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

Importing libraries / Read data¶ ¶

In [235]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [236]:

```
import re
```

In [237]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [238]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [239]:

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

In [240]:

```
df1 = df.copy()
```

In [241]:

```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

1. Exploratory Data Analysis

Shape of the data

In [242]:

```
df.shape
```

Out[242]:

```
(396030, 27)
```

Number and data types of variables

In [243]:

```
df.info()
```

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

In [244]:

```
df.head()
```

Out[244]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	hom
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	
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	

In [245]:

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

Out[245]:

```
Fully Paid     318357
Charged Off     77673
Name: loan_status, dtype: int64
```

Five point summary (Statistical summary)

In [246]:

```
#Categorical variables and numerical variables
numeric_df = df.select_dtypes(include=[np.number])
categorical_df = df.select_dtypes(exclude=[np.number])
```

In [247]:

```
numeric_df.describe()
```

Out[247]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.00(
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.31'
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.00(
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.00(
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.00(
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.00(
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.00(

In [248]:

```
categorical_df.describe()
```

Out[248]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_stat
count	396030	396030	396030	373103	377729	396030	3960
unique	2	7	35	173105	11	6	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Verifi
freq	302005	116018	26655	4389	126041	198348	1395

Missing values

In [249]:

```
df.isnull().sum()
```

Out[249]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
dtype: int64
```

Observations

- There are 396030 records with 27 features.
- 12 features are of type float, 15 features are of type object
- There are null values in the dataset

  Following columns have the missing values:
- emp_title
- emp_length
- title
- revol_util
- mort_acc
- pub_rec_bankruptcies
- Mean and Standard deviation varies for almost all numerical variables, may be due to the presence of outliers in it
- Fully paid customers are more compared to Charged off in loan_status feature (target)

Non-Graphical Analysis: Value counts and unique attributes

In [250]:

```
# number of unique values for categorcal vairbales
for col in categorical_df:
    print(f"{col:20}: {categorical_df[col].nunique()}")
```

```
term                : 2
grade               : 7
sub_grade           : 35
emp_title           : 173105
emp_length          : 11
home_ownership      : 6
verification_status : 3
issue_d             : 115
loan_status         : 2
purpose             : 14
title               : 48817
earliest_cr_line    : 684
initial_list_status : 2
application_type    : 3
address             : 393700
```

In [251]:

```
cat_unwanted = ('emp_title', 'issue_d', 'title', 'earliest_cr_line', 'address')
for column in categorical_df:
    if column not in cat_unwanted:
        print(categorical_df[column].value_counts().sort_values(ascending = False))
        print('\n')
```

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


B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: grade, dtype: int64


B3    26655
B4    25601
C1    23662
C2    22580
B2    22495
B5    22085
C3    21221
C4    20280
B1    19182
A5    18526
C5    18244
D1    15993
A4    15789
D2    13951
D3    12223
D4    11657
A3    10576
A1     9729
D5     9700
A2     9567
E1     7917
E2     7431
E3     6207
E4     5361
E5     4572
F1     3536
F2     2766
F3     2286
F4     1787
F5     1397
G1     1058
G2      754
G3      552
G4      374
G5      316
Name: sub_grade, dtype: int64


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
9 years       15314
Name: emp_length, dtype: int64


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


Verified           139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64


Fully Paid     318357
Charged Off     77673
Name: loan_status, dtype: int64


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


f    238066
w    157964
Name: initial_list_status, dtype: int64


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

Univariate analysis

For continuous variable(s): Boxplot, Distplot for univariate analysis

In [252]:

```
#numeric_df.columns
```

In [253]:

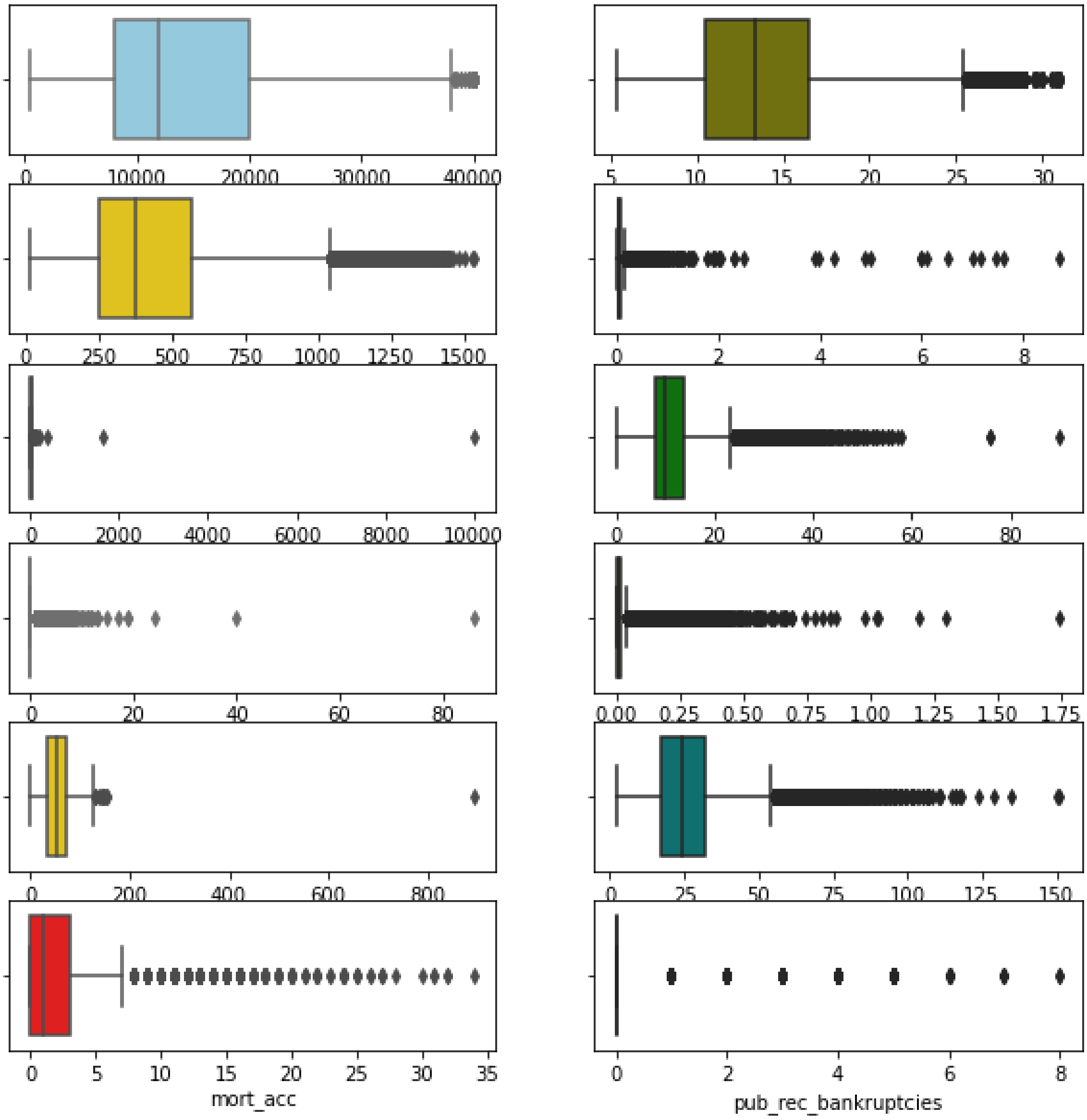
```
fig, axs = plt.subplots(6, 2, figsize=(10, 10))

sns.boxplot(data=numeric_df, x="loan_amnt", color="skyblue", ax=axs[0, 0])
sns.boxplot(data=numeric_df, x="int_rate", color="olive", ax=axs[0, 1])
sns.boxplot(data=numeric_df, x="installment", color="gold", ax=axs[1, 0])
sns.boxplot(data=numeric_df, x="annual_inc", color="teal", ax=axs[1, 1])
sns.boxplot(data=numeric_df, x="dti", color="red", ax=axs[2, 0])
sns.boxplot(data=numeric_df, x="open_acc", color="green", ax=axs[2, 1])

sns.boxplot(data=numeric_df, x="pub_rec", color="skyblue", ax=axs[3, 0])
sns.boxplot(data=numeric_df, x="revol_bal", color="olive", ax=axs[3, 1])
sns.boxplot(data=numeric_df, x="revol_util", color="gold", ax=axs[4, 0])
sns.boxplot(data=numeric_df, x="total_acc", color="teal", ax=axs[4, 1])
sns.boxplot(data=numeric_df, x="mort_acc", color="red", ax=axs[5, 0])
sns.boxplot(data=numeric_df, x="pub_rec_bankruptcies", color="green", ax=axs[5, 1])
```

Out[253]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a810fece50>
```



Histogram with distribution curve

In [254]:

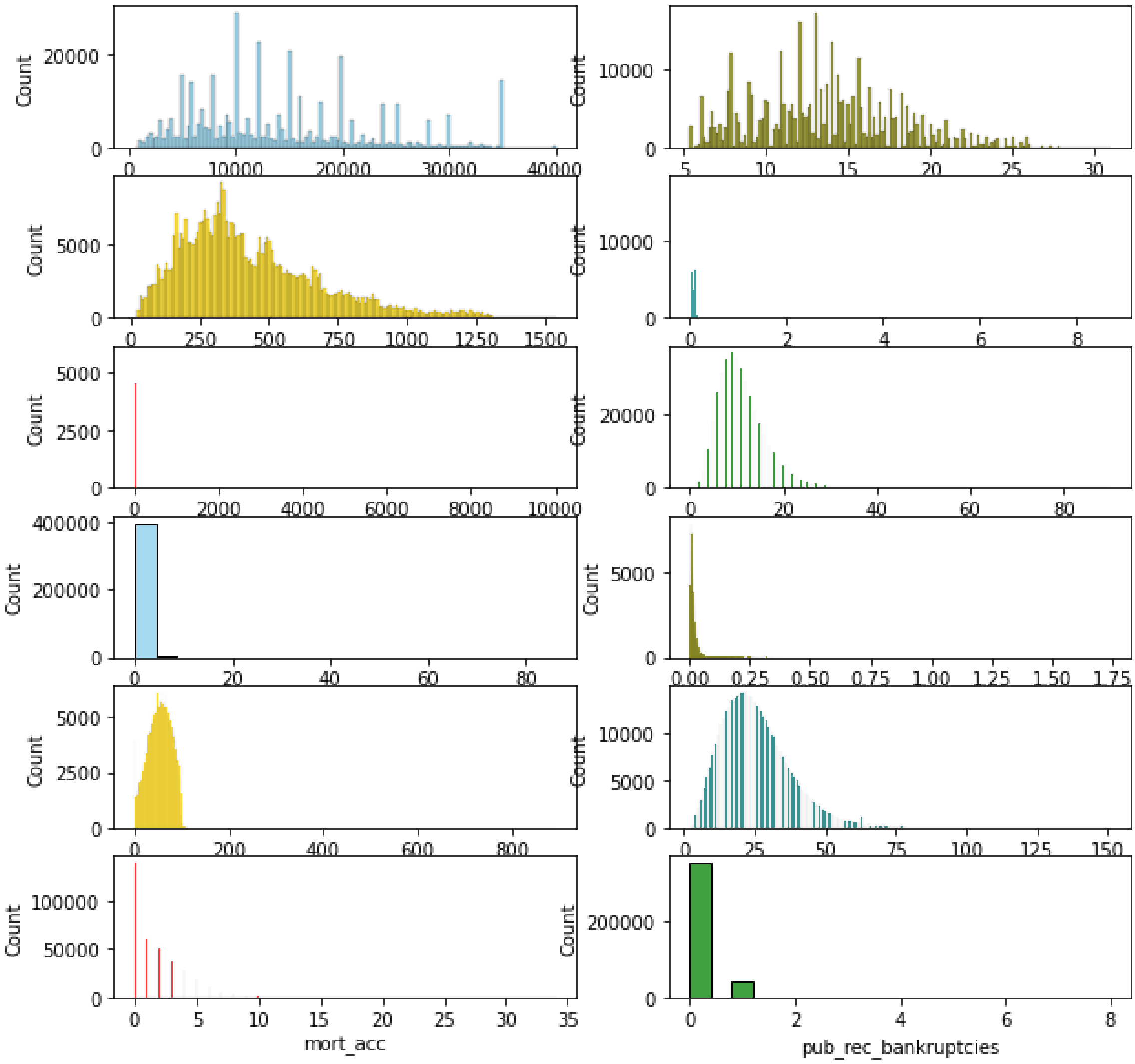
```
fig, axs = plt.subplots(6, 2, figsize=(10, 10))

sns.histplot(data=numeric_df, x="loan_amnt", color="skyblue", ax=axs[0, 0])
sns.histplot(data=numeric_df, x="int_rate", color="olive", ax=axs[0, 1])
sns.histplot(data=numeric_df, x="installment", color="gold", ax=axs[1, 0])
sns.histplot(data=numeric_df, x="annual_inc", color="teal", ax=axs[1, 1])
sns.histplot(data=numeric_df, x="dti", color="red", ax=axs[2, 0])
sns.histplot(data=numeric_df, x="open_acc", color="green", ax=axs[2, 1])

sns.histplot(data=numeric_df, x="pub_rec", color="skyblue", ax=axs[3, 0])
sns.histplot(data=numeric_df, x="revol_bal", color="olive", ax=axs[3, 1])
sns.histplot(data=numeric_df, x="revol_util", color="gold", ax=axs[4, 0])
sns.histplot(data=numeric_df, x="total_acc", color="teal", ax=axs[4, 1])
sns.histplot(data=numeric_df, x="mort_acc", color="red", ax=axs[5, 0])
sns.histplot(data=numeric_df, x="pub_rec_bankruptcies", color="green", ax=axs[5, 1])
```

Out[254]:

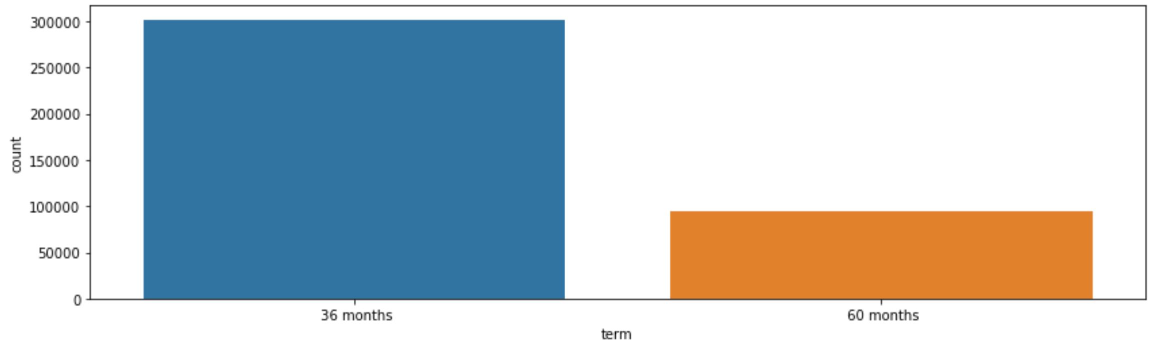
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a81151b5e0>
```

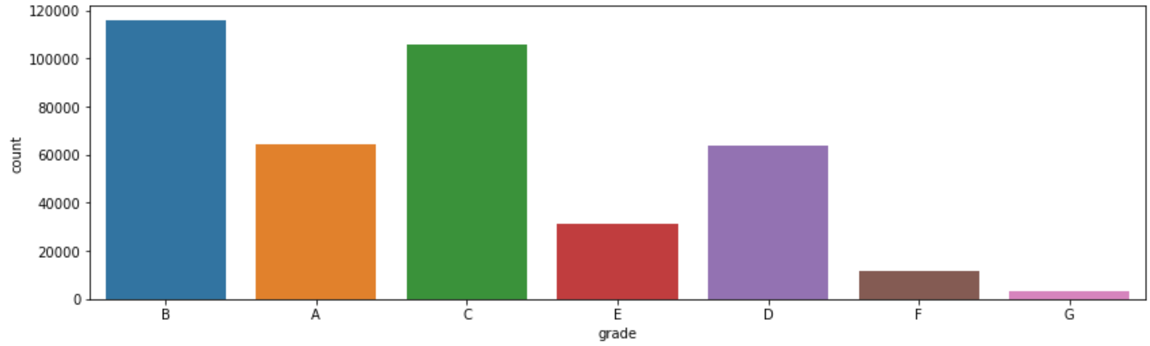


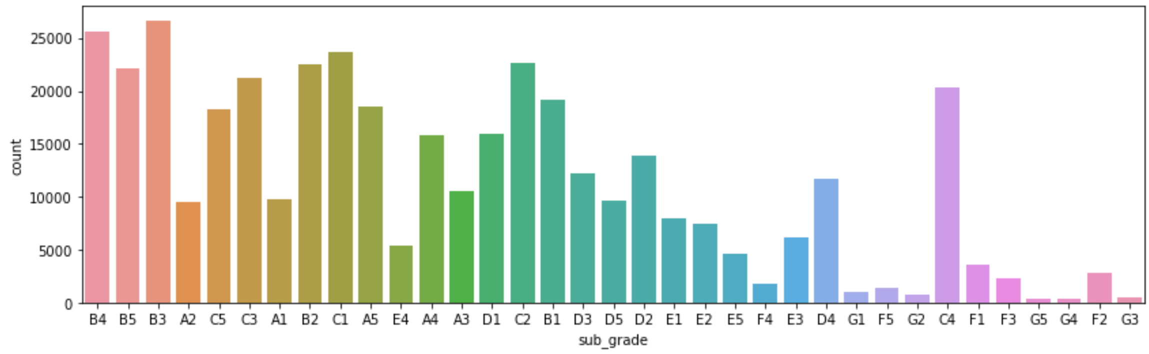
For categorical variable(s): countplot

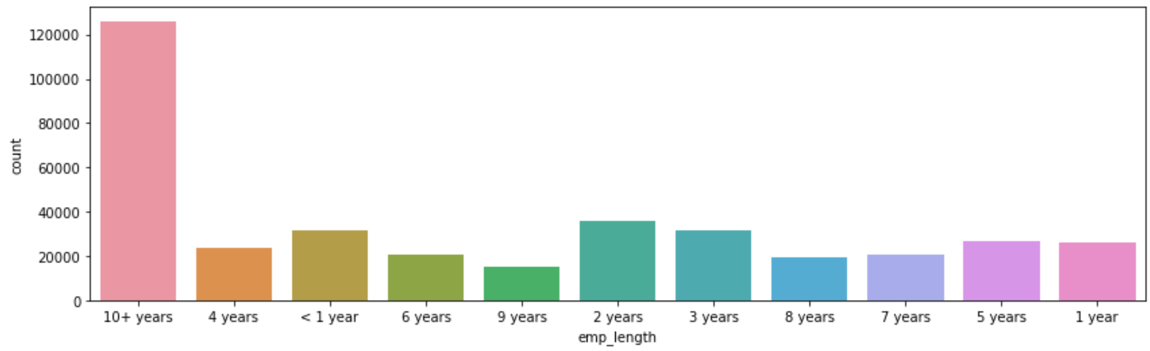
In [255]:

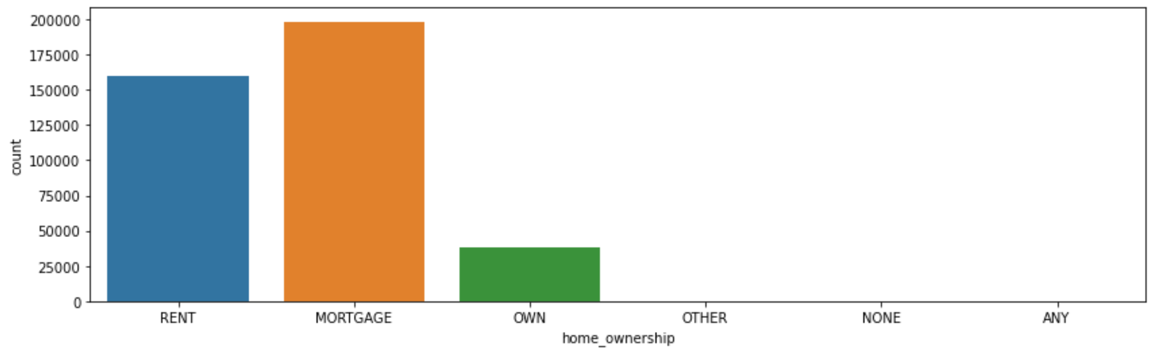
```
cat_unwanted = ('emp_title', 'issue_d', 'title', 'earliest_cr_line', 'address')
for col in categorical_df:
    if col not in cat_unwanted:
        plt.figure(figsize=(14,4))
        sns.countplot(data=categorical_df, x=col)
        plt.show()
```

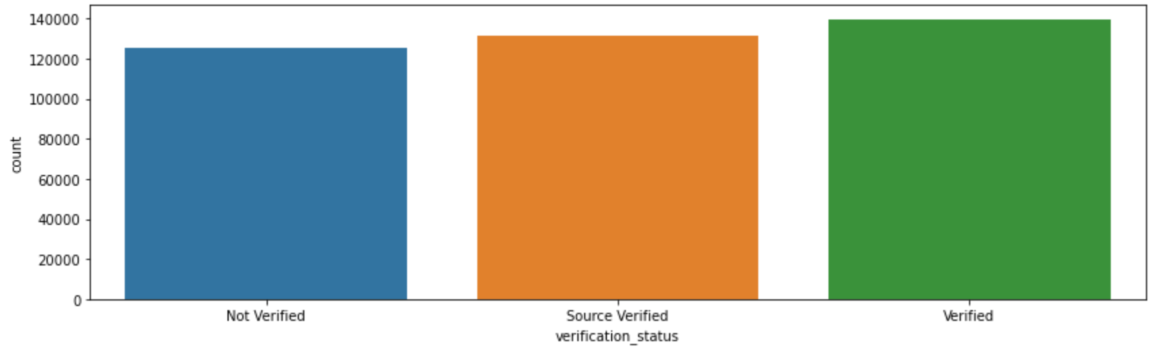

count
300000
250000
200000
150000
100000
50000
0
36 months
60 months
term

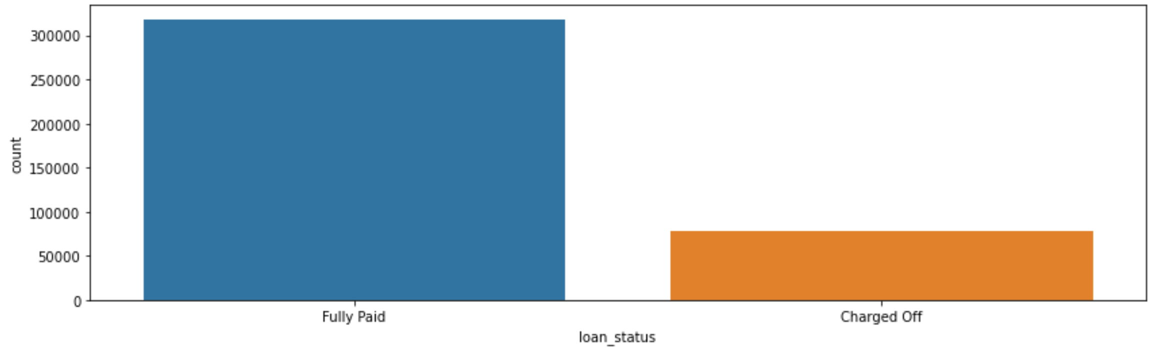

count
120000
100000
80000
60000
40000
20000
0
B
A
C
E
D
F
G
grade

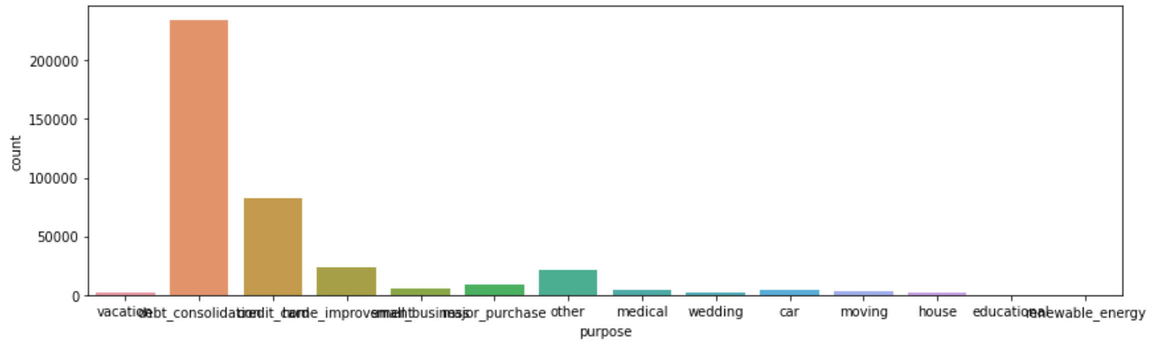

count
25000
20000
15000
10000
5000
0
B4 B5 B3 A2 C5 C3 A1 B2 C1 A5 E4 A4 A3 D1 C2 B1 D3 D5 D2 E1 E2 E5 F4 E3 D4 G1 F5 G2 C4 F1 F3 G5 G4 F2 G3
sub_grade

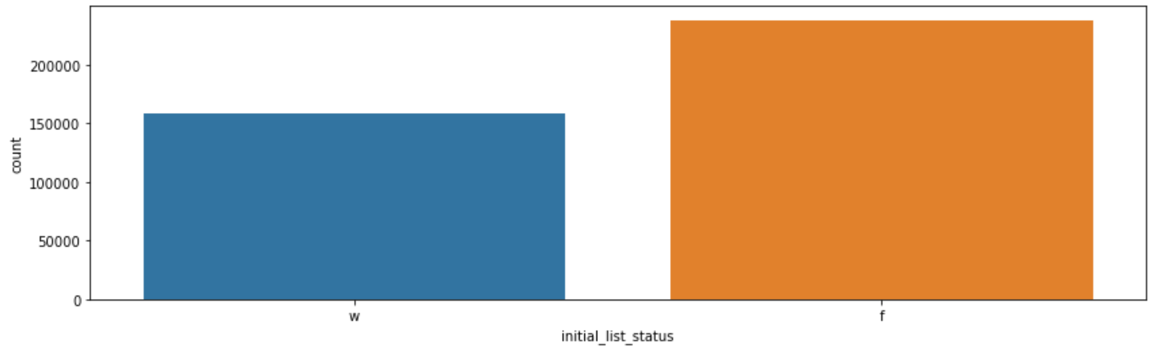

count
120000
100000
80000
60000
40000
20000
0
10+ years
4 years
< 1 year
6 years
9 years
2 years
3 years
8 years
7 years
5 years
1 year
emp_length

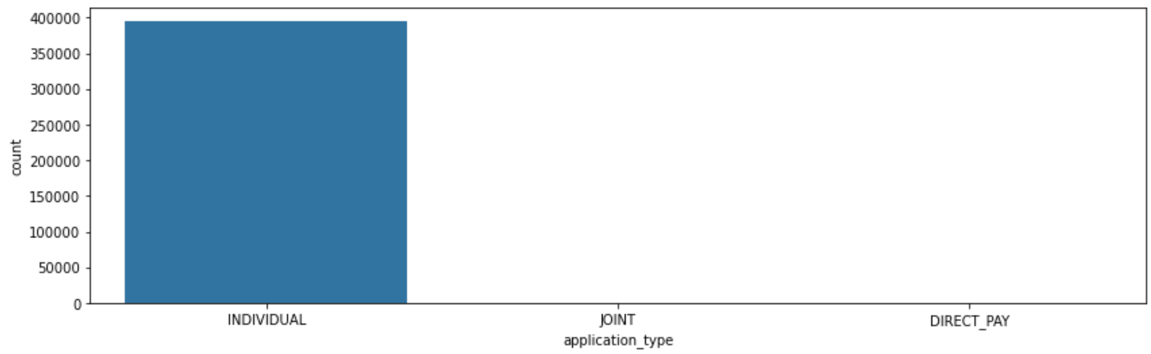

count
200000
175000
150000
125000
100000
75000
50000
25000
0
RENT
MORTGAGE
OWN
OTHER
NONE
ANY
home_ownership


140000
120000
100000
80000
60000
40000
20000
0
count
Not Verified
Source Verified
Verified
verification_status


300000
250000
200000
150000
100000
50000
0
count
Fully Paid
Charged Off
loan_status


200000
150000
100000
50000
0
count
other
medical
wedding
car
moving
house
renewable_energy
purpose


200000
150000
100000
50000
0
count
w
f
initial_list_status


400000
350000
300000
250000
200000
150000
100000
50000
0
count
INDIVIDUAL
JOINT
DIRECT_PAY
application_type

Bivariate Analysis

In [ ]:

```
# Continuos vs Continuous - Scatter plot/Heatmaps
```

In [256]:

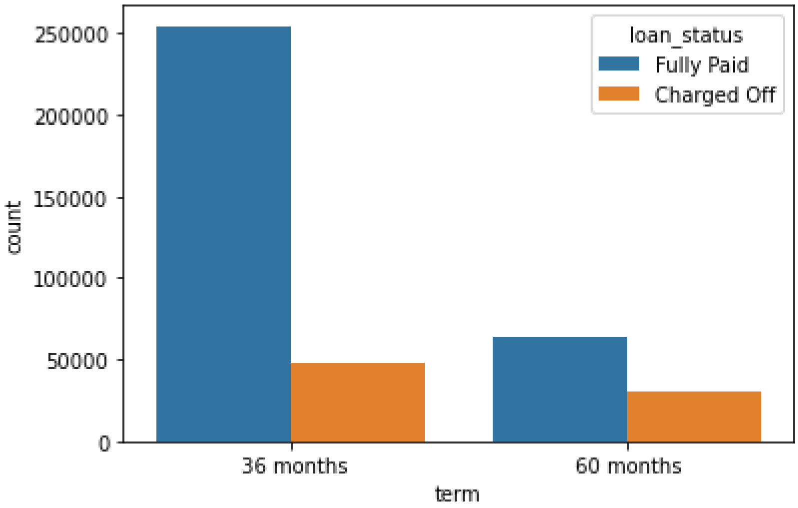
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(method='spearman'), annot=True, cmap='viridis')
plt.show()
```

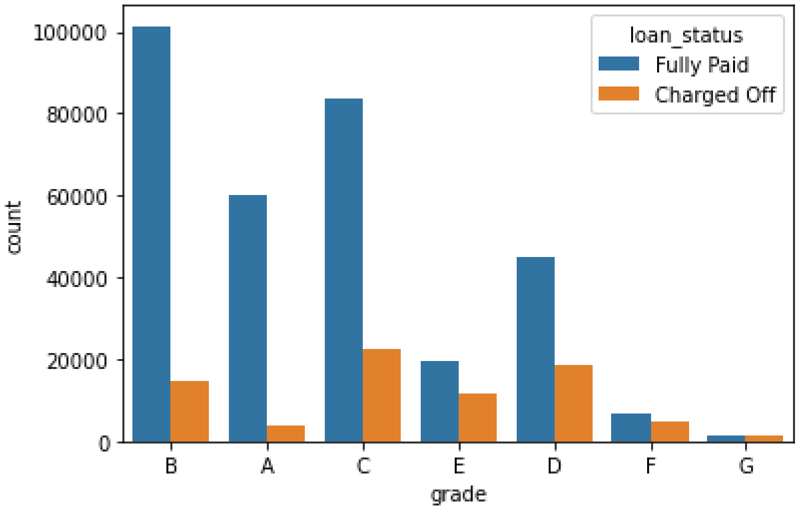
In [ ]:

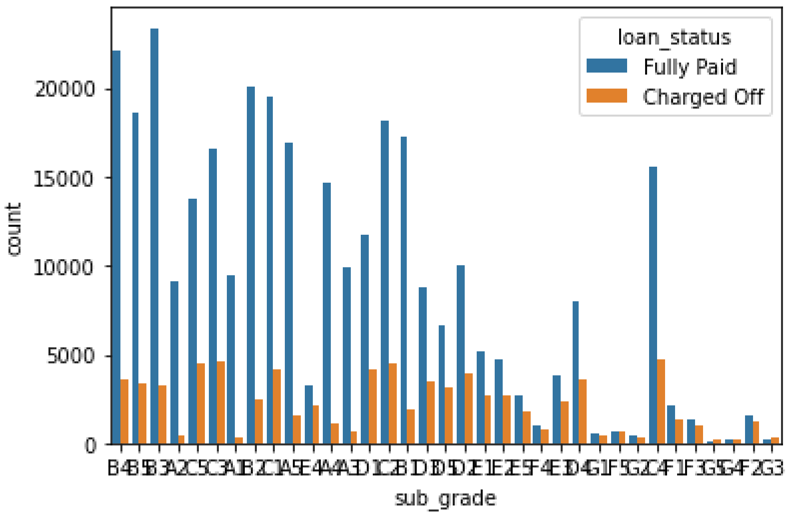
```
# Categorical attributes Bivariate analysis with target variable
```

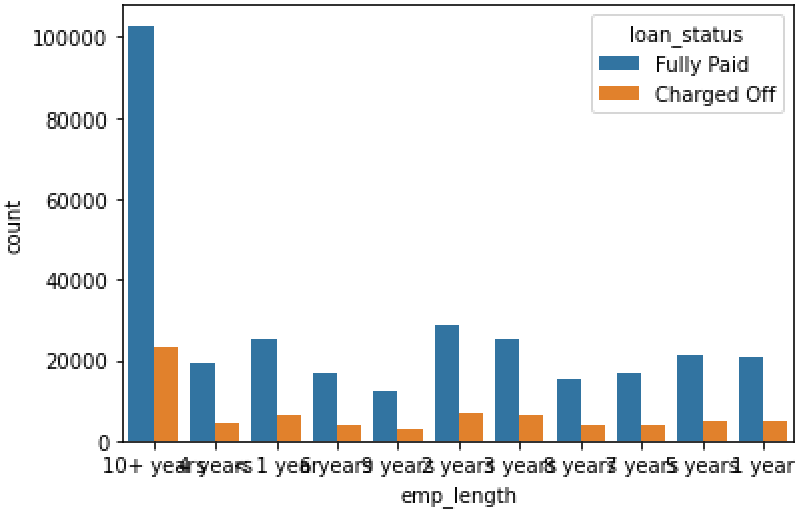
In [257]:

```
cat_unwanted = ('emp_title', 'issue_d', 'title', 'earliest_cr_line', 'address')
for column in categorical_df:
    if column not in cat_unwanted:
        sns.countplot(data=categorical_df, x=column, hue='loan_status')
        plt.show()
```

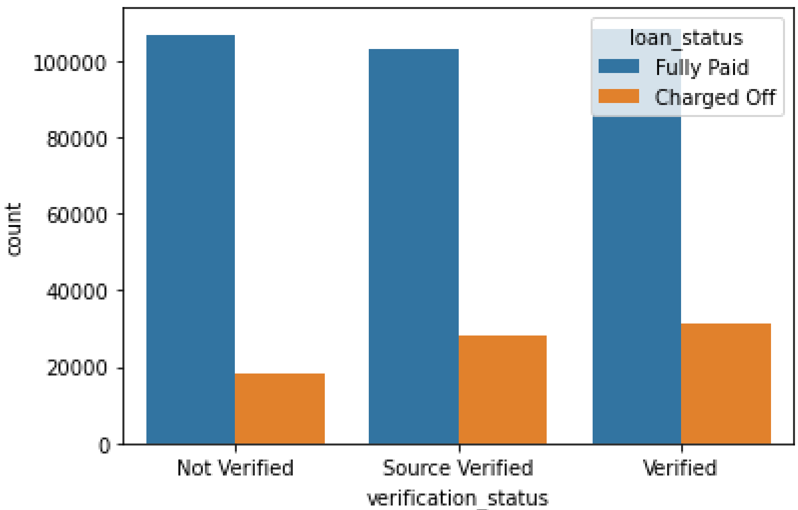

loan_status
Fully Paid
Charged Off
count
250000
200000
150000
100000
50000
0
36 months
60 months
term

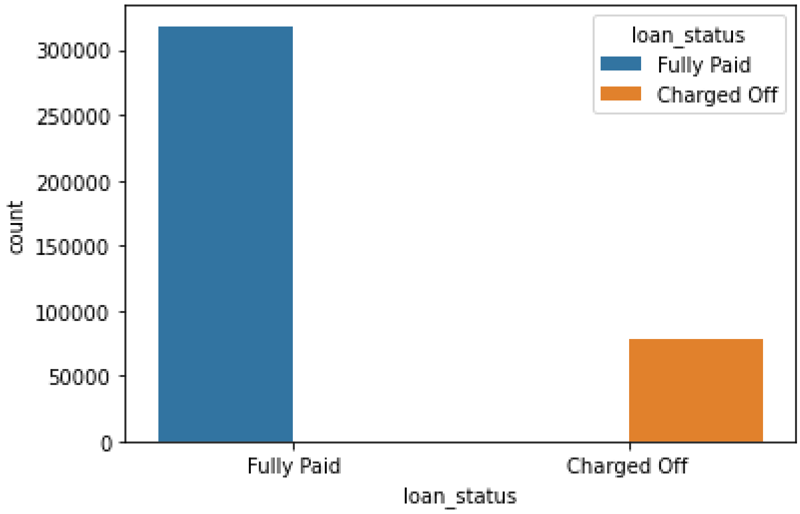

loan_status
Fully Paid
Charged Off
count
100000
80000
60000
40000
20000
0
B
A
C
E
D
F
G
grade

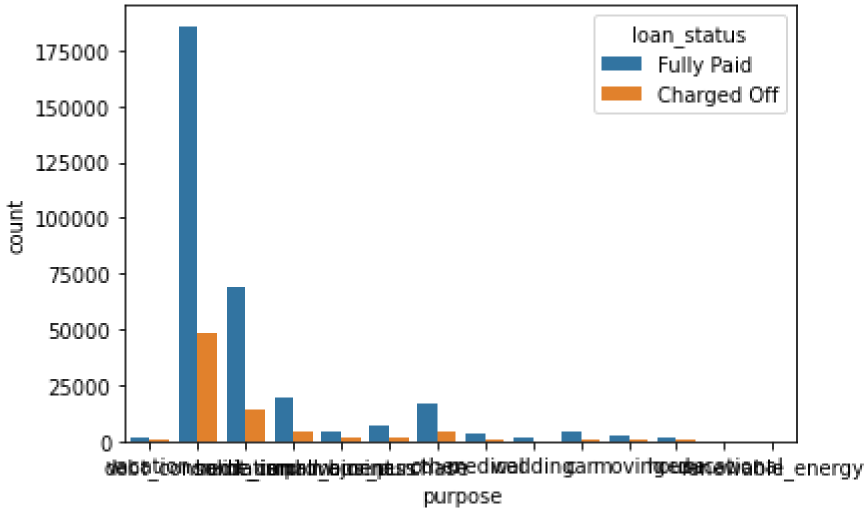

loan_status
Fully Paid
Charged Off
count
20000
15000
10000
5000
0
sub_grade

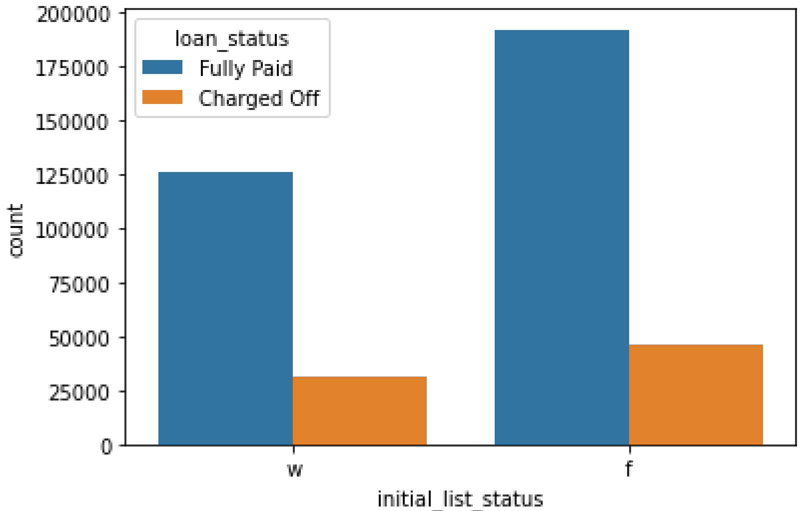

count
100000
80000
60000
40000
20000
0
loan_status
Fully Paid
Charged Off
emp_length

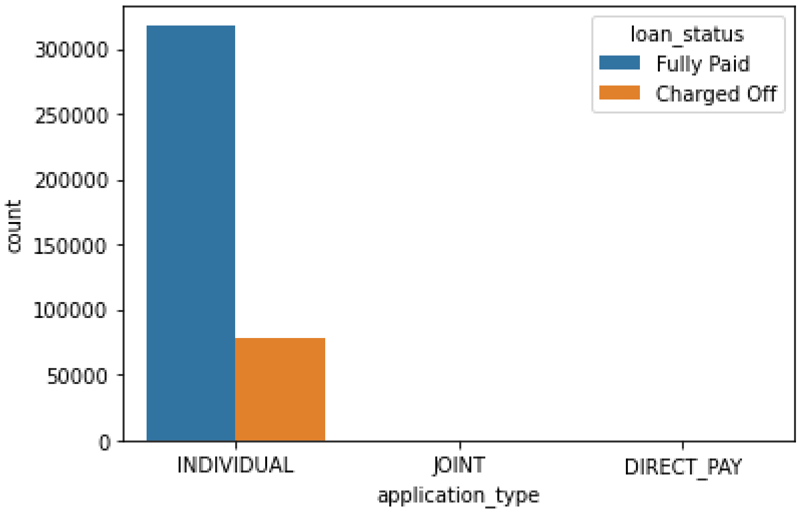

count
160000
140000
120000
100000
80000
60000
40000
20000
0
loan_status
Fully Paid
Charged Off
RENT
MORTGAGE
OWN
OTHER
NONE
ANY
home_ownership


count
100000
80000
60000
40000
20000
0
loan_status
Fully Paid
Charged Off
Not Verified
Source Verified
Verified
verification_status


loan_status
Fully Paid
Charged Off
count
300000
250000
200000
150000
100000
50000
0
Fully Paid
Charged Off
loan_status


loan_status
Fully Paid
Charged Off
count
175000
150000
125000
100000
75000
50000
25000
0
purpose


loan_status
Fully Paid
Charged Off
count
200000
175000
150000
125000
100000
75000
50000
25000
0
w
f
initial_list_status



Data Preprocessing

Duplicate value check

In [258]:

```
df.duplicated().sum()
```

Out[258]:

```
0
```

In [ ]:

```
# There are no duplicate rows present in the dataset.
```

Simple Feature Engineering

In [259]:

```
df['pub_rec'] = [1 if x > 1.0 else 0 for x in df['pub_rec']]
df['mort_acc'] = [1 if x > 1.0 else 0 for x in df['mort_acc']]
df['pub_rec_bankruptcies'] = [1 if x > 1.0 else 0 for x in df['pub_rec_bankruptcies']]
```

Missing value treatment

In [260]:

```
df.isnull().sum()
```

Out[260]:

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                    0
pub_rec_bankruptcies        0
address                     0
dtype: int64
```

In [261]:

```
df['emp_title'] = df['emp_title'].fillna("unknown_emp_title")
df['emp_length'] = df['emp_length'].fillna("unknown_emp_length")
df['title'] = df['mort_acc'].fillna("unknown_title")
df['revol_util'] = df['revol_util'].fillna(0.0)
```

In [262]:

```
df.shape
```

Out[262]:

```
(396030, 27)
```

Outlier treatment

In [263]:

```
#df[df['annual_inc'] >= 30000]
```

Annual income shouldn't be less than 30,000. So let's remove all the rows where income < 30,000. Outliers are removed using iqr method.

In [264]:

```
print(f"Shape before: {df.shape}")
df = df[df['annual_inc'] >= 30000]
print(f"Shape after: {df.shape}")
```

```
Shape before: (396030, 27)
Shape after: (373057, 27)
```

In [265]:

```
#df.skew()
```

In [266]:

```
def check_outliers(num_columns, df):
    # check for outliers
    for col in num_columns:
        q1 = np.percentile(df[col], 25)
        q3 = np.percentile(df[col], 75)
        iqr = q3-q1
        outliers = len(df) - len(df[(df[col]>=(q1-1.5*iqr)) & (df[col]<=(q3+1.5*iqr))])
        print(f"{col:20}: {round(outliers*100/len(df), 6)}")

check_outliers(numeric_df.columns, df)
```

```
loan_amnt           : 0.051199
int_rate            : 0.392969
installment         : 2.76124
annual_inc          : 4.876199
dti                 : 0.061653
open_acc            : 2.728001
pub_rec             : 2.046604
revol_bal           : 5.245579
revol_util          : 0.003217
total_acc           : 1.559279
mort_acc            : 0.0
pub_rec_bankruptcies: 0.589722
```

In [267]:

```
# remove outliers using IQR method
print(f"Shape before: {df.shape}")
for col in numeric_df.columns:
    q1 = np.percentile(df[col], 25)
    q3 = np.percentile(df[col], 75)
    iqr = q3-q1
    df = df[(df[col] >= (q1-1.5*iqr)) & (df[col] <= (q3+1.5*iqr))]

print(f"Shape after: {df.shape}")
```

```
Shape before: (373057, 27)
Shape after: (309469, 27)
```

Feature engineering

In [268]:

```
def get_term_month(text):
    """extract the month from term"""
    nums = re.findall("[0-9]+", text)
    if len(nums)>0:
        return nums[0]
    return 0

def pre_emp_title(text):
    """Pre-process emp_title"""
    text = str(text).lower()
    text = re.sub("[^a-z ]", " ", text)
    text = re.sub(" +", " ", text)
    return text.strip()
```

In [269]:

```
df['term'] = df['term'].apply(get_term_month)
df['emp_title'] = df['emp_title'].apply(pre_emp_title)
df['home_ownership'] = df['home_ownership'].apply(pre_emp_title)
```

In [270]:

```
# get PIN and city code from the address
def get_pin(text):
    text = str(text).split(",")[-1]
    pin = re.findall("[0-9]+", text)
    if len(pin)>0:
        return pin[0]
    return 0

def get_city_code(text):
    text = str(text).split(",")[-1]
    res = re.findall("[a-z]+", text)
    if len(res)>0:
        return res[0]
    return "unk"
```

In [271]:

```
df['address_pincode'] = df['address'].apply(get_pin)
df['address_city_code'] = df['address'].apply(get_city_code)
```

In [272]:

```
# get year and month from the following columns
# - issue_d
# - earliest_cr_line

df['issue_d_year'] = pd.to_datetime(df['issue_d']).dt.year
df['issue_d_month'] = pd.to_datetime(df['issue_d']).dt.month

df['earliest_cr_line_year'] = pd.to_datetime(df['earliest_cr_line']).dt.year
df['earliest_cr_line_month'] = pd.to_datetime(df['earliest_cr_line']).dt.month
```

In [273]:

```
# drop the following columns
# address, issue_d, earliest_cr_line

cols_to_drop = ['address', 'issue_d', 'earliest_cr_line']
df.drop(columns=cols_to_drop, axis=1, inplace=True)
```

Data preparation for modeling

In [274]:

```
one_hot_cols = ['term', 'grade', 'sub_grade', 'emp_length', 'home_ownership', 'verifica
tion_status',
                'purpose', 'initial_list_status', 'application_type']
target_encoding_cols = ['emp_title', 'address_pincode', 'address_city_code']
```

In [275]:

```
# one hot encoding
newdf = pd.get_dummies(df[one_hot_cols], drop_first=True)
```

In [276]:

```
df = pd.concat([newdf, df], axis=1)
df.drop(columns=one_hot_cols, axis=1, inplace=True)
```

In [277]:

```
df['loan_status'] = df['loan_status'].replace({'Fully Paid': 1, 'Charged Off': 0})
```

In [278]:

```
# target encoding
df['emp_title'] = df['emp_title'].map(df.groupby('emp_title')['loan_status'].mean())
df['address_pincode'] = df['address_pincode'].map(df.groupby('address_pincode')['loan_s
tatus'].mean())
df['address_city_code'] = df['address_city_code'].map(df.groupby('address_city_code')
['loan_status'].mean())
```

In [279]:

```
X=df.drop('loan_status',axis=1)
y=df['loan_status']
```

In [280]:

```
X1 = X.copy()
y1 = y.copy()
```

In [281]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X1_scaled = scaler.fit_transform(X1)
```

In [282]:

```
features = X.columns.tolist()
```

In [283]:

```
# standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [284]:

```
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.30,stratify=y,random
_state=42)
```

In [285]:

```
X1_train, X1_test, y1_train, y1_test =train_test_split(X1,y1,test_size=0.30,stratify=y
1,random_state=42)
```

In [286]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(216628, 95)
(92841, 95)
```

In [287]:

```
print(y_train.shape)
print(y_test.shape)
```

```
(216628,)
(92841,)
```

Model building

In [288]:

```
logreg=LogisticRegression()
logreg.fit(X_train,y_train)
```

Out[288]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [289]:

```
y_pred = logreg.predict(X_test)
print('Accuracy of Logistic Regression Classifier on test set: {:.3f}'.format(logreg.sc
ore(X_test, y_test)))
```

```
Accuracy of Logistic Regression Classifier on test set: 0.929
```

Display model coefficients

In [290]:

```
coefs = logreg.coef_.tolist()[0]
feature_coef_df = pd.DataFrame({'Variable': features, 'Coeficient': coefs})
feature_coef_df.sort_values(by=['Coeficient'], ascending=False)
```

Out[290]:

	Variable	Coeficient
89	address_pincode	2.909302
78	emp_title	2.063341
76	int_rate	0.411406
90	address_city_code	0.117730
75	loan_amnt	0.099506
86	total_acc	0.084778
84	revol_bal	0.068441
52	home_ownership_mortgage	0.065932
72	initial_list_status_w	0.044239
92	issue_d_month	0.039160
79	annual_inc	0.037552
41	emp_length_10+ years	0.027016
54	home_ownership_other	0.026939
80	title	0.017373
87	mort_acc	0.017373
44	emp_length_4 years	0.016128
48	emp_length_8 years	0.014812
71	purpose_wedding	0.013978
45	emp_length_5 years	0.009450
93	earliest_cr_line_year	0.009071
55	home_ownership_own	0.003363
49	emp_length_9 years	0.002017
46	emp_length_6 years	0.002014
43	emp_length_3 years	0.000951
83	pub_rec	0.000000
88	pub_rec_bankruptcies	0.000000
42	emp_length_2 years	-0.000564
47	emp_length_7 years	-0.002410
63	purpose_house	-0.002888
94	earliest_cr_line_month	-0.004157
53	home_ownership_none	-0.012512
61	purpose_educational	-0.012530
39	sub_grade_G4	-0.015549
40	sub_grade_G5	-0.017014
50	emp_length_< 1 year	-0.017997
68	purpose_renewable_energy	-0.021513
70	purpose_vacation	-0.030556

	Variable	Coeficient
66	purpose_moving	-0.031613
67	purpose_other	-0.039781
65	purpose_medical	-0.039870
64	purpose_major_purchase	-0.056974
38	sub_grade_G3	-0.057862
58	verification_status_Verified	-0.058600
69	purpose_small_business	-0.059030
11	sub_grade_B1	-0.065803
62	purpose_home_improvement	-0.066156
91	issue_d_year	-0.070155
7	sub_grade_A2	-0.073106
74	application_type_JOINT	-0.076032
56	home_ownership_rent	-0.078599
37	sub_grade_G2	-0.079304
36	sub_grade_G1	-0.083471
57	verification_status_Source Verified	-0.085865
59	purpose_credit_card	-0.097011
8	sub_grade_A3	-0.098203
51	emp_length_unknown_emp_length	-0.099208
73	application_type_INDIVIDUAL	-0.105318
12	sub_grade_B2	-0.107733
35	sub_grade_F5	-0.115380
6	grade_G	-0.130554
60	purpose_debt_consolidation	-0.131878
34	sub_grade_F4	-0.138847
85	revol_util	-0.142730
33	sub_grade_F3	-0.150487
32	sub_grade_F2	-0.153015
82	open_acc	-0.156747
31	sub_grade_F1	-0.161449
9	sub_grade_A4	-0.167805
81	dti	-0.181134
16	sub_grade_C1	-0.181491
13	sub_grade_B3	-0.183059
30	sub_grade_E5	-0.192947
26	sub_grade_E1	-0.194945
77	installment	-0.195795
29	sub_grade_E4	-0.202549
28	sub_grade_E3	-0.204297

	Variable	Coeficient
17	sub_grade_C2	-0.206004
14	sub_grade_B4	-0.217889
27	sub_grade_E2	-0.226320
15	sub_grade_B5	-0.227920
25	sub_grade_D5	-0.228312
10	sub_grade_A5	-0.236445
24	sub_grade_D4	-0.236499
23	sub_grade_D3	-0.237891
18	sub_grade_C3	-0.242607
22	sub_grade_D2	-0.244108
21	sub_grade_D1	-0.244891
19	sub_grade_C4	-0.266034
20	sub_grade_C5	-0.267627
0	term_60	-0.270710
5	grade_F	-0.326504
1	grade_B	-0.421427
4	grade_E	-0.470403
3	grade_D	-0.569594
2	grade_C	-0.587094

Model Evaluation

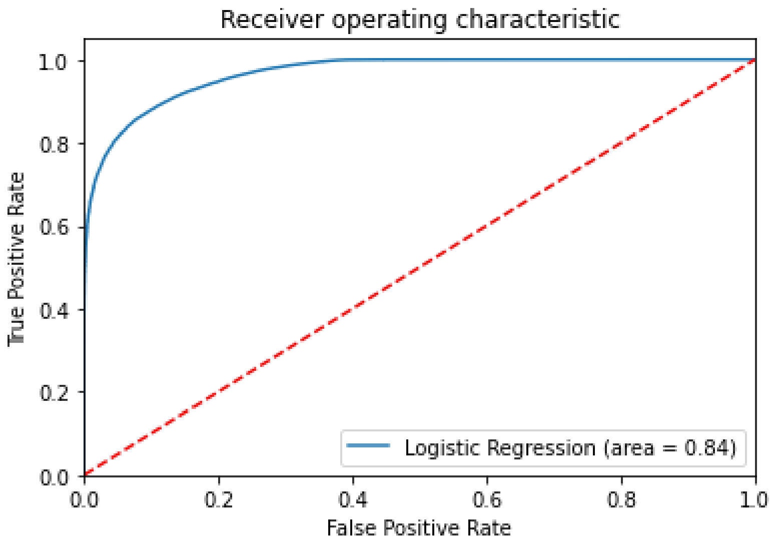
ROC AUC Curve

In [291]:

```
from sklearn.metrics import roc_auc_score,roc_curve,precision_recall_curve,confusion_ma
trix,classification_report,auc
```

In [292]:

```
logit_roc_auc=roc_auc_score(y_test,logreg.predict(X_test))
fpr,tpr,thresholds=roc_curve(y_test,logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr,tpr,label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [293]:

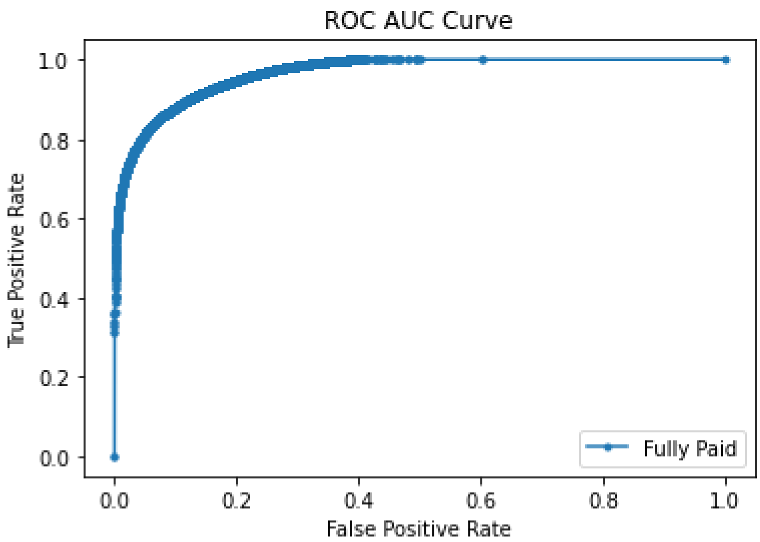
```
y_preds_prob = logreg.predict_proba(X_test)[:, -1]

fpr, tpr, threshold = roc_curve(y_test, y_preds_prob)
score = roc_auc_score(y_test, y_preds_prob)

print(f"ROC AUC Score: {score}")

#plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='Fully Paid')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
plt.title("ROC AUC Curve")
# show the plot
plt.show()
```

```
ROC AUC Score: 0.9662402886311695
```



Precision Recall Curve

In [294]:

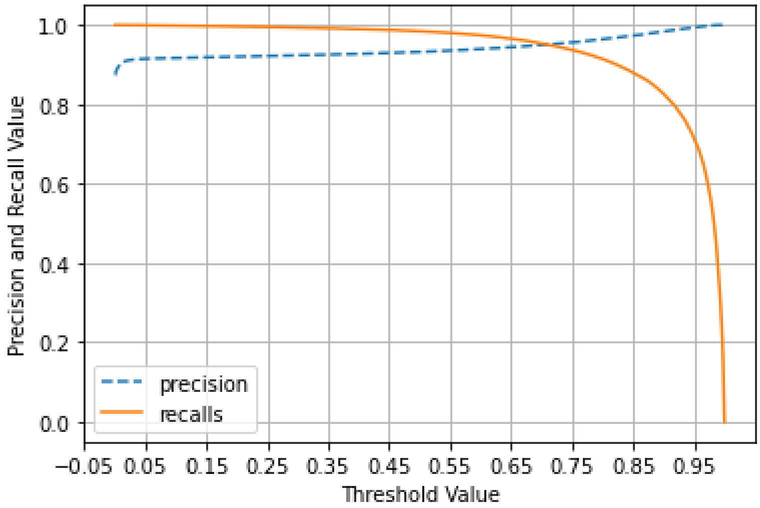
```
def precission_recall_curve_plot(y_test,pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test,pred_proba_c1)

    threshold_boundary = thresholds.shape[0]
    #plot precision
    plt.plot(thresholds,precisions[0:threshold_boundary],linestyle='--',label='precisio
n')
    #plot recall
    plt.plot(thresholds,recalls[0:threshold_boundary],label='recalls')

    start,end=plt.xlim()
    plt.xticks(np.round(np.arange(start,end,0.1),2))

    plt.xlabel('Threshold Value')
    plt.ylabel('Precision and Recall Value')
    plt.legend()
    plt.grid()
    plt.show()

precission_recall_curve_plot(y_test,logreg.predict_proba(X_test)[:,1])
```



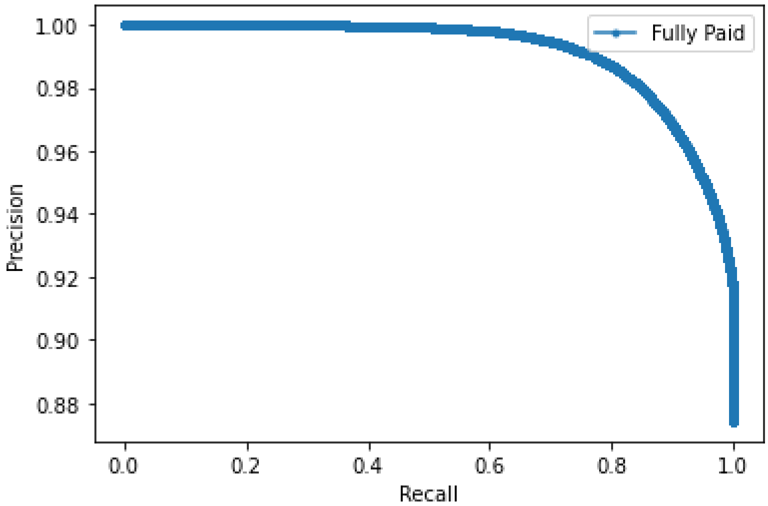
In [295]:

```
precision, recall, threshold = precision_recall_curve(y_test, y_preds_prob)
score = auc(recall, precision)

print(f"AUC Score: {score}")

#plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(recall, precision, marker='.', label='Fully Paid')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
# show the legend
plt.legend()
# show the plot
plt.show()
```

```
AUC Score: 0.991369938170823
```



Classification Report

In [296]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.70      0.79     17862
           1       0.93      0.98      0.96     74979

    accuracy                           0.93     92841
   macro avg       0.92      0.84      0.87     92841
weighted avg       0.93      0.93      0.93     92841
```

Confusion Matrix

In [297]:

```
confusion_matrix=confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

```
[[12482  5380]
 [ 1184 73795]]
```

Questionnaire

1. What percentage of customers have fully paid their Loan Amount?

In [298]:

```
df1['loan_status'].value_counts(normalize = True)
```

Out[298]:

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

Around 80.38% of customers have fully paid their Loan Amount.

1. Comment about the correlation between Loan Amount and Installment features.

In [299]:

```
df1['loan_amnt'].corr(df1['installment'], method='spearman')
```

Out[299]:

```
0.9683337077962306
```

The correlation between loan_amnt and installment features is very high i.e., 0.96. This is very much expected since as the loan amount value increases installment value will be higher.

1. The majority of people have home ownership as ___.

In [300]:

```
df1.columns
```

Out[300]:

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grad
e', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verificati
on_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_bankruptci
es', 'address'], dtype='object')
```

In [301]:

```
df1['home_ownership'].value_counts(normalize = True)
```

Out[301]:

```
MORTGAGE    0.500841
RENT        0.403480
OWN         0.095311
OTHER       0.000283
NONE        0.000078
ANY         0.000008
Name: home_ownership, dtype: float64
```

The majority of people have home ownership as Mortgage (50%)

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

In [303]:

```
df1[df1['grade']== 'A']['loan_status'].value_counts()
```

Out[303]:

```
Fully Paid     60151
Charged Off     4036
Name: loan_status, dtype: int64
```

True. People with grade A are more likely to fully pay their loan.

1. Name the top 2 afforded job titles.

In [304]:

```
df1[['emp_title','loan_status']].value_counts()
```

Out[304]:

```
emp_title                                           loan_status
Teacher                                             Fully Paid     3532
Manager                                             Fully Paid     3321
Registered Nurse                                    Fully Paid     1476
RN                                                  Fully Paid     1467
Supervisor                                          Fully Paid     1425
                                                                   ...
Hunter Truck Sales                                  Fully Paid        1
Hunterdon County Educational Services Commission    Fully Paid        1
Hunterdon Developmental Center                      Fully Paid        1
Hunterdon Healthcare supportive Services            Fully Paid        1
License Compliance Investigator                    Fully Paid        1
Length: 185292, dtype: int64
```

Teacher and Manager are the top 2 afforded job titles

1. Thinking from a bank's perspective, which metric should our primary focus be on..

```
ROC AUC
Precision
Recall
F1 Score
```

The best metric to consider is F1 score We need to give importance to both precision and recall as we don't want to miss potential customers and at the same time we also don't want to give loan to defaulters

1. How does the gap in precision and recall affect the bank?

In [305]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.70      0.79     17862
           1       0.93      0.98      0.96     74979

    accuracy                           0.93     92841
   macro avg       0.92      0.84      0.87     92841
weighted avg       0.93      0.93      0.93     92841
```

=> Recall score: 0.98 and Precision score: 0.93. Which tells us that there are more false positives than the false negatives.

=> If Recall value is low (i.e. FN are high), it means Bank is loosing in opportunity cost.

=> If Precision value is low (i.e. FP are high), it means Bank's NPA (defaulters) may increase.

1. Which were the features that heavily affected the outcome?

In [306]:

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
```

In [307]:

```
rfe_method = RFE(
    RandomForestClassifier(n_estimators=10, random_state=10),
    n_features_to_select=10,
    step=2,
)
```

In [308]:

```
rfe_method.fit(X1_train, y1_train)
```

Out[308]:

```
RFE(estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                     class_weight=None, criterion='gini',
                                     max_depth=None, max_features='auto',
                                     max_leaf_nodes=None, max_samples=Non
e,
                                     min_impurity_decrease=0.0,
                                     min_impurity_split=None,
                                     min_samples_leaf=1, min_samples_split
=2,
                                     min_weight_fraction_leaf=0.0,
                                     n_estimators=10, n_jobs=None,
                                     oob_score=False, random_state=10,
                                     verbose=0, warm_start=False),
    n_features_to_select=10, step=2, verbose=0)
```

In [309]:

```
X1_train.columns[(rfe_method.get_support())]
```

Out[309]:

```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_title', 'annual_inc',
'dti', 'revol_bal', 'revol_util', 'total_acc', 'address_pincode'], dtype
='object')
```

1. Will the results be affected by geographical location? (Yes/No)

Yes, address_pincode feature engineered from 'address' varibale has significant impact on the outcome based on the analysis

Tradeoff Questions

1. How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.

- To keep very less False Positives, oversampling techniques like SMOTE should be used in model creation. Also we can use more advanced algorithms like SVM, Decision-Trees, Random Forest and also we can try various hyperparameter tunning.
- As you can see from the data, the percentage of defaulters is slightly higher than Banking industry.

1. 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

- Yes. LoanTap should not disburse loans to everyone. Company's internal policy and analysis should be in place to identify the correct persons. From data provided, 20% of people default on their loan, which inturn become NPAs for the company.
- Low False positive means we should create the model with high Precision values. This can be achieved if we are keeping high threshold value in logistic Regression model.
- But keeping too high values for threshold will increase False Negatives. This intuen may result in opportunity loss. In this case we will not give loans to persons which will not default but our model has predicted that they will default.

Insights and Recommendations

Around 80% of customers have fully paid their Loan Amount. The defaulters are ~ 20%. From Personal loan business perspective this ratio is high. These 20% will contribute in NPAs of LoanTap. To reduce the risk of NPAs, LoanTap should add slightly stringent rules to bring down this ratio to 5% to 6%. LoanTap should provide loans at slightly higher rate than other Banks. This will offset the risks of defaulters and maintain the profitability of the business. Overall Statistics of the Model: Accuracy = 93% Precision = 93% Recall = 98% F1 -score = 96% Model created has high values for accuracy, precision, recall & f1-score. This means, this model is a good classifier. Overall, it has good prediction capability in identifying right customers (which can be easily converted). However this model has slightly low capability on correctly identifying defaulters. Overall data has 20% defaulters, model is able to predict 10% of them correctly. Using this model, LoanTap can easily reduce the ration of defaulters in their portfolio. Features which have significant impact on outcome are as follows: 'loan_amnt', 'int_rate', 'installment', 'emp_title', 'annual_inc', 'dti', 'revol_bal', 'revol_util', 'total_acc', 'address_pincode' Based on the analysis, following suggestions are given. LoanTap can also decide their social media based marketing based on person's job-titles. LoanTap can promote persons to apply for joint loan. Because of this, chances of default will reduce. LoanTap should stick to giving loans to conventional purposes like Marriage, car etc. LoanTap should focus more on Loans for shorter duration (i.e. 36 months). Their social media campaign and marketing strategy should be based on this consideration. Pincode based market segmentation should be included at strategic levels.