laontap-logistic-regression

February 8, 2024

0.0.1 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.

0.0.2 The data science team at LoanTap is building an underwriting layer to determine the creditworthiness of MSMEs as well as individuals.

Problem Statement: Given a set of attributes for an Individual, determine if a credit line should be extended to them. The main challenge is to minimise the risk of NPAs by flagging defaulters while maximising opportunity to earn interest by disbursing loans to as many customers as possible. Data dictionary:

1. loan_amnt : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
2. term : The number of payments on the loan. Values are in months and can be either 36 or 60.
3. installment : The monthly payment owed by the borrower if the loan originates.
4. sub_grade : LoanTap assigned loan subgrade
5. emp_title :The job title supplied by the Borrower when applying for the loan.
6. 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.
7. home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.
8. annual_inc : The self-reported annual income provided by the borrower during registration.
9. verification_status : Indicates if income was verified by LoanTap, not verified, or if the income source was verified
10. issue_d : The month which the loan was funded
11. loan_status : Current status of the loan - Target Variable
12. purpose : A category provided by the borrower for the loan request.
13. title : The loan title provided by the borrower

14. dti : A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower's self-reported monthly income.
15. earliest_cr_line :The month the borrower's earliest reported credit line was opened
16. open_acc : The number of open credit lines in the borrower's credit file.
17. pub_rec : Number of derogatory public records
18. revol_bal : Total credit revolving balance
19. revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
20. total_acc : The total number of credit lines currently in the borrower's credit file
21. initial_list_status : The initial listing status of the loan. Possible values are – W, F
22. application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers
23. pub_rec_bankruptcies : Number of public record bankruptcies
24. Address: Address of the individual
25. int_rate : Interest Rate on the loan
26. grade : LoanTap assigned loan grade
27. mort_acc : Number of mortgage accounts.

1 Importing Libraries

[19]:
```
#Data processing
import pandas as pd
import numpy as np

#Data Visualisation
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline

#Seting option for full column view of Data
pd.set_option('display.max_columns', None)

#Stats & model building
from scipy import stats
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```python
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")
```

[20]:
```python
df = pd.read_csv('logistic_regression (1).csv')
df.head()
```

[20]:
```
   loan_amnt        term  int_rate  installment grade sub_grade  \
0    10000.0   36 months     11.44       329.48     B        B4
1     8000.0   36 months     11.99       265.68     B        B5
2    15600.0   36 months     10.49       506.97     B        B3
3     7200.0   36 months      6.49       220.65     A        A2
4    24375.0   60 months     17.27       609.33     C        C5

                 emp_title emp_length home_ownership  annual_inc  \
0                Marketing  10+ years           RENT    117000.0
1          Credit analyst     4 years       MORTGAGE     65000.0
2             Statistician   < 1 year           RENT     43057.0
3          Client Advocate    6 years           RENT     54000.0
4  Destiny Management Inc.    9 years       MORTGAGE     55000.0

  verification_status   issue_d  loan_status             purpose  \
0        Not Verified  Jan-2015   Fully Paid            vacation
1        Not Verified  Jan-2015   Fully Paid  debt_consolidation
2     Source Verified  Jan-2015   Fully Paid         credit_card
3        Not Verified  Nov-2014   Fully Paid         credit_card
4            Verified  Apr-2013  Charged Off         credit_card

                     title    dti earliest_cr_line  open_acc  pub_rec  \
0                 Vacation  26.24         Jun-1990      16.0      0.0
1       Debt consolidation  22.05         Jul-2004      17.0      0.0
2  Credit card refinancing  12.79         Aug-2007      13.0      0.0
3  Credit card refinancing   2.60         Sep-2006       6.0      0.0
4    Credit Card Refinance  33.95         Mar-1999      13.0      0.0

   revol_bal  revol_util  total_acc initial_list_status application_type  \
0    36369.0        41.8       25.0                   w       INDIVIDUAL
1    20131.0        53.3       27.0                   f       INDIVIDUAL
```

```
2    11987.0         92.2       26.0                    f       INDIVIDUAL
3     5472.0         21.5       13.0                    f       INDIVIDUAL
4    24584.0         69.8       43.0                    f       INDIVIDUAL

   mort_acc  pub_rec_bankruptcies  \
0       0.0                   0.0
1       3.0                   0.0
2       0.0                   0.0
3       0.0                   0.0
4       1.0                   0.0

                                              address
0     0174 Michelle Gateway\r\nMendozaberg, OK 22690
1  1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113
2  87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113
3            823 Reid Ford\r\nDelacruzside, MA 00813
4             679 Luna Roads\r\nGreggshire, VA 11650
```

[21]:
```python
#shape of data

df.shape
```

[21]:
```
(396030, 27)
```

[22]:
```python
# Statistical summary

df.describe()
```

[22]:
```
            loan_amnt       int_rate    installment    annual_inc  \
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean    14113.888089      13.639400     431.849698  7.420318e+04
std      8357.441341       4.472157     250.727790  6.163762e+04
min       500.000000       5.320000      16.080000  0.000000e+00
25%      8000.000000      10.490000     250.330000  4.500000e+04
50%     12000.000000      13.330000     375.430000  6.400000e+04
75%     20000.000000      16.490000     567.300000  9.000000e+04
max     40000.000000      30.990000    1533.810000  8.706582e+06

                  dti       open_acc        pub_rec     revol_bal  \
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean       17.379514      11.311153       0.178191  1.584454e+04
std        18.019092       5.137649       0.530671  2.059184e+04
min         0.000000       0.000000       0.000000  0.000000e+00
25%        11.280000       8.000000       0.000000  6.025000e+03
50%        16.910000      10.000000       0.000000  1.118100e+04
75%        22.980000      14.000000       0.000000  1.962000e+04
max      9999.000000      90.000000      86.000000  1.743266e+06
```

```
          revol_util      total_acc       mort_acc  pub_rec_bankruptcies
count  395754.000000  396030.000000  358235.000000         395495.000000
mean       53.791749      25.414744       1.813991              0.121648
std        24.452193      11.886991       2.147930              0.356174
min         0.000000       2.000000       0.000000              0.000000
25%        35.800000      17.000000       0.000000              0.000000
50%        54.800000      24.000000       1.000000              0.000000
75%        72.900000      32.000000       3.000000              0.000000
max       892.300000     151.000000      34.000000              8.000000
```

2 Data Cleaning

[23]:
```python
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
```

Checking Column Datatypes

[24]:
```python
# Non-numeric columns
cat_cols = df.select_dtypes(include='object').columns
cat_cols
```

[24]:
```
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')
```

[25]:
```python
# 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: 48817
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
```

[26]:
```python
# 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'])
```

[27]:
```python
#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)
```

[28]:
```python
#Convert columns with less number of unique values to categorical columns
cat_cols = ['term', 'grade','sub_grade','home_ownership',
```

```python
            'verification_status','loan_status','purpose',
            'initial_list_status','application_type']

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

[29]:
```python
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  category
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  category
 5   sub_grade             396030 non-null  category
 6   emp_title             373103 non-null  object
 7   emp_length            377729 non-null  float64
 8   home_ownership        396030 non-null  category
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  category
 11  issue_d               396030 non-null  datetime64[ns]
 12  loan_status           396030 non-null  category
 13  purpose               396030 non-null  category
 14  title                 394275 non-null  object
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  datetime64[ns]
 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  category
 23  application_type      396030 non-null  category
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: category(9), datetime64[ns](2), float64(13), object(3)
memory usage: 57.8+ MB
```

Check for Duplicate Values

[30]:
```python
df.duplicated().sum()
```

[30]: 0

There are no duplicate instances in the data

Handling Missing Values

[31]:
```
df.isna().sum()
```

[31]:
```
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
```

[32]:
```
df.isna().sum()/len(df)*100
```

[32]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
```

```
issue_d                  0.000000
loan_status              0.000000
purpose                  0.000000
title                    0.443148
dti                      0.000000
earliest_cr_line         0.000000
open_acc                 0.000000
pub_rec                  0.000000
revol_bal                0.000000
revol_util               0.069692
total_acc                0.000000
initial_list_status      0.000000
application_type         0.000000
mort_acc                 9.543469
pub_rec_bankruptcies     0.135091
address                  0.000000
dtype: float64
```

[33]:
```
#Filling missing values with 'Unknown' for object dtype
fill_values = {'title': 'Unknown', 'emp_title': 'Unknown'}
df.fillna(value=fill_values, inplace=True)
```

[34]:
```
#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
```

[35]:
```
df['mort_acc'] = df.apply(lambda x: fill_mort(x['total_acc'],x['mort_acc']),
 ↪axis=1)
```

[36]:
```
df.dropna(inplace=True)
```

[37]:
```
df.isna().sum()
```

[37]:
```
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
```

[38]:
```python
df.shape
```

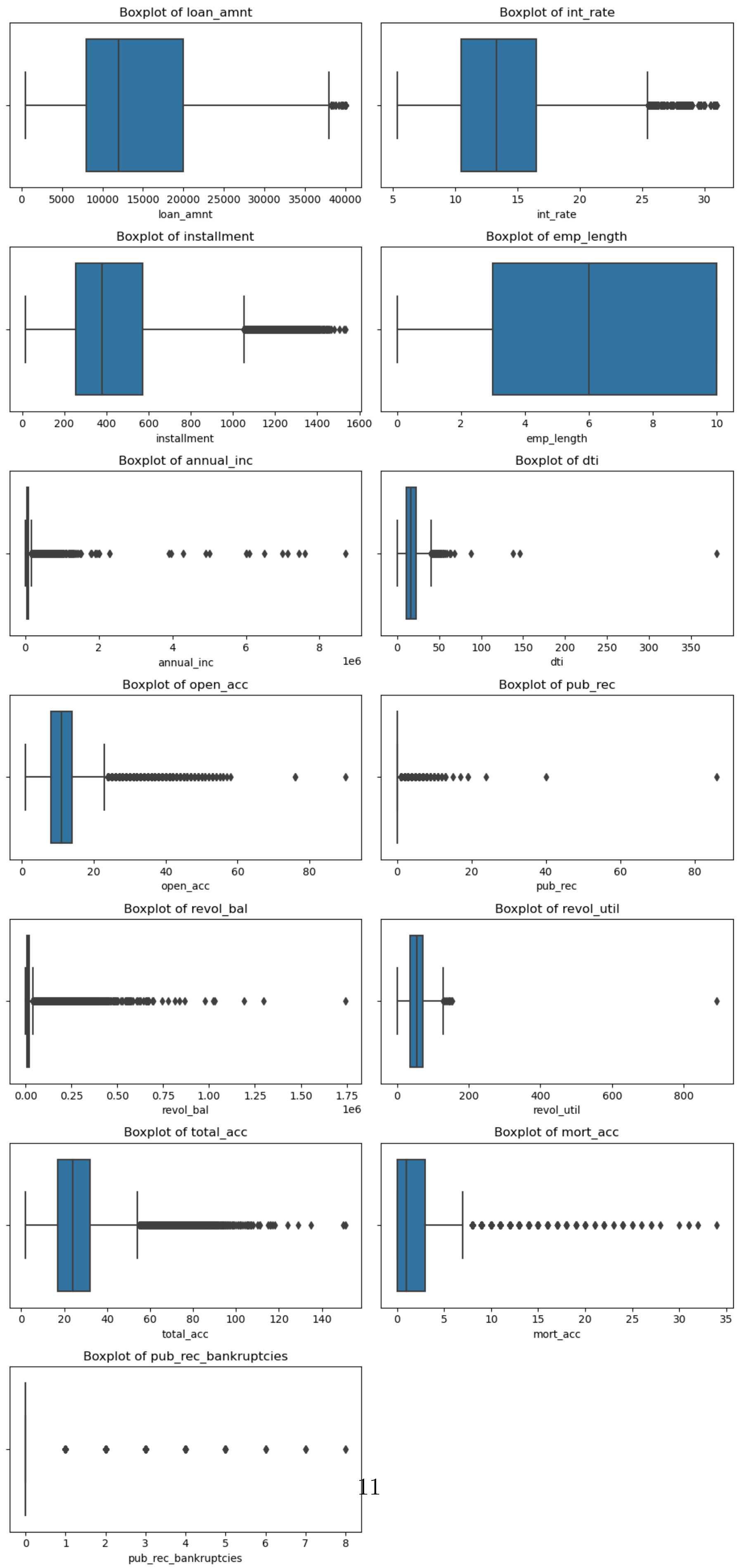
[38]:
```
(376929, 27)
```

Outlier Treatment

[39]:
```python
num_cols = df.select_dtypes(include='number').columns
num_cols
```

[39]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

[40]:
```python
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
0 20 40 60 80 100 120 140
total_acc
Boxplot of mort_acc
0 5 10 15 20 25 30 35
mort_acc
Boxplot of pub_rec_bankruptcies
0 1 2 3 4 5 6 7 8
pub_rec_bankruptcies

Here we can see that many columns have outliers. Lets remove the rows with outliers using standard deviation (99% data is within 3 standard deviations in case of normally distributed data).
For pub_Rec and pub_rec_bankruptcies, we can apply the 0 or 1 approach

[41]:
```
# Convert 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']].
↪astype('category')
```

[42]:
```
# Numeric columns after converting public records to category
num_cols = df.select_dtypes(include='number').columns
num_cols
```

[42]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

[43]:
```
#Removing outliers using standard deviation
for col in num_cols:
  mean=df[col].mean()
  std=df[col].std()
  upper = mean + (3*std)
  df = df[~(df[col]>upper)]
```

[44]:
```
df.shape
```

[44]:
```
(350845, 27)
```

Feature Engineering

[47]:
```
df['address'].sample(10)
```

[47]:
```
166010    95174 Edward Underpass Suite 112\r\nSouth Greg…
290941           2487 Carr Turnpike\r\nMedinaport, IA 48052
229517      82681 Hernandez Lodge\r\nPort Bradley, OK 30723
269296                   PSC 7648, Box 4669\r\nAPO AP 30723
189258    24638 Tiffany Ranch Apt. 554\r\nNew Seanport, …
38679                    PSC 2865, Box 8686\r\nAPO AP 30723
174761    55577 Sean Mills Suite 246\r\nLorimouth, MI 00813
176964                   Unit 4330 Box 8949\r\nDPO AA 29597
138981    4963 Swanson Mount Apt. 781\r\nRyanmouth, OH 2…
14622              0343 Daniels Track\r\nNew Megan, HI 11650
Name: address, dtype: object
```

[48]:
```
# Deriving zip code and state from address
df[['state', 'zip_code']] = df['address'].apply(lambda x: pd.Series([x[-8:-6],␣
 ↪x[-5:]]))
```

[49]:
```
#Drop address
df.drop(["address"], axis = 1, inplace=True)
```

[50]:
```
df.zip_code.nunique()
```

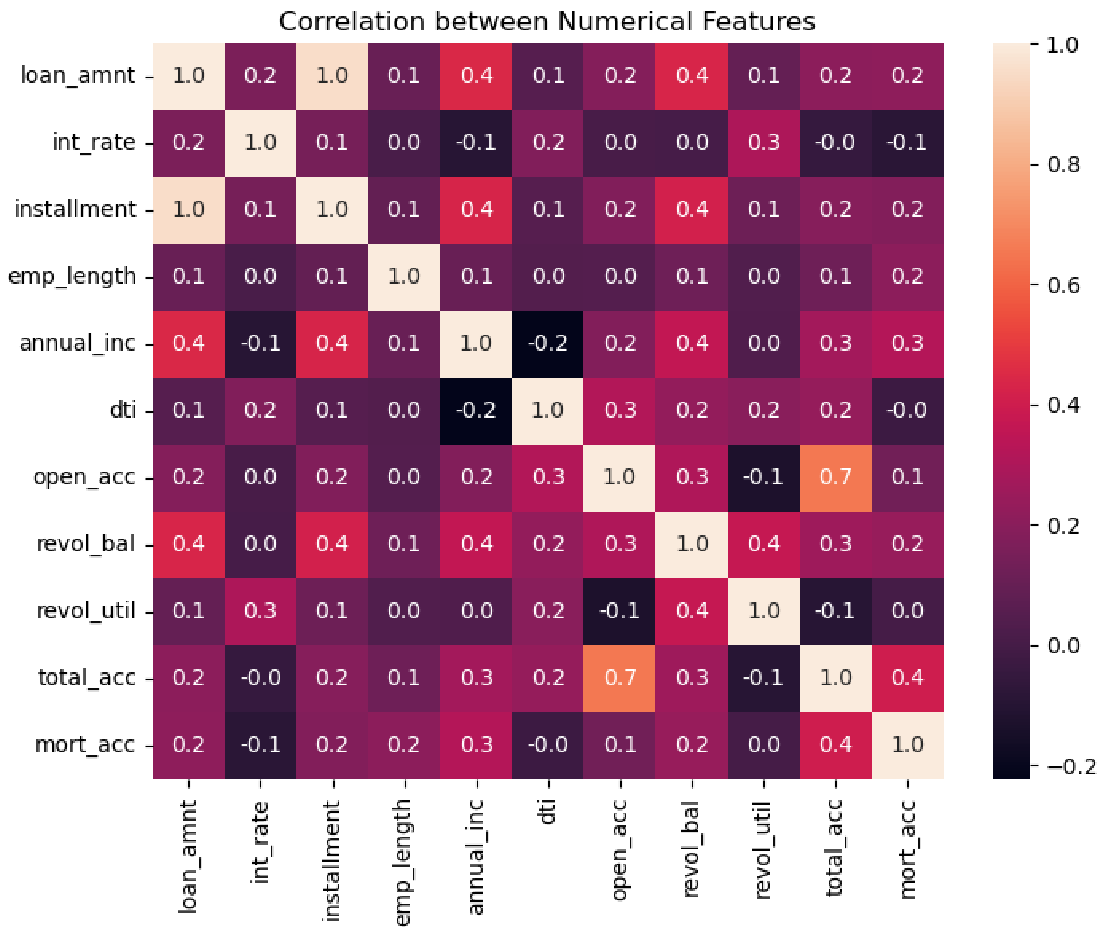
[50]: 10

Since there are only 10 zipcodes, we can change the datatype of zipcodes to categorical

[51]:
```
df['zip_code'] = df['zip_code'].astype('category')
```

3 Exploratory Data Analysis

[52]:
```
#Correlation between numerical features
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, fmt=".1f")
plt.title('Correlation between Numerical Features')
plt.show()
```



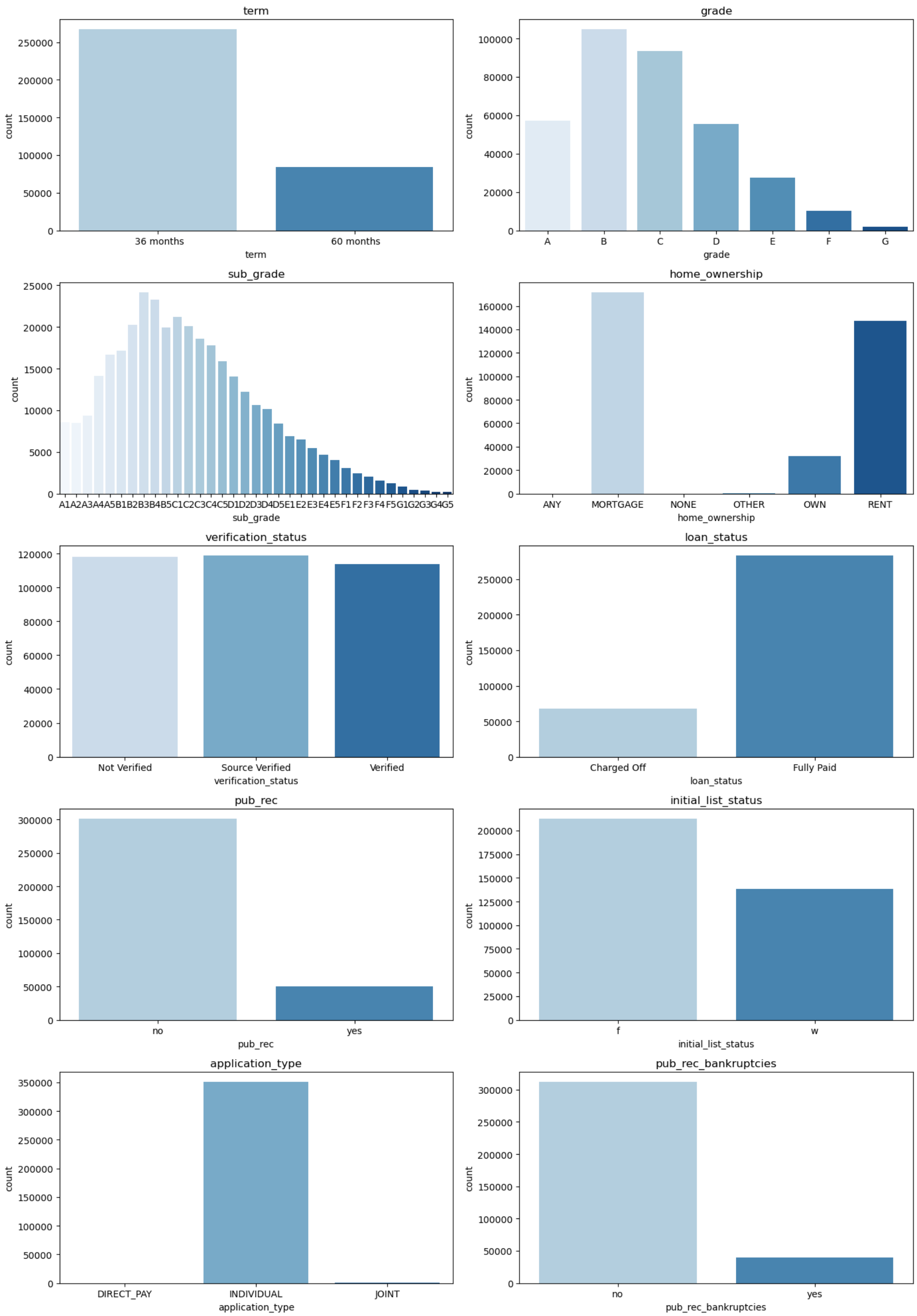
1. loan_amnt and installment are perfectly correlated
2. total_acc is highly correlated with open_acc
3. total_acc is moderately correlated with mort_acc

We can remove some of these correlated features to avoid multicolinearity

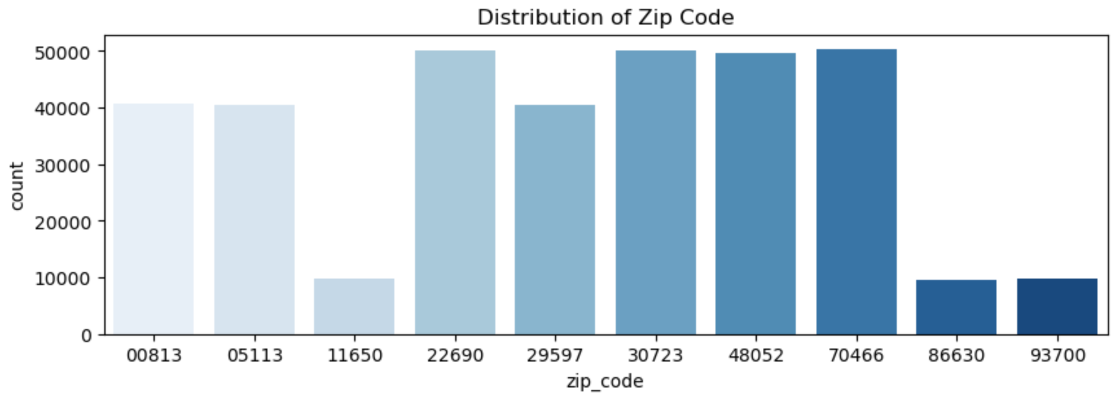
[53]:
```
#Drop installment
df.drop(columns=['installment'], inplace=True)
```

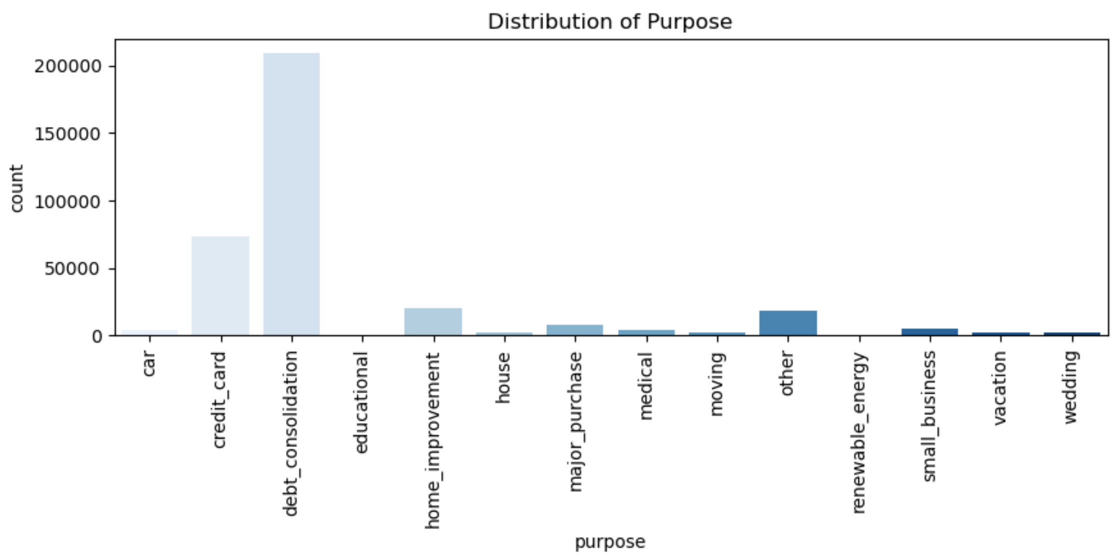
[54]:
```
#Distribution of categorical variables
plot = ['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'loan_status', 'pub_rec', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies']

plt.figure(figsize=(14,20))
i=1
for col in plot:
  ax=plt.subplot(5,2,i)
  sns.countplot(x=df[col], palette='Blues')
```

```
  plt.title(f'{col}')
  i += 1

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


term
count
36 months
60 months
term
grade
count
A
B
C
D
E
F
G
grade
sub_grade
count
A1A2A3A4A5B1B2B3B4B5C1C2C3C4C5D1D2D3D4D5E1E2E3E4E5F1F2F3F4F5G1G2G3G4G5
sub_grade
home_ownership
count
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership
verification_status
count
Not Verified
Source Verified
Verified
verification_status
loan_status
count
Charged Off
Fully Paid
loan_status
pub_rec
count
no
yes
pub_rec
initial_list_status
count
f
w
initial_list_status
application_type
count
DIRECT_PAY
INDIVIDUAL
JOINT
application_type
pub_rec_bankruptcies
count
no
yes
pub_rec_bankruptcies

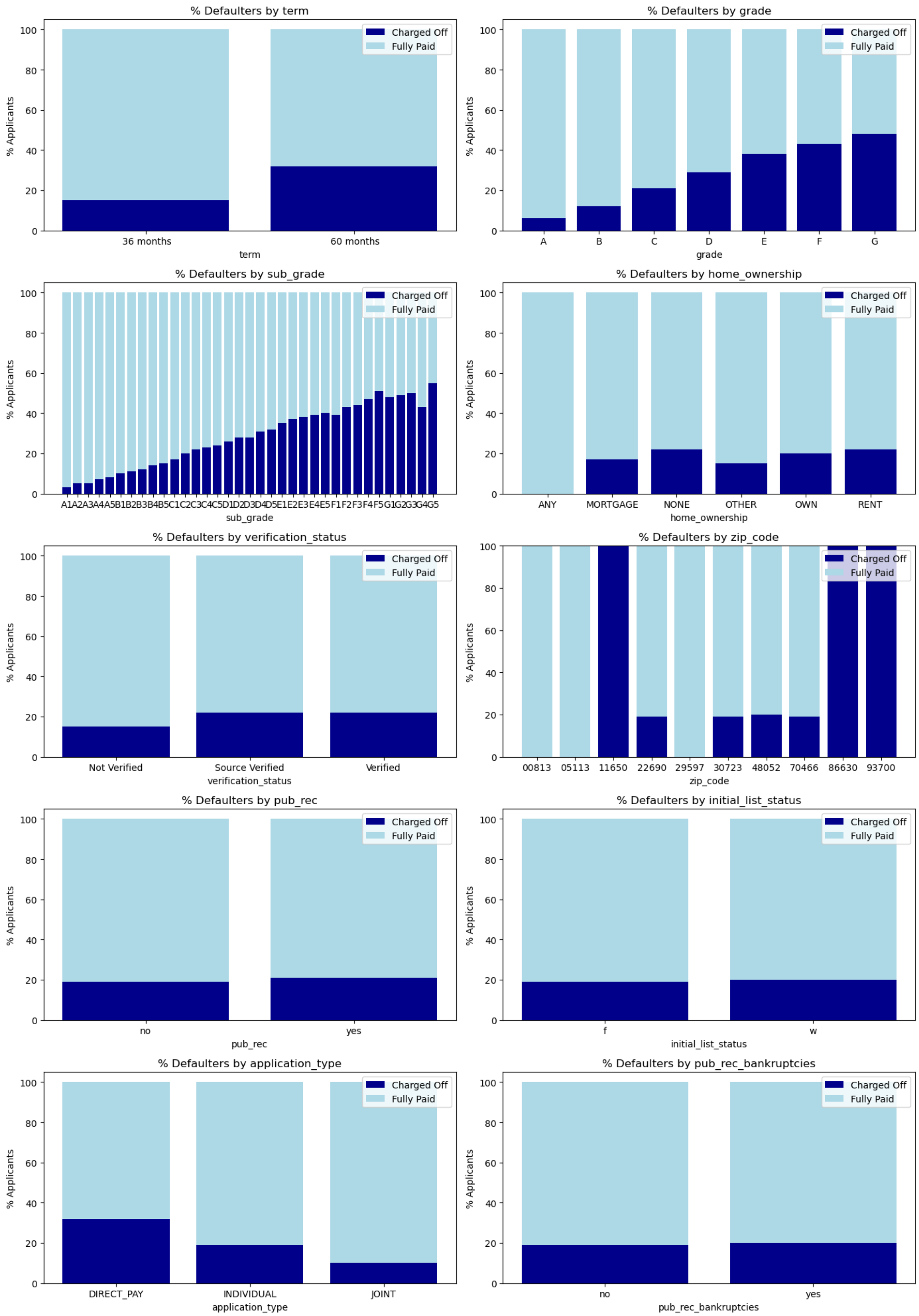
[55]:
```
plt.figure(figsize=(10,3))
sns.countplot(x=df['zip_code'], palette='Blues')
plt.title('Distribution of Zip Code')

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

plt.show()
```





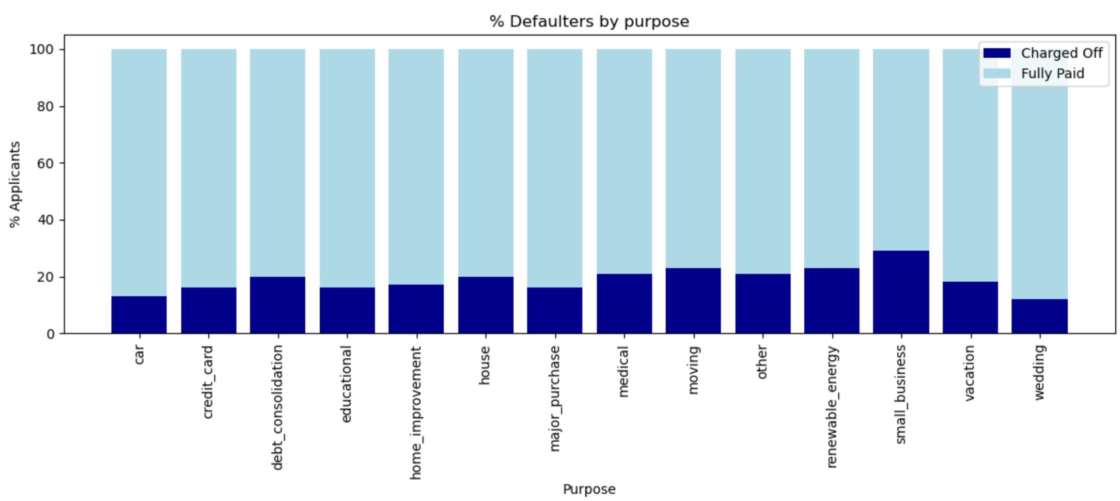
Observations: * Almost 80% loans are of 36 months term * Maximum loans (30%) fall in B grade, followed by C,A & D respectively * The type of home ownership for 50% cases is mortgage * The target variable (loan status) is imbalanced in the favour of fully-paid loans. Defaulters are approx 25% of fully paid instances. * 85% of applicants don't have a public record/haven't filled for bankruptcy * 99% applicants have applied under 'individual' application type * 55% of loans are taken for the purpose of debt consolidation followed by 20% on credit card

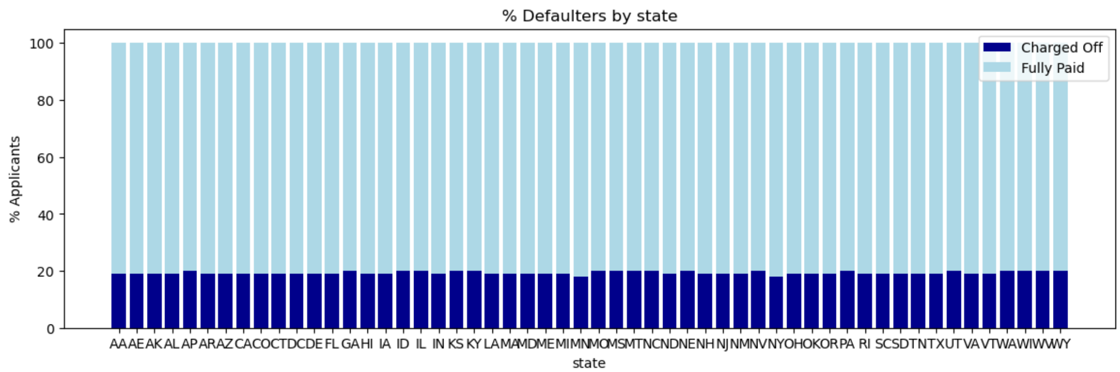
[56]:
```python
# Impact of categorical factors on loan status

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',␣
 ↪values='purpose')
  data = data.div(data.sum(axis=1), axis=0).multiply(100).round()
  data.reset_index(inplace=True)

  plt.bar(data[col],data['Charged Off'], color='#00008b')
  plt.bar(data[col],data['Fully Paid'], color='#add8e6', 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()
```


% Defaulters by term
36 months
60 months
term
% Defaulters by grade
A B C D E F G
grade
% Defaulters by sub_grade
A1A2A3A4A5B1B2B3B4B5C1C2C3C4C5D1D2D3D4D5E1E2E3E4E5F1F2F3F4F5G1G2G3G4G5
sub_grade
% Defaulters by home_ownership
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership
% Defaulters by verification_status
Not Verified
Source Verified
Verified
verification_status
% Defaulters by zip_code
00813 05113 11650 22690 29597 30723 48052 70466 86630 93700
zip_code
% Defaulters by pub_rec
no
yes
pub_rec
% Defaulters by initial_list_status
f
w
initial_list_status
% Defaulters by application_type
DIRECT_PAY
INDIVIDUAL
JOINT
application_type
% Defaulters by pub_rec_bankruptcies
no
yes
pub_rec_bankruptcies
% Applicants
Charged Off
Fully Paid
0
20
40
60
80
100

[57]:

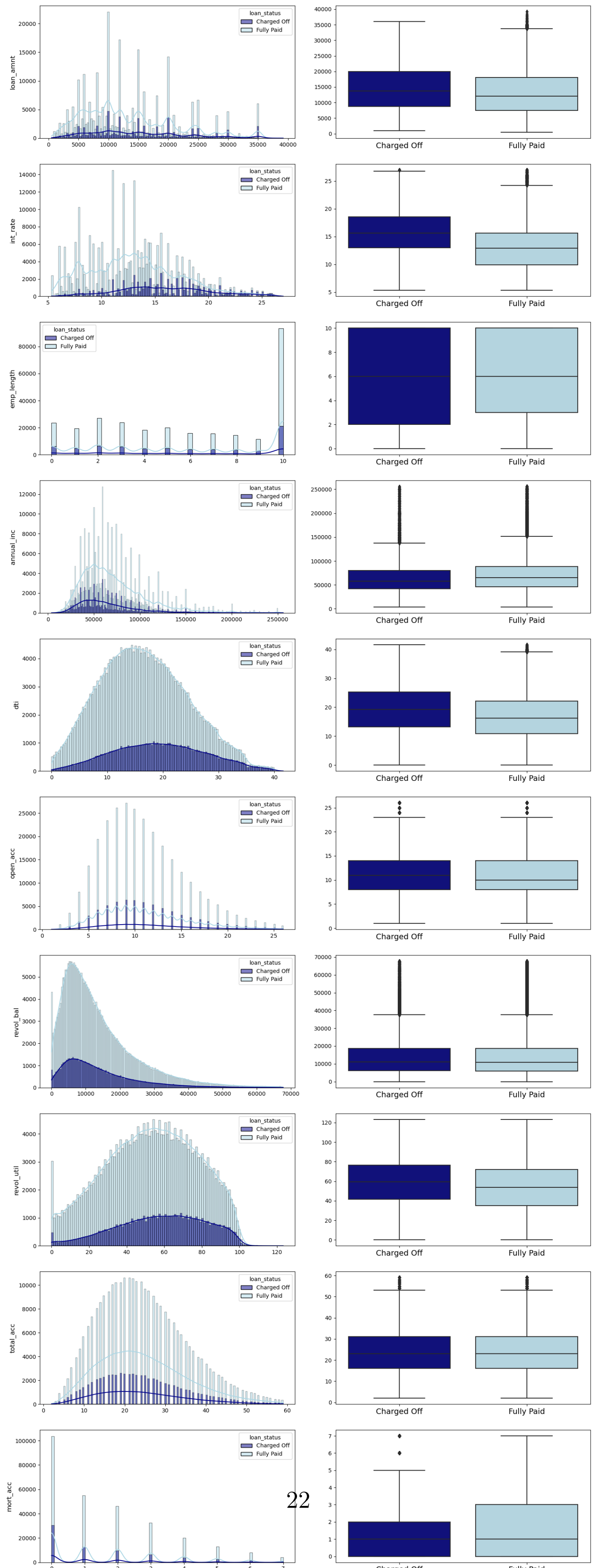
```python
# Impact of Purpose/state on loan status

purpose = df.pivot_table(index='purpose', columns='loan_status',␣
 ↪aggfunc='count', values='sub_grade')
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'], color='#00008b')
plt.bar(purpose['purpose'],purpose['Fully Paid'], color='#add8e6',␣
 ↪bottom=purpose['Charged Off'])
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',␣
 ↪values='sub_grade')
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'], color='#00008b')
plt.bar(state['state'],state['Fully Paid'], color='#add8e6',␣
 ↪bottom=state['Charged Off'])
plt.xlabel('state')
plt.ylabel('% Applicants')
plt.title('% Defaulters by state')
plt.legend(['Charged Off','Fully Paid'])
plt.show()
```





Observations: * The % of defaulters is much higher for longer (60-month) term * As expected, grade/sub-grade has the maximum impact on loan_status with highest grade having maximum defaulters * Zip codes such as 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 * Loan taken for the purpose of small business has the highest rate of default

[58]:
```
# 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=True,
                 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_status
Charged Off
Fully Paid
loan_amnt
int_rate
emp_length
annual_inc
dti
open_acc
revol_bal
revol_util
total_acc
mort_acc

Observations:
* From the boxplots, it can be observed that the mean loan_amnt, int_rate, dti, open_acc and revol_util are slightly higher for defaulters while annual income is lower

[59]:
```
# 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
```

4 Data Pre-Processing

[60]:
```
# Encoding Target Variable

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

[61]:
```
x = df.drop(columns=['loan_status'])
x.reset_index(inplace=True, drop=True)
y = df['loan_status']
y.reset_index(drop=True, inplace=True)
```

[62]:
```
# 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}).
 ↪astype(int)
```

One Hot Encoding of Categorical Features

[63]:
```
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_out(cat_cols))
x = pd.concat([x,encoded_df], axis=1)
x.drop(columns=cat_cols, inplace=True)
x.head()
```

[63]:
```
   loan_amnt  term  int_rate  emp_length  annual_inc    dti  open_acc  \
0    10000.0    36     11.44        10.0    117000.0  26.24      16.0
```

```
1    8000.0    36     11.99         4.0     65000.0  22.05      17.0
2   15600.0    36     10.49         0.0     43057.0  12.79      13.0
3    7200.0    36      6.49         6.0     54000.0   2.60       6.0
4   24375.0    60     17.27         9.0     55000.0  33.95      13.0

   pub_rec  revol_bal  revol_util  total_acc  mort_acc  pub_rec_bankruptcies  \
0        0    36369.0        41.8       25.0       0.0                     0
1        0    20131.0        53.3       27.0       3.0                     0
2        0    11987.0        92.2       26.0       0.0                     0
3        0     5472.0        21.5       13.0       0.0                     0
4        0    24584.0        69.8       43.0       1.0                     0

   grade_A  grade_B  grade_C  grade_D  grade_E  grade_F  grade_G  \
0      0.0      1.0      0.0      0.0      0.0      0.0      0.0
1      0.0      1.0      0.0      0.0      0.0      0.0      0.0
2      0.0      1.0      0.0      0.0      0.0      0.0      0.0
3      1.0      0.0      0.0      0.0      0.0      0.0      0.0
4      0.0      0.0      1.0      0.0      0.0      0.0      0.0

   home_ownership_ANY  home_ownership_MORTGAGE  home_ownership_NONE  \
0                 0.0                      0.0                  0.0
1                 0.0                      1.0                  0.0
2                 0.0                      0.0                  0.0
3                 0.0                      0.0                  0.0
4                 0.0                      1.0                  0.0

   home_ownership_OTHER  home_ownership_OWN  home_ownership_RENT  \
0                   0.0                 0.0                  1.0
1                   0.0                 0.0                  0.0
2                   0.0                 0.0                  1.0
3                   0.0                 0.0                  1.0
4                   0.0                 0.0                  0.0

   verification_status_Not Verified  verification_status_Source Verified  \
0                               1.0                                  0.0
1                               1.0                                  0.0
2                               0.0                                  1.0
3                               1.0                                  0.0
4                               0.0                                  0.0

   verification_status_Verified  purpose_car  purpose_credit_card  \
0                           0.0          0.0                  0.0
1                           0.0          0.0                  0.0
2                           0.0          0.0                  1.0
3                           0.0          0.0                  1.0
4                           1.0          0.0                  1.0
```

```
   purpose_debt_consolidation  purpose_educational  purpose_home_improvement  \
0                         0.0                  0.0                       0.0
1                         1.0                  0.0                       0.0
2                         0.0                  0.0                       0.0
3                         0.0                  0.0                       0.0
4                         0.0                  0.0                       0.0

   purpose_house  purpose_major_purchase  purpose_medical  purpose_moving  \
0            0.0                     0.0              0.0             0.0
1            0.0                     0.0              0.0             0.0
2            0.0                     0.0              0.0             0.0
3            0.0                     0.0              0.0             0.0
4            0.0                     0.0              0.0             0.0

   purpose_other  purpose_renewable_energy  purpose_small_business  \
0            0.0                       0.0                     0.0
1            0.0                       0.0                     0.0
2            0.0                       0.0                     0.0
3            0.0                       0.0                     0.0
4            0.0                       0.0                     0.0

   purpose_vacation  purpose_wedding  application_type_DIRECT_PAY  \
0               1.0              0.0                          0.0
1               0.0              0.0                          0.0
2               0.0              0.0                          0.0
3               0.0              0.0                          0.0
4               0.0              0.0                          0.0

   application_type_INDIVIDUAL  application_type_JOINT  zip_code_00813  \
0                          1.0                     0.0             0.0
1                          1.0                     0.0             0.0
2                          1.0                     0.0             0.0
3                          1.0                     0.0             1.0
4                          1.0                     0.0             0.0

   zip_code_05113  zip_code_11650  zip_code_22690  zip_code_29597  \
0             0.0             0.0             1.0             0.0
1             1.0             0.0             0.0             0.0
2             1.0             0.0             0.0             0.0
3             0.0             0.0             0.0             0.0
4             0.0             1.0             0.0             0.0

   zip_code_30723  zip_code_48052  zip_code_70466  zip_code_86630  \
0             0.0             0.0             0.0             0.0
1             0.0             0.0             0.0             0.0
2             0.0             0.0             0.0             0.0
3             0.0             0.0             0.0             0.0
```

```
4            0.0          0.0          0.0          0.0

   zip_code_93700
0            0.0
1            0.0
2            0.0
3            0.0
4            0.0
```

Train-Test Split

[64]:
```python
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.
 ↪20,stratify=y,random_state=42)
```

[65]:
```python
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[65]:
```
((280676, 56), (280676,), (70169, 56), (70169,))
```

Scaling Numeric Features

[67]:
```python
scaler = MinMaxScaler()
x_train = pd.DataFrame(scaler.fit_transform(x_train), columns=x_train.columns)
x_test = pd.DataFrame(scaler.transform(x_test), columns=x_test.columns)
```

[68]:
```python
x_train.tail()
```

[68]:
```
        loan_amnt  term  int_rate  emp_length  annual_inc       dti  open_acc  \
280671   0.167959   0.0  0.141671         0.7    0.194444  0.255954      0.60
280672   0.497416   0.0  0.445778         0.4    0.182540  0.414482      0.24
280673   0.064599   0.0  0.686664         0.7    0.238095  0.220111      0.32
280674   0.245478   1.0  0.177665         0.9    0.313492  0.134953      0.92
280675   0.646641   1.0  0.885095         0.6    0.349206  0.747173      0.88

        pub_rec  revol_bal  revol_util  total_acc  mort_acc  \
280671      0.0   0.104275    0.271695   0.578947  0.428571
280672      0.0   0.224536    0.670722   0.263158  0.285714
280673      0.0   0.249454    0.622871   0.385965  0.428571
280674      0.0   0.080701    0.039740   0.842105  0.428571
280675      1.0   0.213775    0.543390   0.596491  0.714286

        pub_rec_bankruptcies  grade_A  grade_B  grade_C  grade_D  grade_E  \
280671                   0.0      1.0      0.0      0.0      0.0      0.0
280672                   0.0      0.0      0.0      1.0      0.0      0.0
280673                   0.0      0.0      0.0      0.0      1.0      0.0
280674                   0.0      0.0      1.0      0.0      0.0      0.0
280675                   1.0      0.0      0.0      0.0      0.0      0.0

        grade_F  grade_G  home_ownership_ANY  home_ownership_MORTGAGE  \
```

```
280671      0.0      0.0                 0.0                        0.0
280672      0.0      0.0                 0.0                        0.0
280673      0.0      0.0                 0.0                        1.0
280674      0.0      0.0                 0.0                        1.0
280675      1.0      0.0                 0.0                        1.0

        home_ownership_NONE  home_ownership_OTHER  home_ownership_OWN  \
280671                  0.0                   0.0                 0.0
280672                  0.0                   0.0                 0.0
280673                  0.0                   0.0                 0.0
280674                  0.0                   0.0                 0.0
280675                  0.0                   0.0                 0.0

        home_ownership_RENT  verification_status_Not Verified  \
280671                  1.0                               1.0
280672                  1.0                               0.0
280673                  0.0                               0.0
280674                  0.0                               0.0
280675                  0.0                               0.0

        verification_status_Source Verified  verification_status_Verified  \
280671                                  0.0                           0.0
280672                                  0.0                           1.0
280673                                  1.0                           0.0
280674                                  0.0                           1.0
280675                                  0.0                           1.0

        purpose_car  purpose_credit_card  purpose_debt_consolidation  \
280671          0.0                  1.0                         0.0
280672          0.0                  0.0                         1.0
280673          0.0                  0.0                         0.0
280674          0.0                  0.0                         0.0
280675          0.0                  0.0                         0.0

        purpose_educational  purpose_home_improvement  purpose_house  \
280671                  0.0                       0.0            0.0
280672                  0.0                       0.0            0.0
280673                  0.0                       0.0            0.0
280674                  0.0                       0.0            0.0
280675                  0.0                       1.0            0.0

        purpose_major_purchase  purpose_medical  purpose_moving  \
280671                     0.0              0.0             0.0
280672                     0.0              0.0             0.0
280673                     0.0              0.0             1.0
280674                     0.0              0.0             0.0
280675                     0.0              0.0             0.0
```

```
        purpose_other  purpose_renewable_energy  purpose_small_business  \
280671            0.0                       0.0                     0.0
280672            0.0                       0.0                     0.0
280673            0.0                       0.0                     0.0
280674            0.0                       1.0                     0.0
280675            0.0                       0.0                     0.0

        purpose_vacation  purpose_wedding  application_type_DIRECT_PAY  \
280671               0.0              0.0                          0.0
280672               0.0              0.0                          0.0
280673               0.0              0.0                          0.0
280674               0.0              0.0                          0.0
280675               0.0              0.0                          0.0

        application_type_INDIVIDUAL  application_type_JOINT  zip_code_00813  \
280671                          1.0                     0.0             0.0
280672                          1.0                     0.0             1.0
280673                          1.0                     0.0             0.0
280674                          1.0                     0.0             1.0
280675                          1.0                     0.0             0.0

        zip_code_05113  zip_code_11650  zip_code_22690  zip_code_29597  \
280671             0.0             0.0             0.0             0.0
280672             0.0             0.0             0.0             0.0
280673             0.0             0.0             0.0             0.0
280674             0.0             0.0             0.0             0.0
280675             0.0             1.0             0.0             0.0

        zip_code_30723  zip_code_48052  zip_code_70466  zip_code_86630  \
280671             0.0             1.0             0.0             0.0
280672             0.0             0.0             0.0             0.0
280673             0.0             1.0             0.0             0.0
280674             0.0             0.0             0.0             0.0
280675             0.0             0.0             0.0             0.0

        zip_code_93700
280671             0.0
280672             0.0
280673             0.0
280674             0.0
280675             0.0
```

Oversampling with SMOTE

[69]:
```
# Oversampling to balance the target variable
```

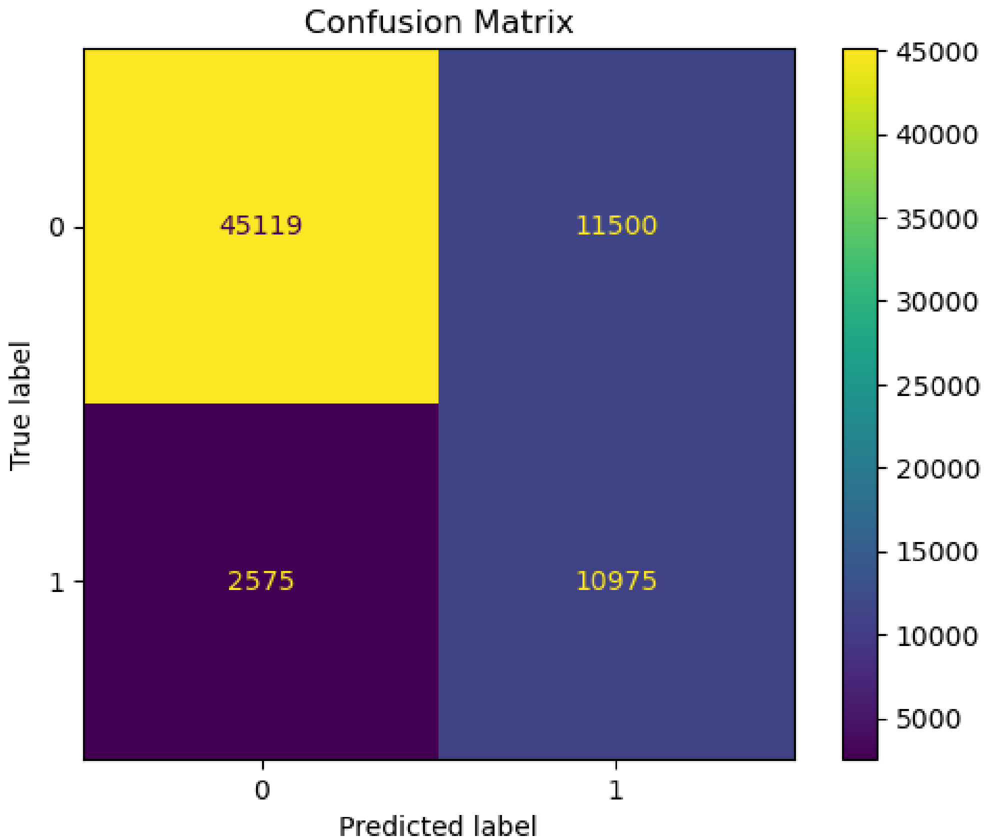
```
sm=SMOTE(random_state=42)
x_train_res, y_train_res = sm.fit_resample(x_train,y_train.ravel())

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
```

5 Logistic Regression

[70]:
```
model = LogisticRegression()
model.fit(x_train_res, y_train_res)
train_preds = model.predict(x_train)
test_preds = model.predict(x_test)

#Model Evaluation
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))

# Confusion Matrix
cm = confusion_matrix(y_test, test_preds)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```

```
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
```



Classification Report

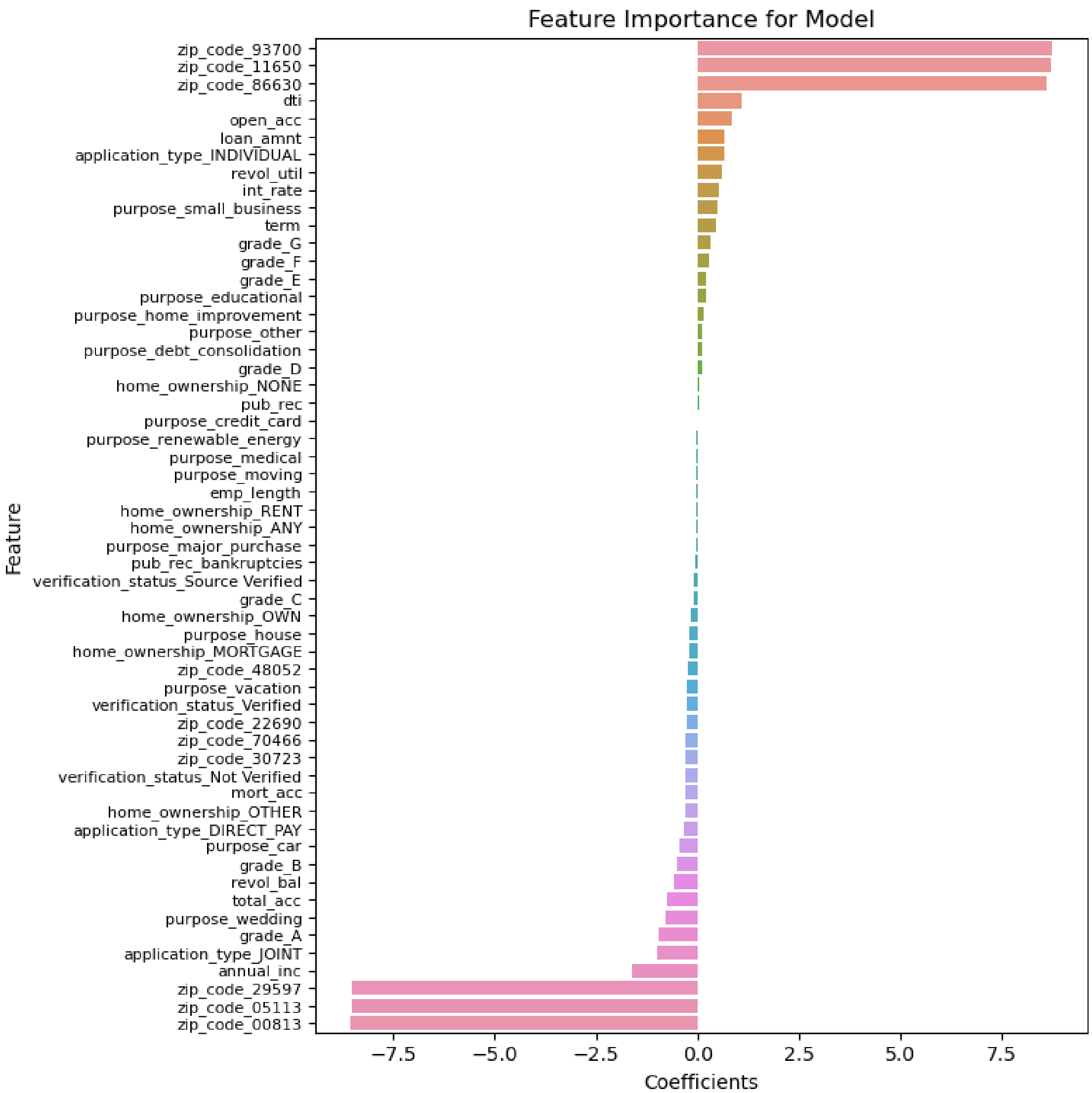
[71]:
```
print(classification_report(y_test, test_preds))
```

```
              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
```

- It can be observed 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).
- Although this model is effective in reducing NPAs by flagging most of the defaulters, 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%

[72]:

```
feature_imp = pd.DataFrame({'Columns':x_train.columns, 'Coefficients':model.
 ↪coef_[0]}).round(2).sort_values('Coefficients', ascending=False)

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()
```



- The model has assigned large weightage to zip_code features followed by dti, open_acc,

loan_amnt
- Similarly, large negative coefficients are assigned to a few zip codes, followed by annual income and joint application type

ROC Curve & AUC

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model. It helps evaluate and compare different models by illustrating the trade-off between the true positive rate (TPR) and false positive rate (FPR) at various classification thresholds.

The ROC curve is created by plotting the TPR on the y-axis against the FPR on the x-axis for different threshold values.

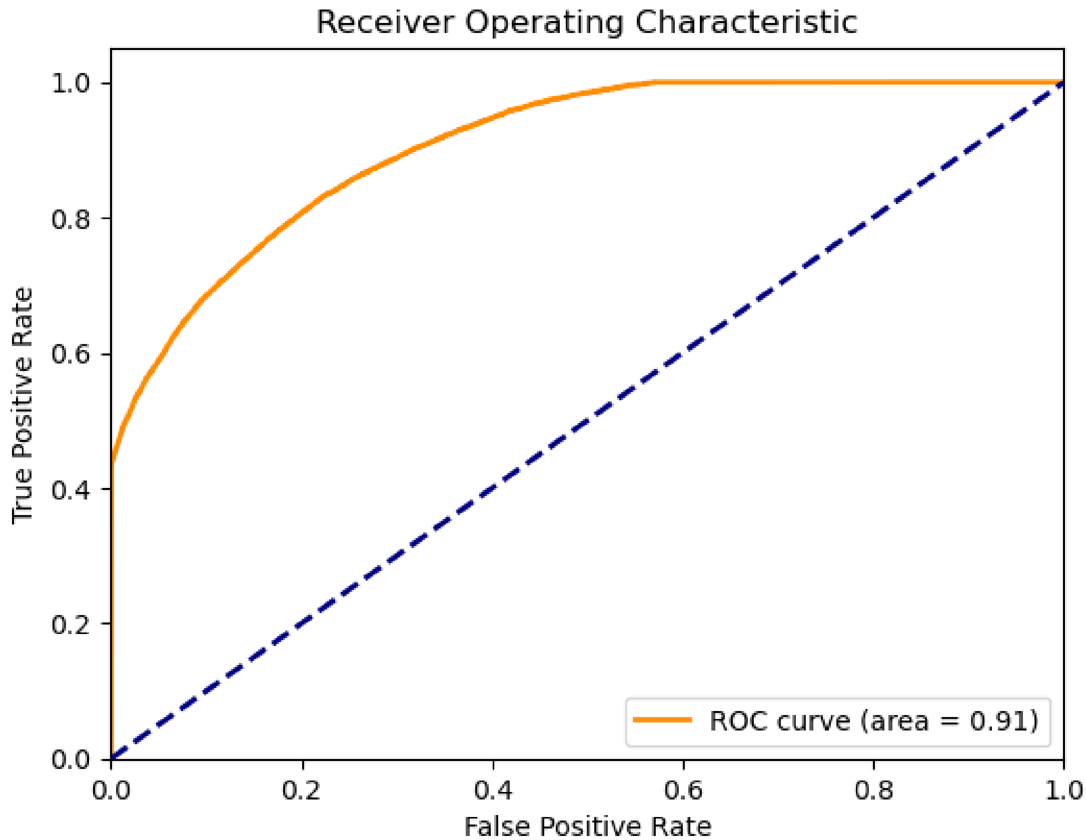
- TPR: Also known as sensitivity or recall, is the proportion of true positive predictions out of all actual positive instances.
- FPR: Proportion of false positive predictions out of all actual negative instances.

A perfect classifier would have a TPR of 1 and an FPR of 0, resulting in a point at the top-left corner of the ROC curve. On the other hand, a random classifier would have an ROC curve following the diagonal line, as it has an equal chance of producing true positive and false positive predictions.

The area under the ROC curve (AUC) is a commonly used metric to quantify the overall performance of a classifier.

A perfect classifier would have an AUC of 1, while a random classifier would have an AUC of 0.5. The higher the AUC value, the better the classifier's performance in distinguishing between positive and negative instances.

[73]:
```
# Predict probabilities for the test set
probs = model.predict_proba(x_test)[:,1]

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

# Compute 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)' %␣
 ↪roc_auc)
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()
```



- AUC of 0.91 signifies 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 because it may be high even when the classifier has a poor score on the minority class.

- This can happen when the classifier performs well on the majority class instances, which dominate the dataset. As a result, the AUC may appear high, but the model may not effectively identify the minority class instances.

Lets plot the Precision-Recall curve which is more suited for evaluation of imbalanced data

Precision Recall Curve

The Precision-Recall (PR) curve is another graphical representation commonly used to evaluate the performance of a binary classification model. It provides insights into the trade-off between precision and recall at various classification thresholds.

- **Precision** represents the proportion of correctly predicted positive instances out of all instances predicted as positive. It focuses on the accuracy of positive predictions.
- **Recall**, also known as sensitivity or true positive rate, represents the proportion of correctly predicted positive instances out of all actual positive instances. It focuses on capturing all positive instances.

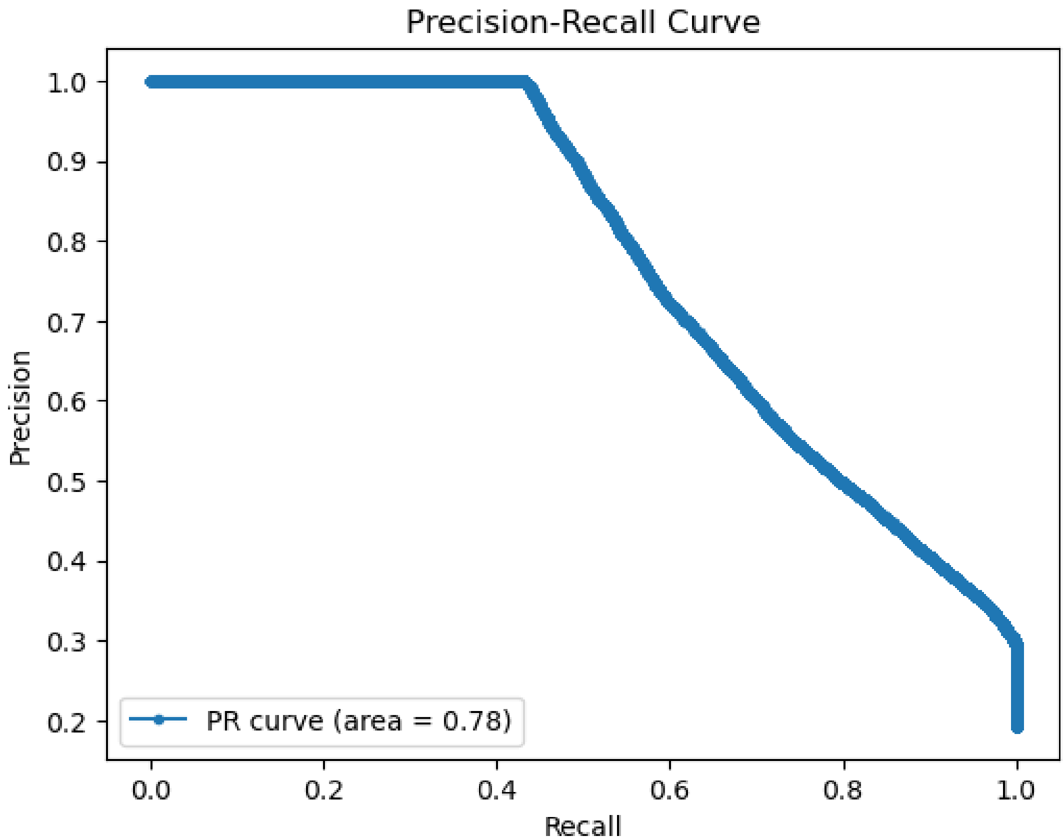
Similar to the ROC curve, the PR curve is created by plotting recall on the x-axis and precision on the y-axis for different threshold values. The curve illustrates the relationship between precision and recall as the classification threshold changes.

A perfect classifier would have a precision of 1 and a recall of 1, resulting in a point at the top-right corner of the PR curve. Conversely, a random classifier would have a PR curve following the horizontal line defined by the ratio of positive instances in the dataset.

The **PR curve is useful when dealing with imbalanced datasets**, where the number of negative instances far outweighs the positives. In such cases, the PR curve provides a more comprehensive evaluation of the model's performance compared to the ROC curve. This is because the ROC curve can be misleading when the majority of instances are negative, as it primarily focuses on the true negative rate.

The area under the PR curve (AUPRC) is a commonly used metric to quantify the overall performance of a classifier. A perfect classifier would have an AUPRC of 1, while a random classifier would have an AUPRC equal to the ratio of positive instances. Generally, a higher AUPRC indicates better performance.

[74]:
```
# 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)' % auprc)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```



As expected, the area under precision recall curve is not as high. It is a decent model as the area is more than 0.5 (random model benchmark) but there is still scope for improvement

6 Conclusion

Q1. 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.
Ans: Precision score is an indicator of type1 error. Increasing precision score of the model will minimise false positives and ensure that the company is not losing out on the opportunity to finance worthy individuals.

Q2. 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.
Ans: Recall score is an indicator of how many actual defaulters are flagged by the model. By increasing the recall score, we can minimise false negatives (type2 error) and ensure that loans are not disbursed to defaulters.

Insights

1) Impact of Categorical Attributes on loan_status (target variable):

- The % of defaulters is much higher for longer (60-month) term
- As expected, grade/sub-grade has the maximum impact on loan_status with highest grade having maximum defaulters
- Zip codes such as 11650, 86630 and 93700 have 100% defaulters
- We can remove initial_list_status and state as they have no impact on loan_status
- Direct pay application type has higher default rate compared to individual/joint
- Loan taken for the purpose of small business has the highest rate of default

2) Impact of Numerical Attributes on loan_status (target variable):

- It can be observed that the mean loan_amnt, int_rate, dti, open_acc and revol_util are higher for defaulters
- The mean annual income is lower for defaulters

3) A Logistic Regression model (trained after upsampling the data to balance the target variable) performed well, rendering accuracy of 80%.
4) The model had a precision score of 95%, recall score of 80%, and f1 score of 87% on the negative class
5) The model had a precision score of 49%, recall score of 81%, and f1 score of 61% on the positive class
6) The ROC plot shows that the area under ROC curve is 0.91, which signifies that the model is able to differentiate well between both classes
7) The area under Precision Recall curve is 0.78 (can be improved using hyperparameter tuning/increasing model complexity)

Recommendations * The optimal strategy to achieve the objective of balancing the risk of increasing NPAs by disbursing loans to defaulters with the opportunity to earn interest by disbursing loans to as many worthy customers as possible: maximise the F1 score along with the area under Precision Recall Curve (precision-recall trade-off) * More complex classifiers like random forest would give better results compared to logistic regression because they are not restricted by the linearity of decision boundary

[ ]:

```

```