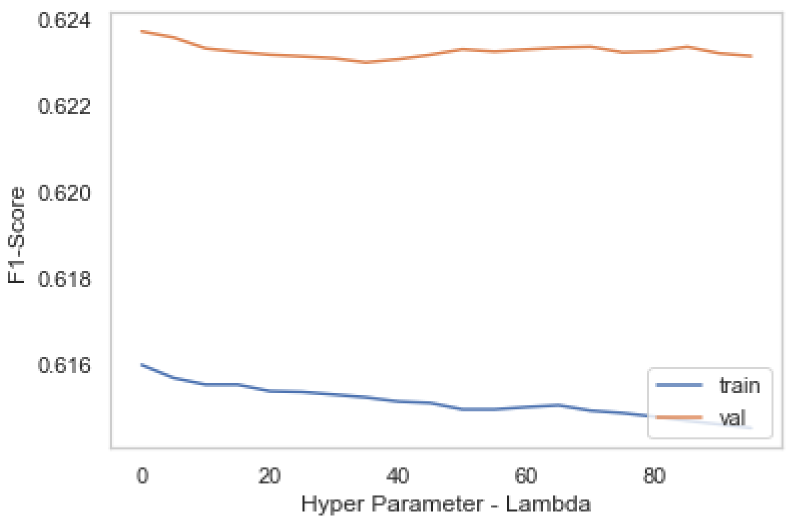
2. Using Hyper-parmeter Optimization

In [105…
```
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 [105…

```
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 [106…

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

```
0.623721793157009
```

In [106…

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

In [106…

```
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)  0.6227887617065556
```

In [106…

```
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 : 89.01598363760321%
recall_score : 46.35043562439496%
precision_score : 94.8870392390012%
f1_score : 62.278876170655565%
AUC score : 72.87150259818421%
confusion_matrix :
[[63324   387]
 [ 8313  7182]]
```

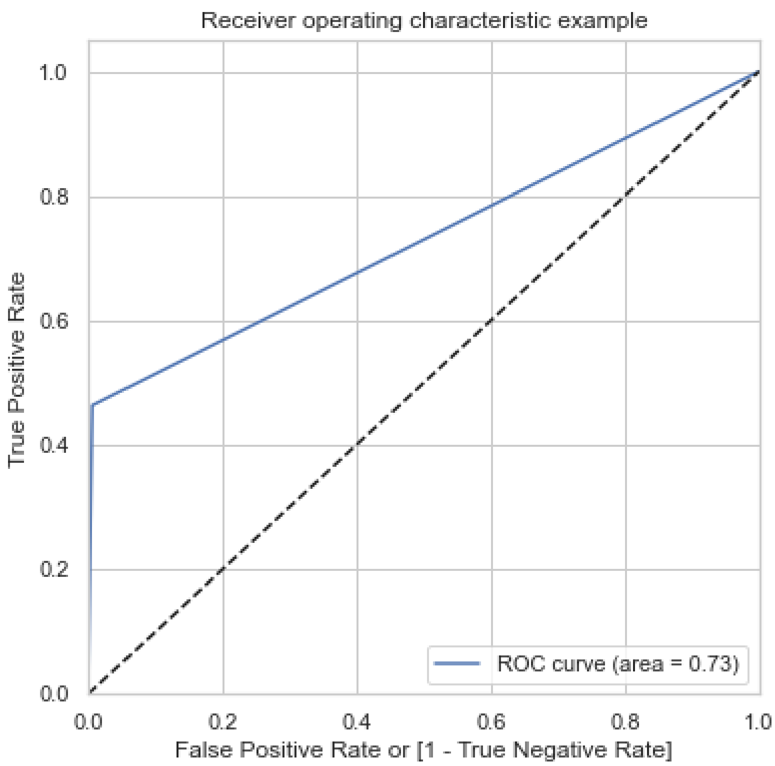
In [106…

```
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 [106…

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



Out[106…

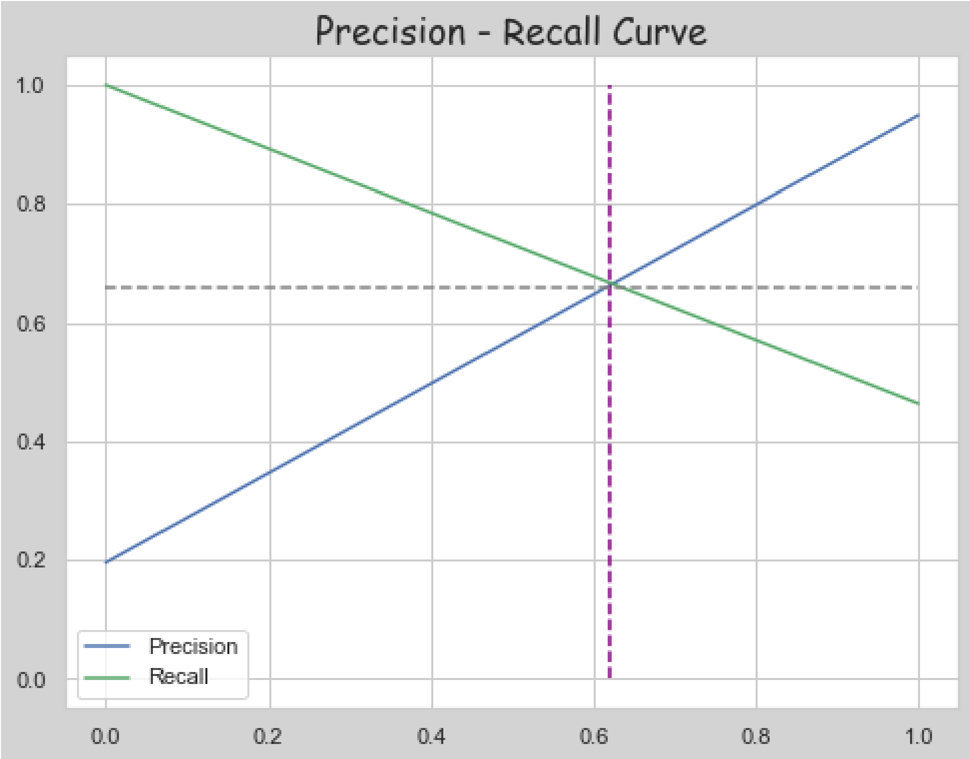
```
(array([0.       , 0.0060743, 1.       ]),
 array([0.        , 0.46350436, 1.        ]),
 array([2, 1, 0], dtype=int64))
```

Recall vs Precision

In [106…

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

```
fig.set_facecolor("lightgrey")

# Precision Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_hp_test)
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 [ ]:

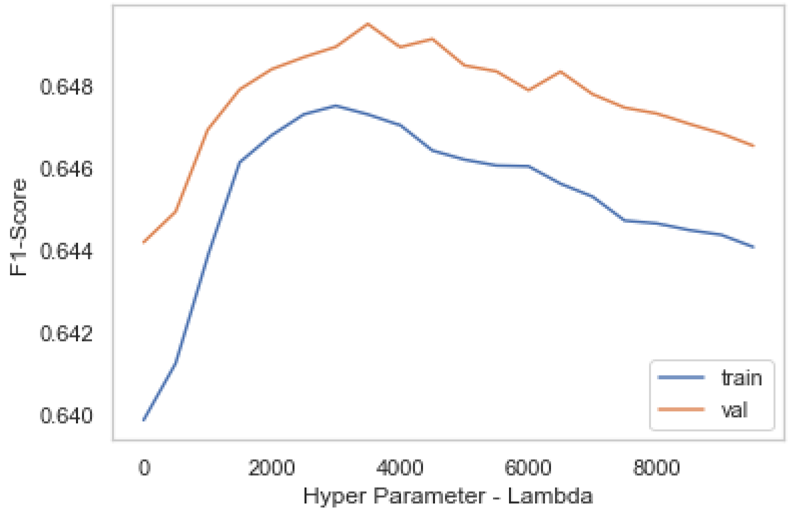
3. Advanced Model with Hyper-parameter, and balancing the data using class weights

In [106…

```
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_,class
    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 [106…

```
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 [106…

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

```
0.649514408533673
```

In [107…

```
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_
                             ])
best_hp_clwg_logreg.fit(X_train, y_train)

y_pred_test = best_hp_clwg_logreg.predict(X_test)
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)  0.6489930522204078
```

In [107…

```
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 : 86.15887685276368%
recall_score : 65.4081961923201%
precision_score : 64.39827169907231%
f1_score : 64.89930522204078%
AUC score : 78.30689824056212%
confusion_matrix :
```

```
[[58108  5603]
 [ 5360 10135]]
```

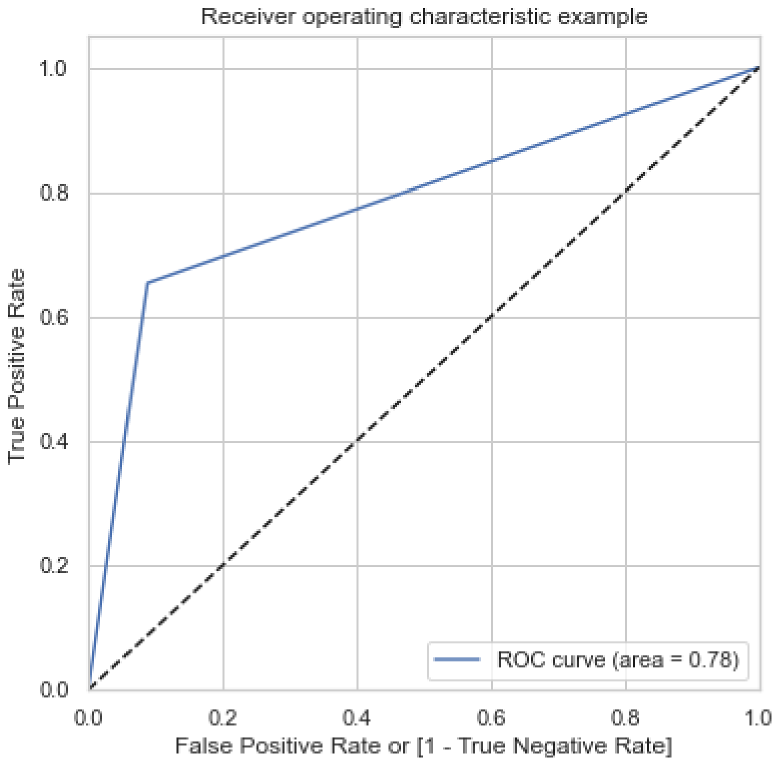
In [107…

```
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 [107…

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



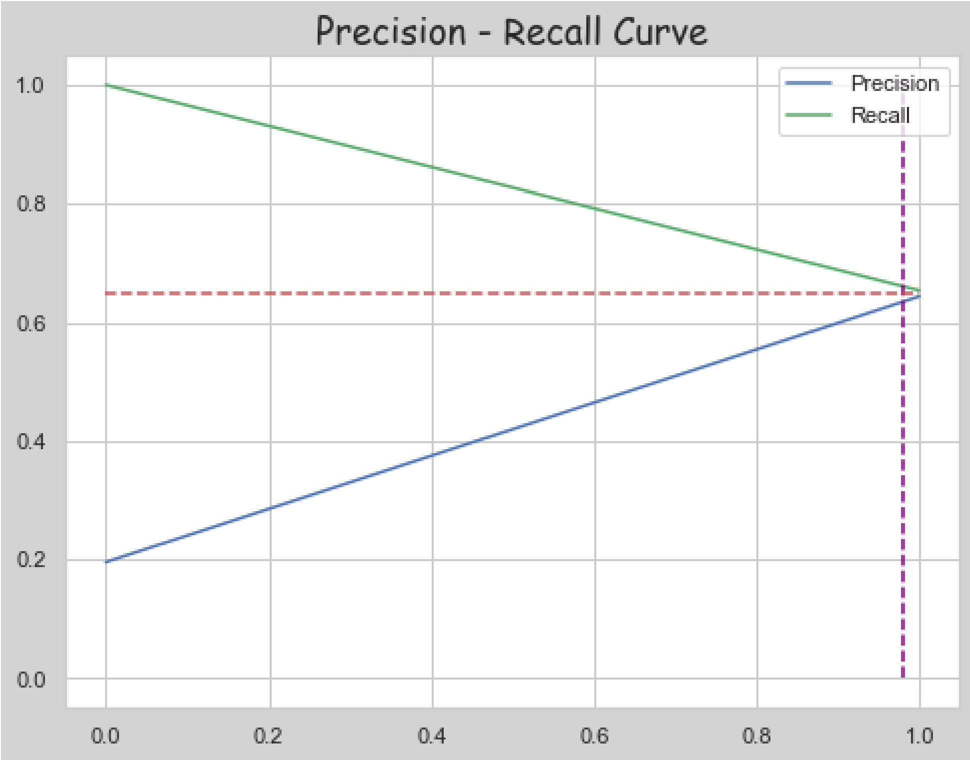
Out[107…

```
(array([0.      , 0.087944, 1.      ]),
 array([0.        , 0.65408196, 1.        ]),
 array([2, 1, 0], dtype=int64))
```

Recall vs Precision

In [107…

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

# Precision Recall Curve
precision, recall, thresholds = 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 [107…
```
X_train['revol_util'] = X_train['revol_util'].fillna(X_train['revol_util'].median())
```

In [108…
```
rfe = RFE(best_hp_clwg_logreg['logistic_model'], n_features_to_select=15)
rfe = rfe.fit(X_train, y_train)
```

In [108…
```
cols=X_train.columns[rfe.support_]
cols
```

Out[108…
```
Index(['int_rate', 'sub_grade_C4', 'home_ownership_RENT',
       'verification_status_Source 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 [108…
```
#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 [108…
```
# 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 [108…

```
# 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:                -67617.
Date:                Sun, 08 May 2022   Deviance:                   1.3523e+05
Time:                        23:13:04   Pearson chi2:                 1.60e+05
No. Iterations:                    30   Pseudo R-squ. (CS):             0.3437
Covariance Type:            nonrobust
====================================================================================
===================
                                          coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------------
-------------------
const                                  -4.0047      0.033   -120.623      0.000
-4.070      -3.940
int_rate                                0.1317      0.002     84.289      0.000
0.129       0.135
sub_grade_C4                            0.1597      0.029      5.578      0.000
0.104       0.216
home_ownership_RENT                     0.1826      0.017     10.768      0.000
0.149       0.216
verification_status_Source Verified     0.1862      0.015     12.836      0.000
0.158       0.215
purpose_debt_consolidation              0.0741      0.014      5.148      0.000
0.046       0.102
initial_list_status_w                   0.1049      0.014      7.325      0.000
0.077       0.133
zipcode_05113                         -29.8621   2.53e+04     -0.001      0.999    -
4.96e+04    4.96e+04
zipcode_11650                          33.1021   5.09e+04      0.001      0.999    -
9.98e+04    9.98e+04
zipcode_29597                         -29.8709   2.53e+04     -0.001      0.999    -
4.96e+04    4.96e+04
zipcode_86630                          33.1017   5.21e+04      0.001      0.999    -
1.02e+05    1.02e+05
zipcode_93700                          33.1084   5.16e+04      0.001      0.999    -
1.01e+05    1.01e+05
dti_cat_10-20                           0.2132      0.021     10.248      0.000
0.172       0.254
dti_cat_20-30                           0.4833      0.022     22.478      0.000
0.441       0.525
dti_cat_Above 30                        0.7460      0.028     26.343      0.000
0.691       0.802
mort_acc_cat_1.0                       -0.1390      0.017     -8.186      0.000
-0.172      -0.106
====================================================================================
===================
```

In [109…

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

In [109…

```
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:            -1.0890e+05
Date:                Sun, 08 May 2022   Deviance:                   2.1781e+05
Time:                        23:13:35   Pearson chi2:                 2.34e+05
No. Iterations:                     5   Pseudo R-squ. (CS):            0.07096
Covariance Type:            nonrobust
====================================================================================
===================
                                          coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------------
-------------------
const                                  -3.7838      0.026   -146.886      0.000
-3.834      -3.733
int_rate                                0.1322      0.001    107.986      0.000
0.130       0.135
sub_grade_C4                            0.1484      0.022      6.669      0.000
0.105       0.192
home_ownership_RENT                     0.1856      0.013     14.119      0.000
0.160       0.211
verification_status_Source Verified     0.1737      0.011     15.376      0.000
0.152       0.196
purpose_debt_consolidation              0.0523      0.011      4.689      0.000
0.030       0.074
initial_list_status_w                   0.0996      0.011      8.947      0.000
0.078       0.121
dti_cat_10-20                           0.2124      0.016     13.181      0.000
0.181       0.244
dti_cat_20-30                           0.5073      0.017     30.506      0.000
0.475       0.540
dti_cat_Above 30                        0.7856      0.022     35.744      0.000
0.743       0.829
mort_acc_cat_1.0                       -0.1411      0.013    -10.720      0.000
-0.167      -0.115
====================================================================================
===================
```

In [109…

```
# 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[109…

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:**	-1.0890e+05

Date:	Sun, 08 May 2022	**Deviance:**	2.1781e+05
Time:	23:13:50	**Pearson chi2:**	2.34e+05
No. Iterations:	5	**Pseudo R-squ. (CS):**	0.07096
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.7838	0.026	-146.886	0.000	-3.834	-3.733
int_rate	0.1322	0.001	107.986	0.000	0.130	0.135
sub_grade_C4	0.1484	0.022	6.669	0.000	0.105	0.192
home_ownership_RENT	0.1856	0.013	14.119	0.000	0.160	0.211
verification_status_Source Verified	0.1737	0.011	15.376	0.000	0.152	0.196
purpose_debt_consolidation	0.0523	0.011	4.689	0.000	0.030	0.074
initial_list_status_w	0.0996	0.011	8.947	0.000	0.078	0.121
dti_cat_10-20	0.2124	0.016	13.181	0.000	0.181	0.244
dti_cat_20-30	0.5073	0.017	30.506	0.000	0.475	0.540
dti_cat_Above 30	0.7856	0.022	35.744	0.000	0.743	0.829
mort_acc_cat_1.0	-0.1411	0.013	-10.720	0.000	-0.167	-0.115

In [109…

```
# Make a VIF dataframe for all the variables present

vif = pd.DataFrame()
vif['Features'] = X_train_GM.columns
vif['VIF'] = [variance_inflation_factor(X_train_GM.values, i) for i in range(X_train
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[109…

	Features	VIF
0	int_rate	6.20
9	mort_acc_cat_1.0	2.82
6	dti_cat_10-20	2.78
4	purpose_debt_consolidation	2.43
7	dti_cat_20-30	2.30
2	home_ownership_RENT	2.17
5	initial_list_status_w	1.68
3	verification_status_Source Verified	1.49
8	dti_cat_Above 30	1.38
1	sub_grade_C4	1.06

Inferences

- Key features that heavily affected the outcome are -

- dti, mort_acc, verification_status, sub_grade & int_rate

Confusion Metrics w.r.t. Jai Kisan case Study

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

Depending on the business 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, Jai Kisan Neo bank 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 2 & Model 3

	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 62**only.
- 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 Model

- 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 62**only.
- 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 [ ]:

```
```