About LoanTap

LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen.

LoanTap deploys formal credit to salaried individuals and businesses 4 main financial instruments:

Personal Loan EMI Free Loan Personal Overdraft Advance Salary Loan This case study will focus on the underwriting process behind Personal Loan only

Problem Statement

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

Importing Libraries

In [56]:
```
import pandas as pd
import numpy as np

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline
pd.set_option('display.max_columns', None)


from scipy import stats
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (accuracy_score, confusion_matrix,
                             roc_curve, auc, ConfusionMatrixDisplay,
                             f1_score, recall_score,
                             precision_score, precision_recall_curve,
                             average_precision_score, classification_report
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE

#Hide warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets

df.head(5)
```

Out[2]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	h
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 [3]:
```
df.shape
```

Out[3]:
```
(396030, 27)
```

In [4]:
```
df.describe()
```

Out[4]:

	loan_amnt	int_rate	installment	annual_inc	dti	
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396(
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	

In [5]:
```
df.describe(include = 'object')
```

Out[5]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_s
count	396030	396030	396030	373103	377729	396030	39
unique	2	7	35	173105	11	6	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	V
freq	302005	116018	26655	4389	126041	198348	1:

In [6]:
```
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                 394274 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 [7]:
```
df.dtypes
```

Out[7]:
```
loan_amnt                float64
term                      object
int_rate                 float64
installment              float64
grade                     object
sub_grade                 object
emp_title                 object
emp_length                object
home_ownership            object
annual_inc               float64
verification_status       object
issue_d                   object
loan_status               object
purpose                   object
title                     object
dti                      float64
earliest_cr_line          object
open_acc                 float64
pub_rec                  float64
revol_bal                float64
revol_util               float64
total_acc                float64
initial_list_status       object
application_type          object
mort_acc                 float64
pub_rec_bankruptcies     float64
address                   object
dtype: object
```

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
0
```

Dataset has no duplicate values

Checking Column Datatypes

In [9]:
```
# Non-numeric columns
cat_cols = df.select_dtypes(include='object').columns
cat_cols
```

Out[9]:
```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'title', 'earliest_cr_line', 'initial_list_status',
       'application_type', 'address'],
      dtype='object')
```

In [10]:
```
# Number of unique values in all non-numeric columns
for col in cat_cols:
  print(f"No. of unique values in {col}: {df[col].nunique()}")
```

```
No. of unique values in term: 2
No. of unique values in grade: 7
No. of unique values in sub_grade: 35
No. of unique values in emp_title: 173105
No. of unique values in emp_length: 11
No. of unique values in home_ownership: 6
No. of unique values in verification_status: 3
No. of unique values in issue_d: 115
No. of unique values in loan_status: 2
No. of unique values in purpose: 14
No. of unique values in title: 48816
No. of unique values in earliest_cr_line: 684
No. of unique values in initial_list_status: 2
No. of unique values in application_type: 3
No. of unique values in address: 393700
```

In [11]:

```
# Convert earliest credit line & issue date to datetime
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
df['issue_d'] = pd.to_datetime(df['issue_d'])
```

In [12]:

```
#Convert employment length to numeric
d = {'10+ years':10, '4 years':4, '< 1 year':0,
     '6 years':6, '9 years':9,'2 years':2, '3 years':3,
     '8 years':8, '7 years':7, '5 years':5, '1 year':1}
df['emp_length']=df['emp_length'].replace(d)
```

In [13]:

```
#Convert columns with less number of unique values to categorical columns
cat_cols = ['term', 'grade','sub_grade','home_ownership',
            'verification_status','loan_status','purpose',
            'initial_list_status','application_type']

df[cat_cols] = df[cat_cols].astype('category')
```

In [14]:

```
# Checking for missing values
df.isnull().sum()
```

Out[14]:
```
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                    1756
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
```

We have a bunch of missing values

In [15]:
```
#Filling missing values with 'Unknown'
fill_values = {'title': 'Unknown', 'emp_title': 'Unknown'}
df.fillna(value=fill_values, inplace=True)
```

In [16]:
```
#Mean aggregation of mort_acc by total_acc to fill missing values

avg_mort = df.groupby('total_acc')['mort_acc'].mean()

def fill_mort(total_acc, mort_acc):
  if np.isnan(mort_acc):
    return avg_mort[total_acc].round()
  else:
    return mort_acc
```

In [17]:
```
df['mort_acc'] = df.apply(lambda x: fill_mort(x['total_acc'],x['mort_acc']),
```

In [18]:
```
df.dropna(inplace=True)
```

In [19]:
```
df.isna().sum()
```

Out[19]:
```
loan_amnt               0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
title                   0
dti                     0
earliest_cr_line        0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
initial_list_status     0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
address                 0
dtype: int64
```

In [20]:
```
df.shape
```

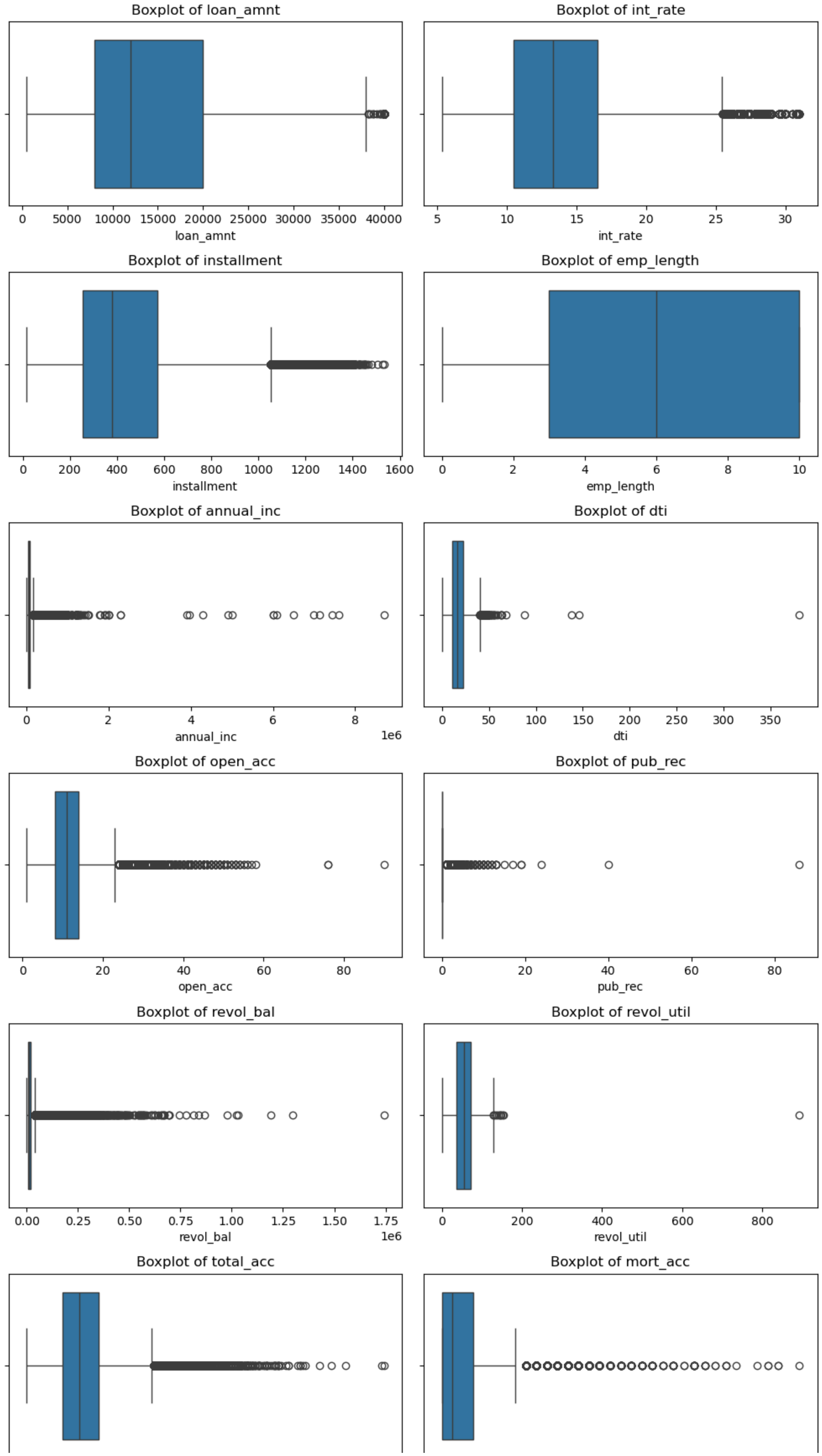
Out[20]:
```
(376929, 27)
```

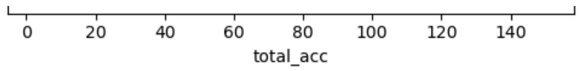
Outlier Treatment -

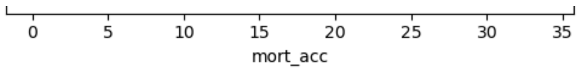
In [21]:
```
num_cols = df.select_dtypes(include='number').columns
num_cols
```

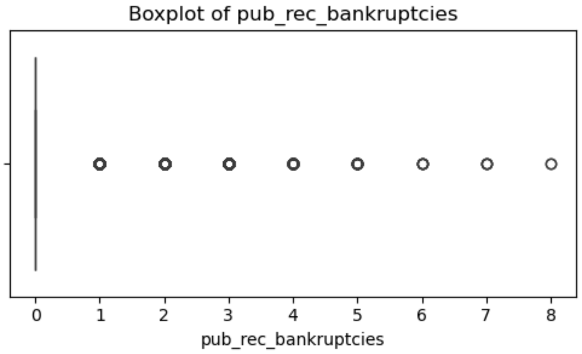
Out[21]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_ac
c',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

In [22]:
```
fig = plt.figure(figsize=(10,21))
i=1
for col in num_cols:
  ax = plt.subplot(7,2,i)
  sns.boxplot(x=df[col])
  plt.title(f'Boxplot of {col}')
  i += 1

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


Boxplot of loan_amnt
0 5000 10000 15000 20000 25000 30000 35000 40000
loan_amnt
Boxplot of int_rate
5 10 15 20 25 30
int_rate
Boxplot of installment
0 200 400 600 800 1000 1200 1400 1600
installment
Boxplot of emp_length
0 2 4 6 8 10
emp_length
Boxplot of annual_inc
0 2 4 6 8
annual_inc
1e6
Boxplot of dti
0 50 100 150 200 250 300 350
dti
Boxplot of open_acc
0 20 40 60 80
open_acc
Boxplot of pub_rec
0 20 40 60 80
pub_rec
Boxplot of revol_bal
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
revol_bal
1e6
Boxplot of revol_util
0 200 400 600 800
revol_util
Boxplot of total_acc
Boxplot of mort_acc







Here we can see that most of the columns have outliers

In [23]:
```
# Converting pub_rec and pub_rec_bankruptcies to categorical variables

df['pub_rec_bankruptcies'] = np.where(df['pub_rec_bankruptcies']>0,'yes','no
df['pub_rec'] = np.where(df['pub_rec']>0,'yes','no')
df[['pub_rec_bankruptcies','pub_rec']] = df[['pub_rec_bankruptcies','pub_rec
```

In [24]:
```
# Numeric columns after converting public records to categorical variables
num_cols = df.select_dtypes(include='number').columns
num_cols
```

Out[24]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_ac
c'],
      dtype='object')
```

In [25]:
```
#Removing outliers using SD
for col in num_cols:
  mean=df[col].mean()
  std=df[col].std()
  upper = mean + (3*std)
  df = df[~(df[col]>upper)]
```

In [26]:
```
df.shape
```

Out[26]:
```
(350845, 27)
```

In [27]:
```
df['address'].sample(10)
```

Out[27]:
```
156215                             USCGC Gallegos\r\nFPO AE 30723
1351          07151 Jackson Light\r\nSouth Danielle, HI 70466
100755    80671 Gonzalez Station Suite 455\r\nSheenaches...
185923    4815 Osborne Crossroad Suite 501\r\nWest Rose,...
331303        64415 Jeffrey Walks\r\nRamirezhaven, PA 00813
126409    659 Stephen Stream Apt. 800\r\nNew Jennifer, A...
372875    3924 Brenda Lock Apt. 613\r\nNorth Stephanieha...
321434         78716 Bryan Drive\r\nHendersonbury, MI 00813
360433             4670 John Forges\r\nAngelastad, ND 00813
201379    49362 Romero Ville Apt. 011\r\nDerrickfort, SD...
Name: address, dtype: object
```

In [28]:
```
# Deriving zip code and state from address
df[['state', 'zip_code']] = df['address'].apply(lambda x: pd.Series([x[-8:-
```

In [29]:
```
#Drop address
df.drop(["address"], axis = 1, inplace=True)
```

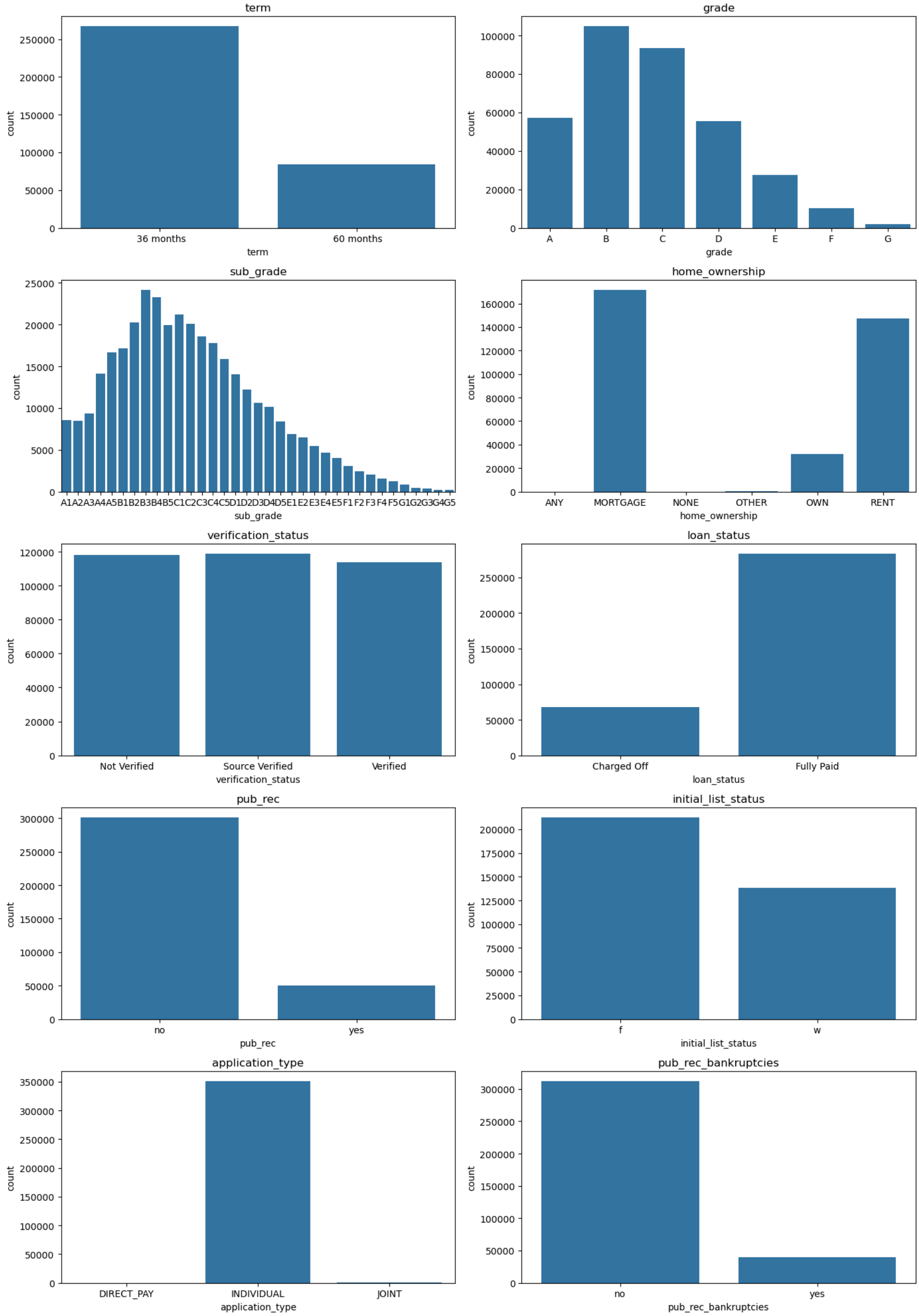
In [30]:
```
df.zip_code.nunique()
```

Out[30]:
```
10
```

In [31]:
```
# There are only 10 zipcodes. so we can convert the datatype of zipcode to c
df['zip_code'] = df['zip_code'].astype('category')
```

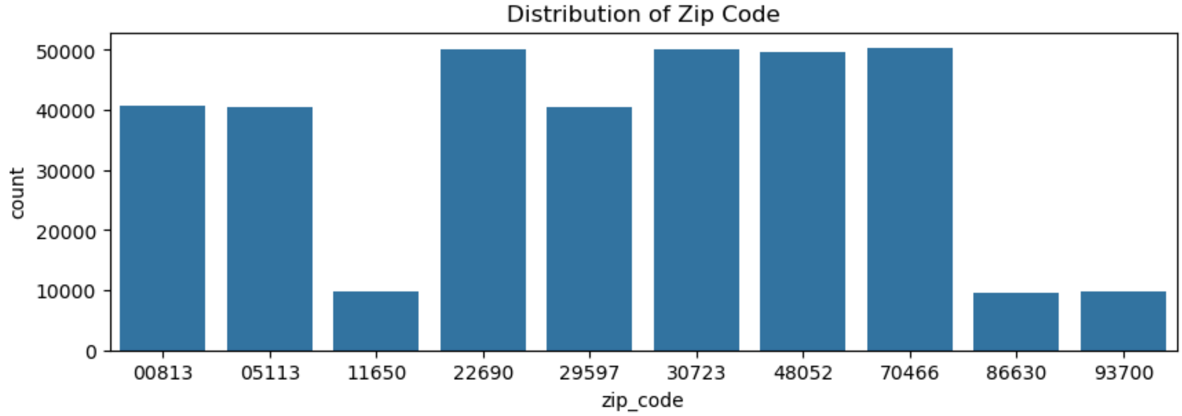
Visual Analysis

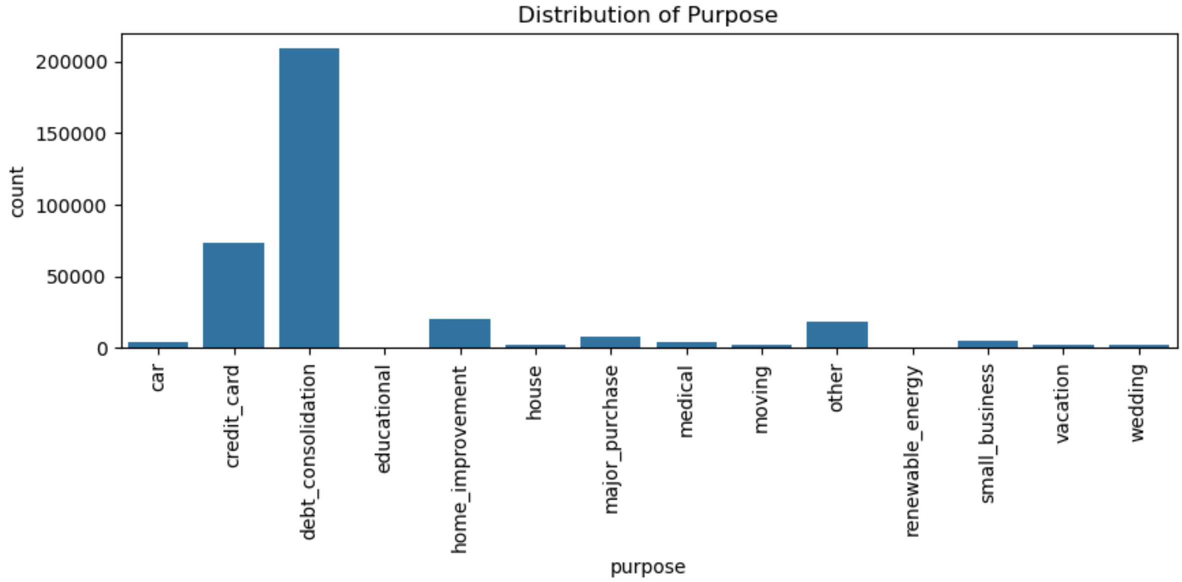
In [32]:
```
#Distribution of categorical variables
plot = ['term', 'grade', 'sub_grade', 'home_ownership', 'verification_statu
       'loan_status', 'pub_rec', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies']

plt.figure(figsize=(14,20))
i=1
for col in plot:
  ax=plt.subplot(5,2,i)
  sns.countplot(x=df[col])
  plt.title(f'{col}')
  i += 1

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



In [33]:

```
plt.figure(figsize=(10,3))
sns.countplot(x=df['zip_code'])
plt.title('Distribution of Zip Code')

plt.figure(figsize=(10,3))
sns.countplot(x=df['purpose'])
plt.xticks(rotation=90)
plt.title('Distribution of Purpose')

plt.show()
```





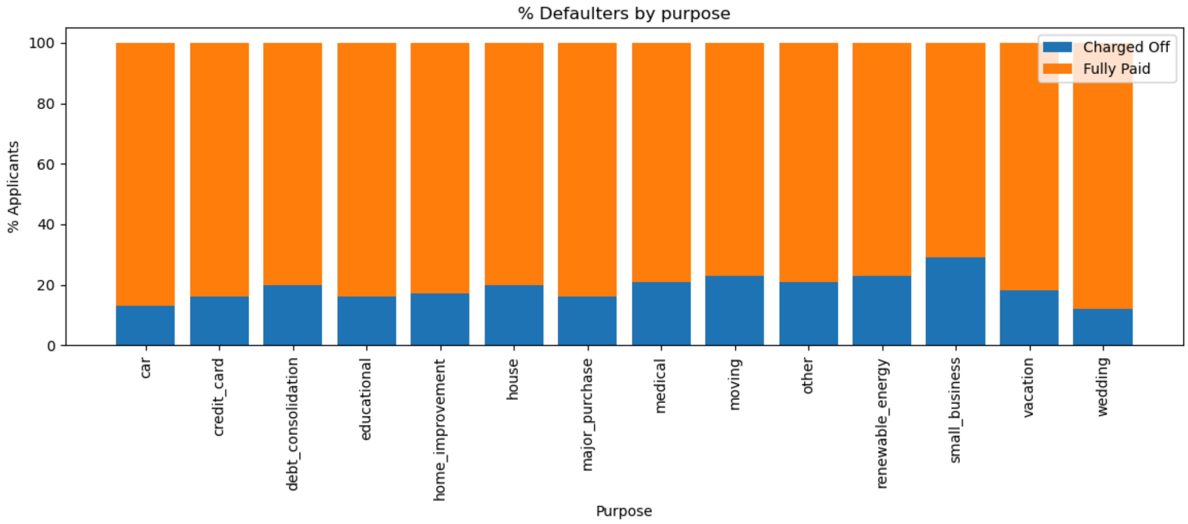
Observations -

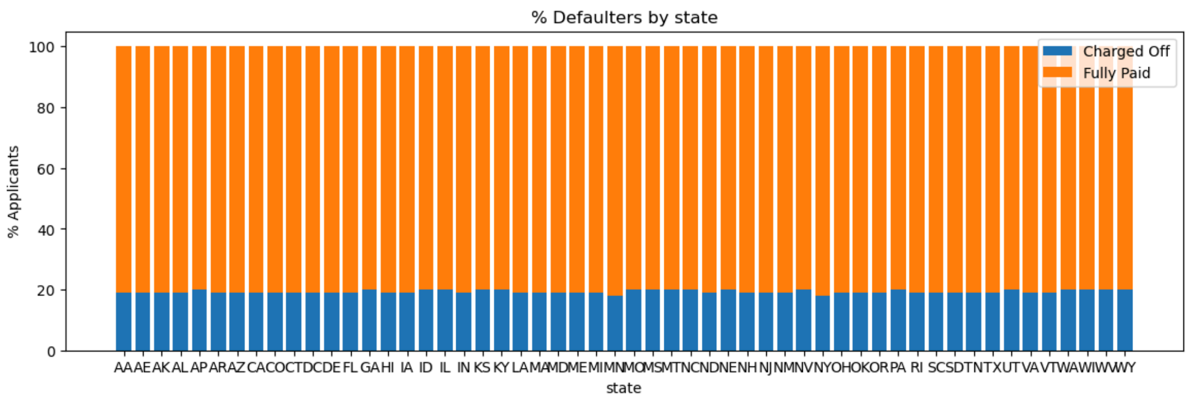
- Most of the loans have a time period of 36 months
- Majority of loans comes under Grade B
- Under Grade B, B3 is the most common sub-grade
- Most common form of home ownership is Mortgage and Rent
- Most of the applicants don't have a derogatory public record
- Majority of applicants have applied under 'individual' category
- Majority of customers don't have a public record of bankruptcies
- Debt consolidation is the main purpose for which customers are taking loans

In [34]:
```
# Impact of categorical factors on loan status

plot = ['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status
       'zip_code', 'pub_rec', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies']

plt.figure(figsize=(14,20))
i=1
for col in plot:
  ax=plt.subplot(5,2,i)

  data = df.pivot_table(index=col, columns='loan_status', aggfunc='count', v
  data = data.div(data.sum(axis=1), axis=0).multiply(100).round()
  data.reset_index(inplace=True)

  plt.bar(data[col],data['Charged Off'])
```

```
    plt.bar(data[col],data['Fully Paid'], bottom=data['Charged Off'])
    plt.xlabel(f'{col}')
    plt.ylabel('% Applicants')
    plt.title(f'% Defaulters by {col}')
    plt.legend(['Charged Off','Fully Paid'])
    i += 1

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

In [35]:
```
# Impact of Purpose/state on loan status

purpose = df.pivot_table(index='purpose', columns='loan_status', aggfunc='co
purpose = purpose.div(purpose.sum(axis=1), axis=0).multiply(100).round()
purpose.reset_index(inplace=True)

plt.figure(figsize=(14,4))
plt.bar(purpose['purpose'],purpose['Charged Off'])
plt.bar(purpose['purpose'],purpose['Fully Paid'], bottom=purpose['Charged O
plt.xlabel('Purpose')
plt.ylabel('% Applicants')
plt.title('% Defaulters by purpose')
plt.legend(['Charged Off','Fully Paid'])
plt.xticks(rotation=90)
plt.show()

state = df.pivot_table(index='state', columns='loan_status', aggfunc='count
state = state.div(state.sum(axis=1), axis=0).multiply(100).round()
state.reset_index(inplace=True)

plt.figure(figsize=(14,4))
plt.bar(state['state'],state['Charged Off'])
plt.bar(state['state'],state['Fully Paid'], bottom=state['Charged Off'])
plt.xlabel('state')
plt.ylabel('% Applicants')
plt.title('% Defaulters by state')
plt.legend(['Charged Off','Fully Paid'])
plt.show()
```



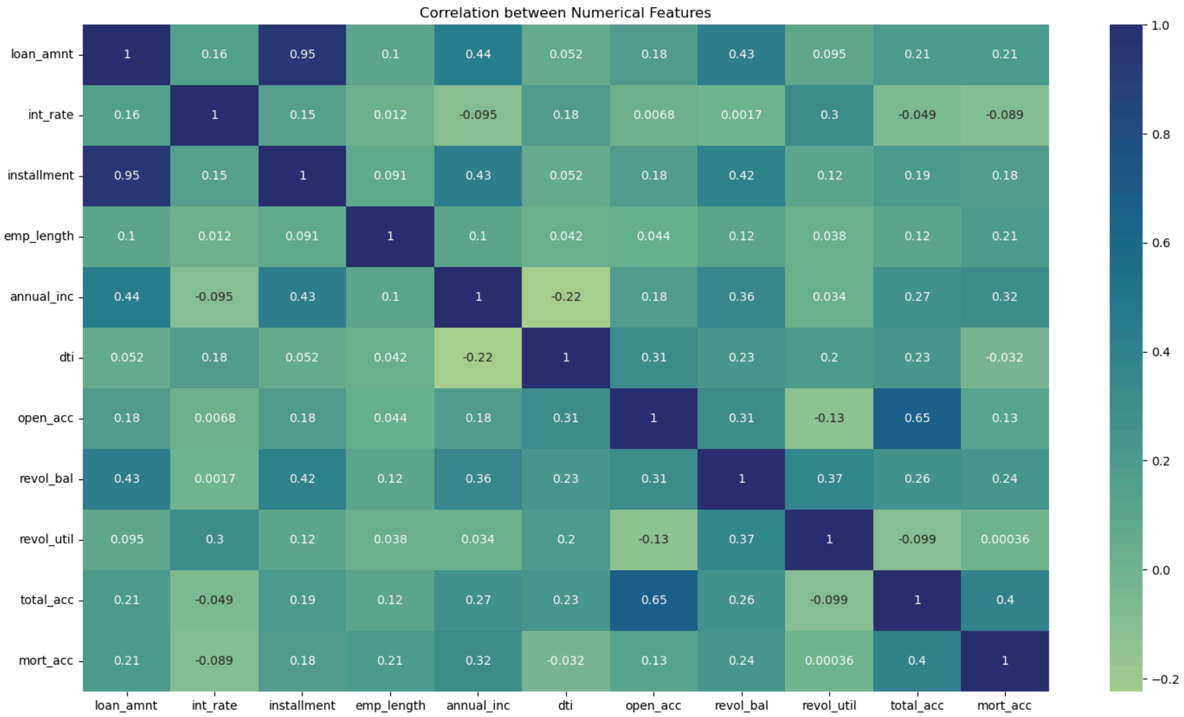


Observations -

- Nummber of defaulters is higher for (60-month) term loans than (36 months) term loan

- Most of the Grade A customers fully paid of their loans
- Whereas Grade G customers is comparitively bad at fully paying back their loans
- Zip codes like 11650, 86630 and 93700 have 100% defaulters
- We can remove initial_list_status and state as they have no impact on loan_status public records also don't seem to have any impact on loan_status surprisingly
- Direct pay application type has higher default rate compared to individual/joint application type
- Among the loans, Small business loans have a higher rate of default

In [36]:
```
plt.figure(figsize=(18,10))
sns.heatmap(df.corr(numeric_only=True), cmap = 'crest', annot = True)
plt.title('Correlation between Numerical Features')
plt.show()
```



Observations -

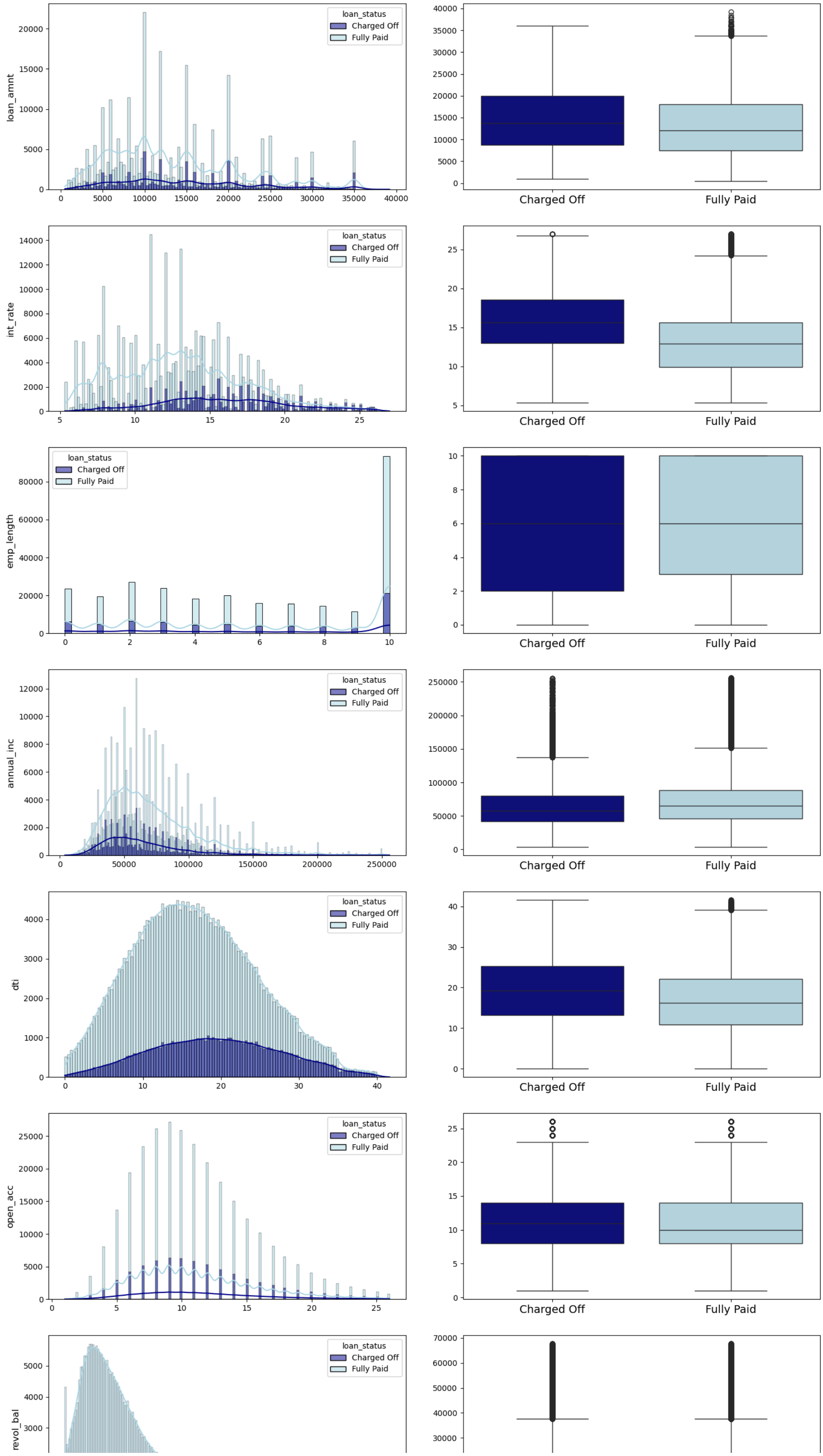
- Loan Amount have a high correlation with Installment with a correlation factor of 0.95
- There is also good correlation between open_acc and total_acc

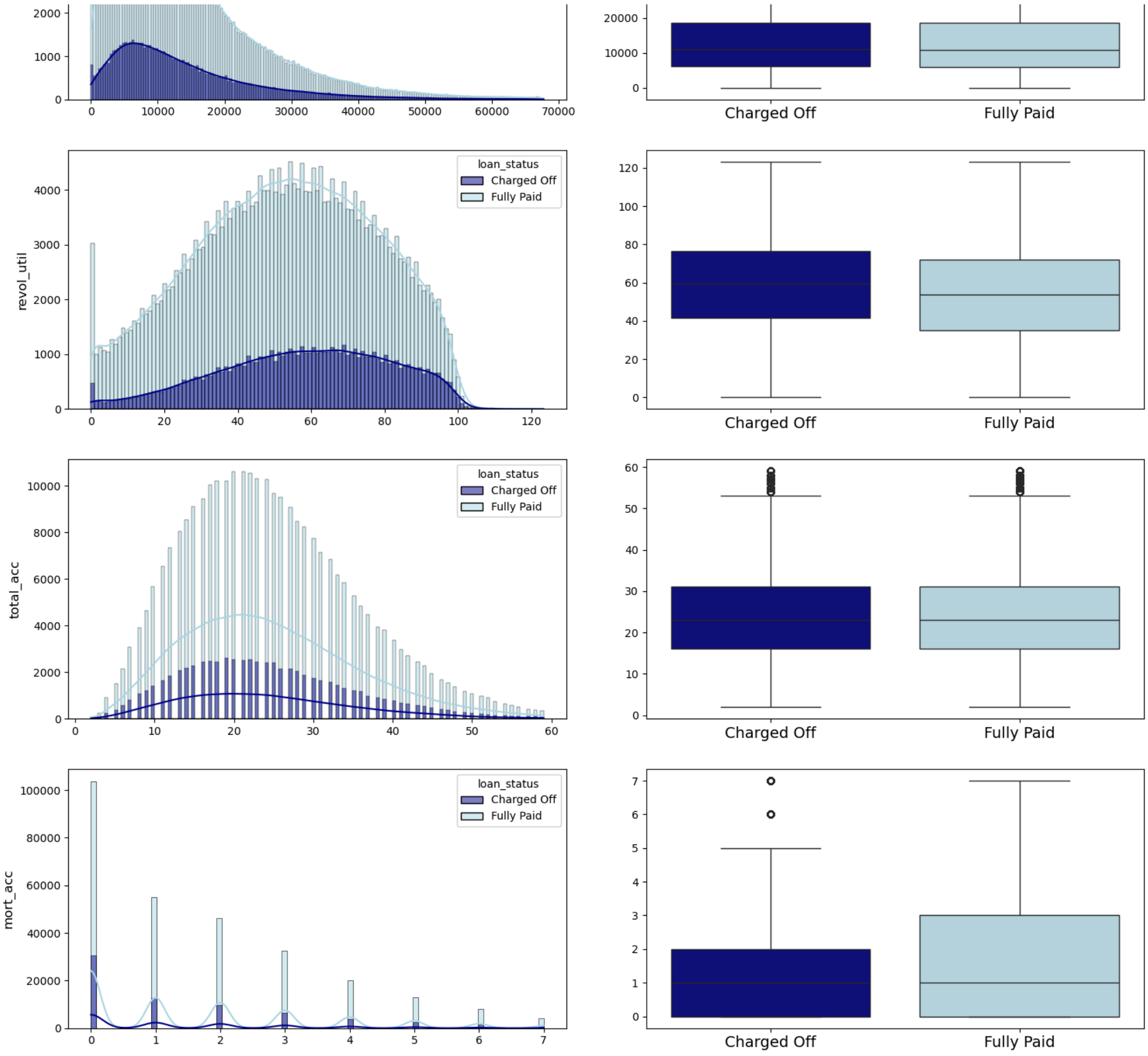
To avoid Multicollinaerity we can remove some of these correlated features

In [37]:
```
#Drop installment
df.drop(columns=['installment'], inplace=True)
```

In [38]:
```
# Impact of numerical features on loan_status

num_cols = df.select_dtypes(include='number').columns

fig, ax = plt.subplots(10,2,figsize=(15,40))
i=0
color_dict = {'Fully Paid': matplotlib.colors.to_rgba('#add8e6', 0.5),
              'Charged Off': matplotlib.colors.to_rgba('#00008b', 1)}
for col in num_cols:
    sns.histplot(data=df, x=col, hue='loan_status', ax=ax[i, 0], legend=Tru
```

```
                palette=color_dict, kde=True, fill=True)
    sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
               palette=('#00008b', '#add8e6'))
    ax[i,0].set_ylabel(col, fontsize=12)
    ax[i,0].set_xlabel(' ')
    ax[i,1].set_xlabel(' ')
    ax[i,1].set_ylabel(' ')
    ax[i,1].xaxis.set_tick_params(labelsize=14)
    i += 1

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


loan_amnt
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
int_rate
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
emp_length
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
annual_inc
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
dti
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
open_acc
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
revol_bal
loan_status
Charged Off
Fully Paid



Observations -

- It can be seen that the mean Loan Amount, Interest Rate, dti, open_acc and revol_util are slightly higher for defaulters
- Whereas Annual Income is lower for defaulters

In [39]:
```
# Remove columns which do not have an impact on loan_status
df.drop(columns=['initial_list_status','state',
                 'emp_title', 'title','earliest_cr_line',
                 'issue_d','sub_grade'], inplace=True)

# Subgrade is removed because grade and subgrade are similar features
```

In [40]:
```
# Encoding Target Variable

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

In [41]:
```
x = df.drop(columns=['loan_status'])
x.reset_index(inplace=True, drop=True)
y = df['loan_status']
y.reset_index(drop=True, inplace=True)
```

In [42]:
```
# Encoding Binary features into numerical dtype

x['term']=x['term'].map({' 36 months': 36, ' 60 months':60}).astype(int)
```

```
x['pub_rec']=x['pub_rec'].map({'no': 0, 'yes':1}).astype(int)
x['pub_rec_bankruptcies']=x['pub_rec_bankruptcies'].map({'no': 0, 'yes':1})
```

One Hot Encoding of Categorical Features

In [43]:
```
cat_cols = x.select_dtypes('category').columns

encoder = OneHotEncoder(sparse=False)
encoded_data = encoder.fit_transform(x[cat_cols])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_ou
x = pd.concat([x,encoded_df], axis=1)
x.drop(columns=cat_cols, inplace=True)
x.head()
```

Out[43]:

	loan_amnt	term	int_rate	emp_length	annual_inc	dti	open_acc	pub_rec	revol_bal
0	10000.0	36	11.44	10.0	117000.0	26.24	16.0	0	36369.0
1	8000.0	36	11.99	4.0	65000.0	22.05	17.0	0	20131.0
2	15600.0	36	10.49	0.0	43057.0	12.79	13.0	0	11987.0
3	7200.0	36	6.49	6.0	54000.0	2.60	6.0	0	5472.0
4	24375.0	60	17.27	9.0	55000.0	33.95	13.0	0	24584.0

Train-Test Split

In [44]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20,strat
```

In [45]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[45]:
```
((280676, 56), (280676,), (70169, 56), (70169,))
```

Scaling Numeric Features

In [46]:
```
scaler = MinMaxScaler()
x_train = pd.DataFrame(scaler.fit_transform(x_train), columns=x_train.columr
x_test = pd.DataFrame(scaler.transform(x_test), columns=x_test.columns)
```

In [47]:
```
x_train.tail()
```

Out[47]:

	loan_amnt	term	int_rate	emp_length	annual_inc	dti	open_acc	pub_rec
280671	0.167959	0.0	0.141671	0.7	0.194444	0.255954	0.60	0.0
280672	0.497416	0.0	0.445778	0.4	0.182540	0.414482	0.24	0.0
280673	0.064599	0.0	0.686664	0.7	0.238095	0.220111	0.32	0.0
280674	0.245478	1.0	0.177665	0.9	0.313492	0.134953	0.92	0.0
280675	0.646641	1.0	0.885095	0.6	0.349206	0.747173	0.88	1.0

Oversampling using SMOTE

In [48]:
```
# Oversampling to balance the target variable

sm=SMOTE(random_state=42)
x_train_res, y_train_res = sm.fit_resample(x_train,y_train.ravel())
```

```
print(f"Before OverSampling, count of label 1: {sum(y_train == 1)}")
print(f"Before OverSampling, count of label 0: {sum(y_train == 0)}")
print(f"After OverSampling, count of label 1: {sum(y_train_res == 1)}")
print(f"After OverSampling, count of label 0: {sum(y_train_res == 0)}")
```

```
Before OverSampling, count of label 1: 54200
Before OverSampling, count of label 0: 226476
After OverSampling, count of label 1: 226476
After OverSampling, count of label 0: 226476
```

Logistic Regression

In [49]:
```
model = LogisticRegression()
model.fit(x_train_res, y_train_res)
train_preds = model.predict(x_train)
test_preds = model.predict(x_test)
```

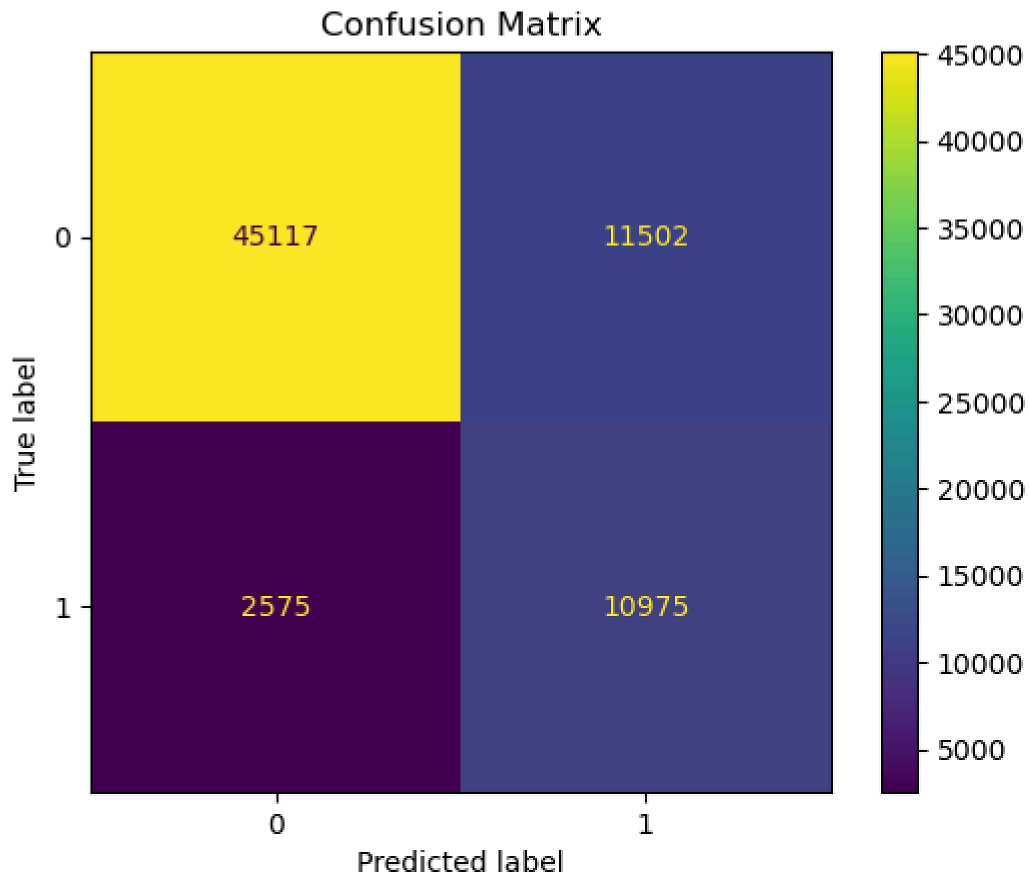
Model Evaluation

In [50]:
```
print('Train Accuracy :', model.score(x_train, y_train).round(2))
print('Train F1 Score:',f1_score(y_train,train_preds).round(2))
print('Train Recall Score:',recall_score(y_train,train_preds).round(2))
print('Train Precision Score:',precision_score(y_train,train_preds).round(2

print('\nTest Accuracy :',model.score(x_test,y_test).round(2))
print('Test F1 Score:',f1_score(y_test,test_preds).round(2))
print('Test Recall Score:',recall_score(y_test,test_preds).round(2))
print('Test Precision Score:',precision_score(y_test,test_preds).round(2))
```

```
Train Accuracy : 0.8
Train F1 Score: 0.61
Train Recall Score: 0.81
Train Precision Score: 0.49

Test Accuracy : 0.8
Test F1 Score: 0.61
Test Recall Score: 0.81
Test Precision Score: 0.49
```

Confusion Matrix

In [51]:
```
cm = confusion_matrix(y_test, test_preds)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```



Classification Report

In [52]:
```
print(classification_report(y_test, test_preds))
```

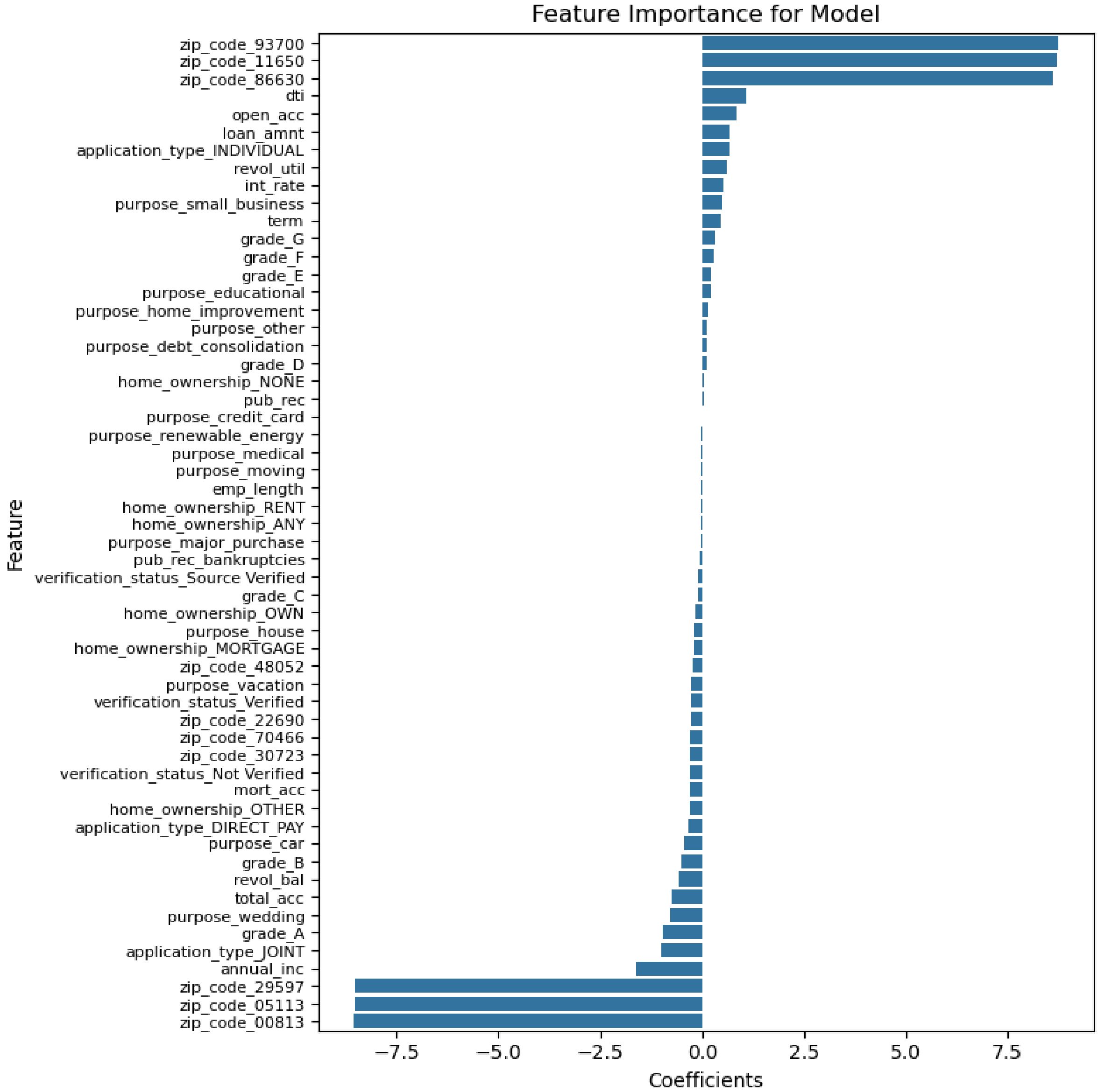
```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87     56619
           1       0.49      0.81      0.61     13550

    accuracy                           0.80     70169
   macro avg       0.72      0.80      0.74     70169
weighted avg       0.86      0.80      0.82     70169
```

- From the report we can see that the recall score is very high (our model is able to identify 80% of actual defaulters) but the precision is low for positive class (of all the predicted defaulters, only 50% are actually defaulters).
- The model is effective in reducing NPAs by flagging most of the defaulters, but it may cause loantap to deny loans to many deserving customers due to low precision (false positives)
- Low precision has also caused F1 score to drop to 60% even though accuracy is 80%

Feature Importance

In [53]:
```
feature_imp = pd.DataFrame({'Columns':x_train.columns, 'Coefficients':model

plt.figure(figsize=(8,8))
sns.barplot(y = feature_imp['Columns'],
            x =  feature_imp['Coefficients'])
```

```
plt.title("Feature Importance for Model")
plt.yticks(fontsize=8)
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

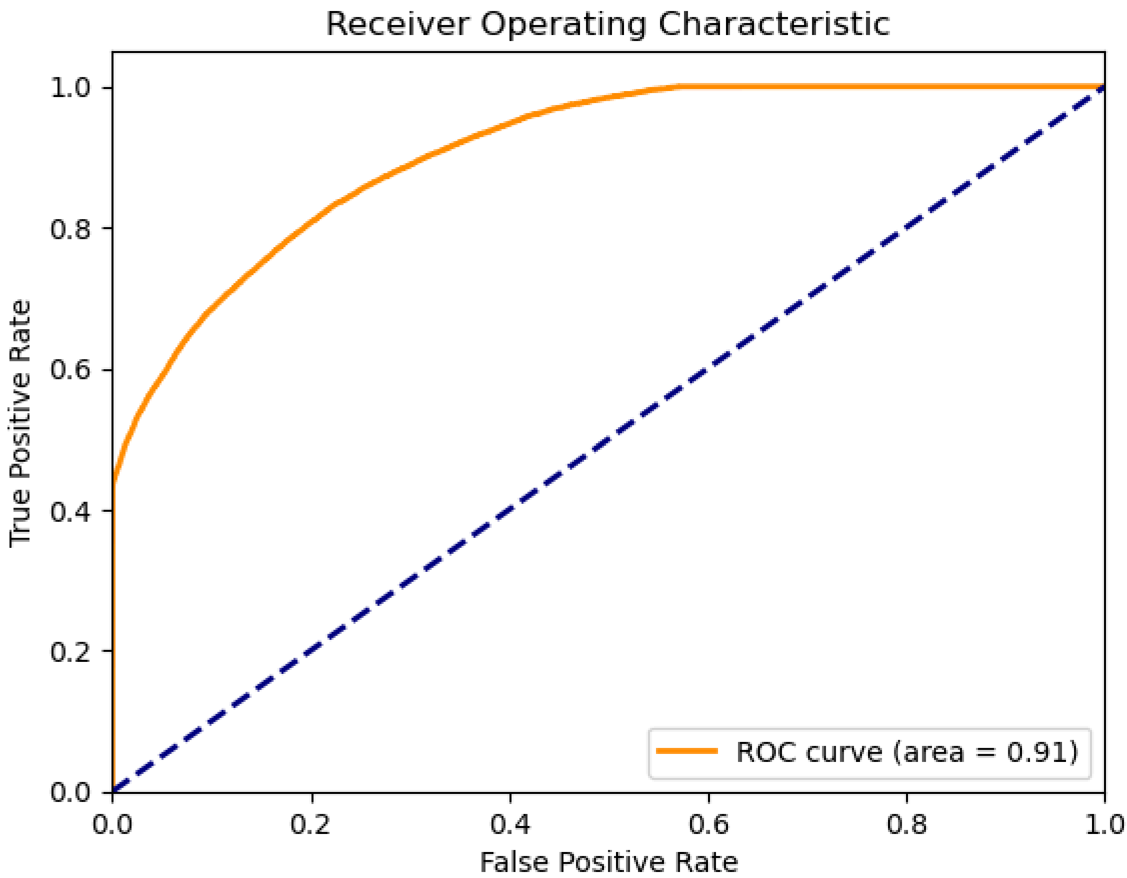


- It can be observed that the model is giving high importance to features like zip code, dti, open_acc, loan_amount, application_type_individual
- Also certain zipcodes, annual_inc, application_type_JOINT have negative coefficients

ROC/AUC

In [54]:
```
# Predicting probabilities for the test set
probs = model.predict_proba(x_test)[:,1]

# Computing the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Computing the area under the ROC curve
roc_auc = auc(fpr, tpr)
```

```
# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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()
```



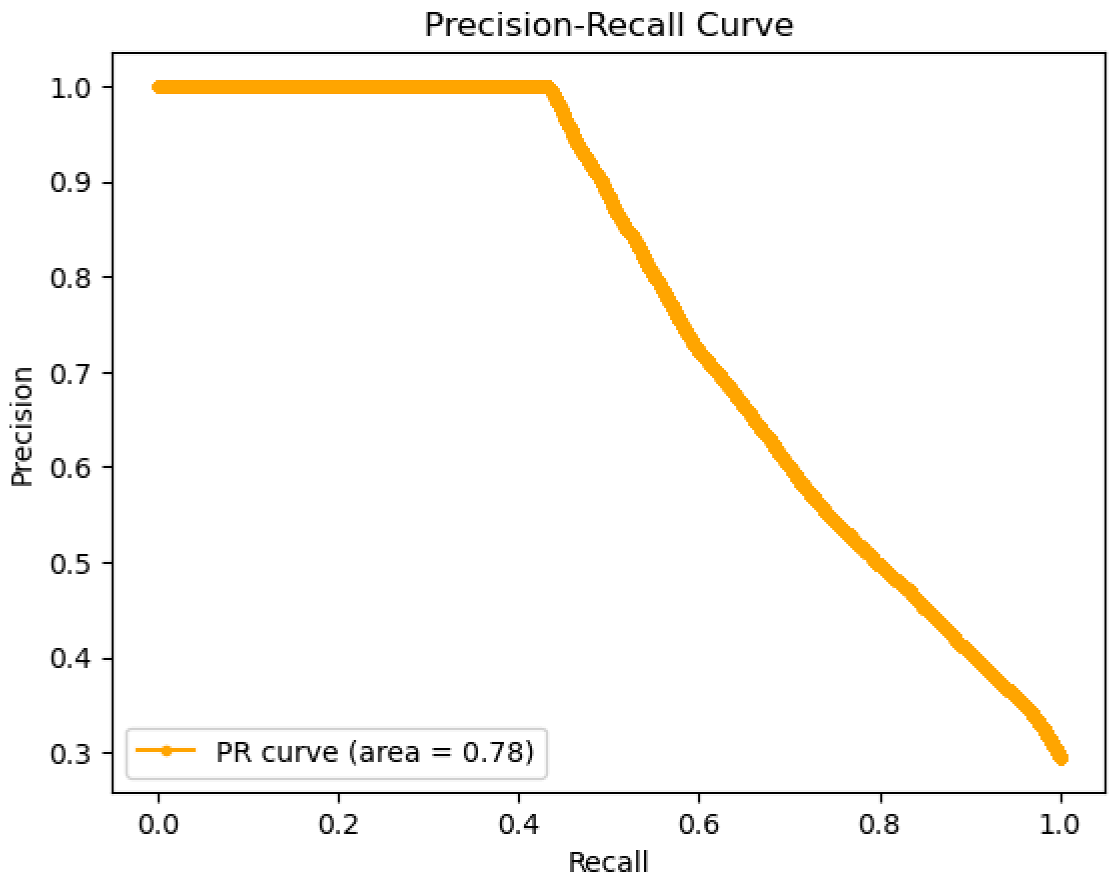
Observations -

- AUC of 0.91 shows that the model is able to discriminate well between the positive and the negative class.
- But it is not a good measure for an imbalanced target variable as it may be high even when the classifier has a poor score on the minority class.
- By collecting more data,or using a more complex model, or tuning the hyperparameters, it is possible to improve the model's performance.

Precision Recall Curve

In [55]:
```
# Compute the false precision and recall at all thresholds
precision, recall, thresholds = precision_recall_curve(y_test, probs)

# Area under Precision Recall Curve
auprc = average_precision_score(y_test, probs)

# Plot the precision-recall curve
plt.plot(recall, precision, marker='.', label='PR curve (area = %0.2f)' % au
plt.xlabel('Recall')
```

```
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```



Observations -

- Precision score is highest at 0.55 threshold. High precision value indicates that model is positevly predicting the charged off loan status which helps business to take more stable decision.
- It is a decent model as the area is more than 0.5 (random model benchmark) but there is still room for improvement

Conclusion

- 80% of the customers have paid their loan fully. 20% of the customers are defaulters.
- The Percentage of defaulters are much higher for the longer term ie. 60 months
- Grade/Sub-grade have the maximum impact on loan_status as it clearly shows that the highest grade have maximum defaulters
- Direct pay application type defaults on loans more compared to individual/joint application types
- Similarly small businesses default on their loans compared to others
- Features like mean loan_amnt, int_rate, dti, open_acc and revol_util are higher for defaulters

- Whereas mean annual_income is low for defaulters
- The logistic Regression model performed well with an accuracy of 80%
- The Area under ROC curve is 0.91, which signifies that the model is able to differentiate well between both classes
- The Area under Precision Recall curve is 0.78 but there is still room for improvement
- By collecting more data, using a more complex model or tuning the hyperparameters, it is possible to improve the model's performance.

Recommendations

- We can reduce the number of loans given for long term(60 months) as they have higher chance of being written off. Instead increase the number of loans with short term periods
- We can try to improve the F1 score and the area under Precision Recall Curve
- We can advertise more to customers of Grade A, B, C as they have a high probability of paying back the loan. We can give them attractive interest rates and flexible time periods.

In [ ]: