LoanTap_LogisticRegression

May 2, 2024

[74]:
```
import pandas as pd
import numpy as np
```

[75]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

[129]:
```
from scipy import stats
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import (accuracy_score, confusion_matrix, roc_curve, auc,␣
↪confusion_matrix, ConfusionMatrixDisplay,f1_score, recall_score,␣
↪precision_score, precision_recall_curve, average_precision_score,␣
↪classification_report)
```

[77]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

[78]:
```
df = pd.read_csv("logistic_regression.csv")
df.shape
```

[78]:
```
(396030, 27)
```

[79]:
```
df.head()
```

[79]:
```
   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  …

   open_acc pub_rec revol_bal revol_util total_acc  initial_list_status  \
0      16.0     0.0   36369.0       41.8      25.0                    w
1      17.0     0.0   20131.0       53.3      27.0                    f
2      13.0     0.0   11987.0       92.2      26.0                    f
3       6.0     0.0    5472.0       21.5      13.0                    f
4      13.0     0.0   24584.0       69.8      43.0                    f

   application_type  mort_acc  pub_rec_bankruptcies  \
0        INDIVIDUAL       0.0                   0.0
1        INDIVIDUAL       3.0                   0.0
2        INDIVIDUAL       0.0                   0.0
3        INDIVIDUAL       0.0                   0.0
4        INDIVIDUAL       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

[5 rows x 27 columns]
```

[80]:
```
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
```

0.0.1 Data cleaning

[81]:
```
df.describe()
```

[81]:
```
           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
```

[82]:
```python
cat_cols = df.select_dtypes(include='object').columns
cat_cols
```

[82]:
```
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')
```

[83]:
```python
for col in cat_cols:
  print(f"No. of unique values in column {col:<25}: {df[col].nunique()}")
```

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

[84]:
```python
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
df['issue_d'] = pd.to_datetime(df['issue_d'])
```

```
<ipython-input-84-fd8269940f9a>:1: UserWarning: Could not infer format, so each
element will be parsed individually, falling back to `dateutil`. To ensure
parsing is consistent and as-expected, please specify a format.
  df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
<ipython-input-84-fd8269940f9a>:2: UserWarning: Could not infer format, so each
element will be parsed individually, falling back to `dateutil`. To ensure
parsing is consistent and as-expected, please specify a format.
  df['issue_d'] = pd.to_datetime(df['issue_d'])
```

[85]:
```python
df['emp_length'].value_counts()
```

[85]: emp_length
```
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: count, dtype: int64
```

[86]:
```
#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)
```

[87]:
```
#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')
```

[88]:
```
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                 394274 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
```

0.0.2 Check for Duplicate Values

[89]:
```
df.duplicated().sum()
```

[89]:
```
0
```

- There are no duplicate instances in the data.

0.0.3 Handling Missing Values

[90]:
```
df.isna().sum()
```

[90]:
```
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
```

[91]:
```
#Filling missing values with 'Unknown' for object dtype
fill_values = {'title': 'Unknown', 'emp_title': 'Unknown'}
df.fillna(value=fill_values, inplace=True)
```

[92]:
```
#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):
    if np.isnan(total_acc):
      return np.nan  # Return NaN for missing values in both columns
    else:
      return avg_mort[total_acc].round()
  else:
    return mort_acc
```

[93]:
```
df['mort_acc'] = df.apply(lambda x: fill_mort(x['total_acc'],x['mort_acc']),
 ↪axis=1)
```

[94]:
```
df.dropna(inplace=True)
```

[95]:
```
df.isna().sum()
```

[95]:
```
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
```

[96]:
```
df.shape
```

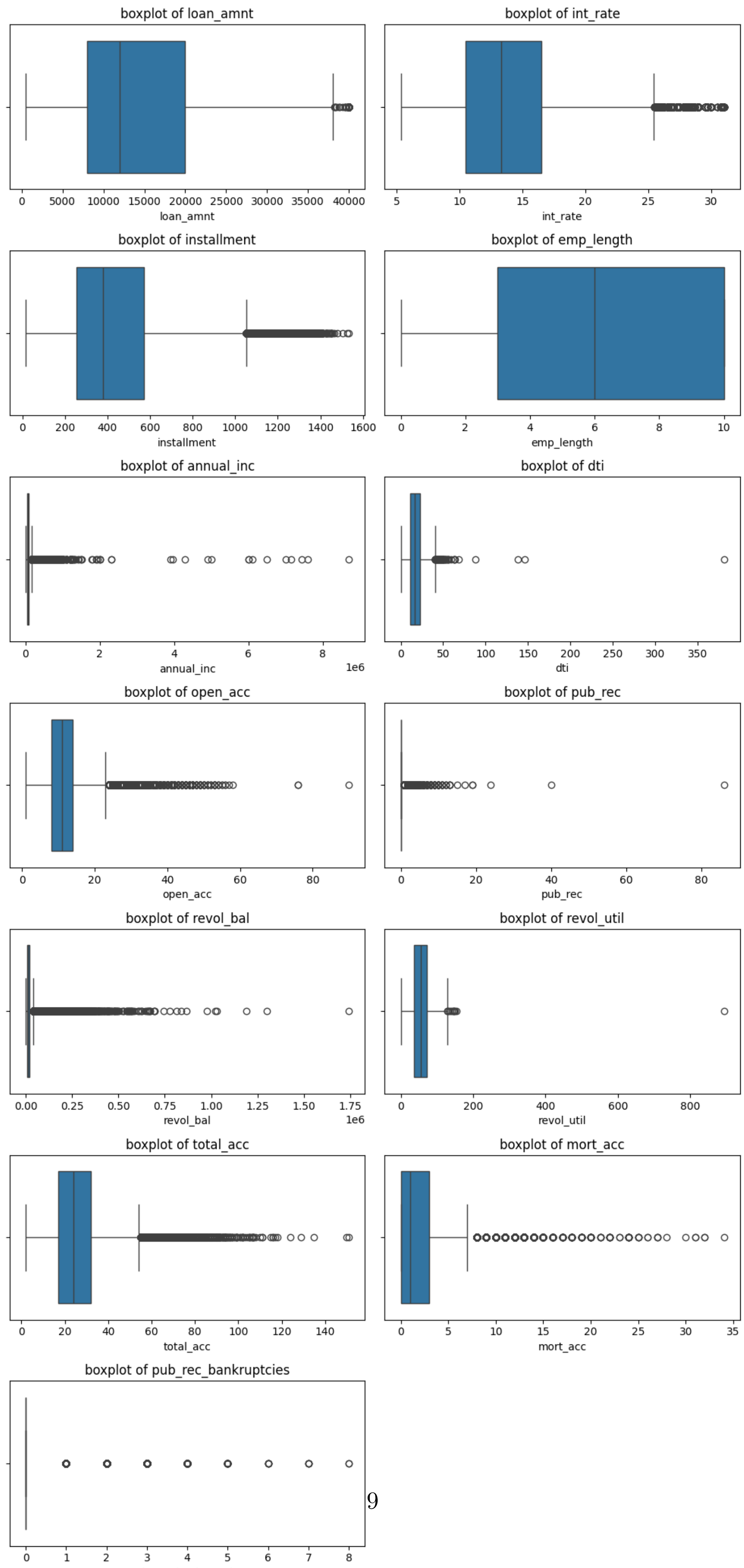
[96]: (376929, 27)

0.0.4 Outlier Treatment

[97]:
```
num_cols = df.select_dtypes(include='number').columns
num_cols
```

[97]:
```
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')
```

[98]:
```
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
loan_amnt
boxplot of int_rate
int_rate
boxplot of installment
installment
boxplot of emp_length
emp_length
boxplot of annual_inc
annual_inc
1e6
boxplot of dti
dti
boxplot of open_acc
open_acc
boxplot of pub_rec
pub_rec
boxplot of revol_bal
revol_bal
1e6
boxplot of revol_util
revol_util
boxplot of total_acc
total_acc
boxplot of mort_acc
mort_acc
boxplot of pub_rec_bankruptcies
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

[99]:
```
# 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')
```

[100]:
```
# Numeric columns after converting public records to category
num_cols = df.select_dtypes(include='number').columns
num_cols
```

[100]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

[101]:
```
#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)]
```

[102]:
```
df.shape
```

[102]:
```
(350845, 27)
```

0.0.5 Feature Engineering

[103]:
```
df['address'].sample(10)
```

[103]:
```
287902         26152 Jason Gardens\r\nNixonburgh, ME 48052
78203                         USNS Robinson\r\nFPO AE 00813
257459              049 Ware Avenue\r\nGregoryland, FL 70466
364063    85118 Tamara Court Apt. 063\r\nWest Marc, CT 3…
98347     3244 Jacob Locks Suite 469\r\nNorth Jessicamou…
387964            457 Terrell Drives\r\nEast Mindy, ME 93700
304892    87228 Nguyen Drives Apt. 885\r\nMcculloughches…
59862     662 Dorothy Heights Suite 175\r\nWest Kaylee, …
347284          2927 Clark Stravenue\r\nSanchezmouth, NC 48052
160835    11342 Frank Corner Apt. 727\r\nSouth Robertbur…
```

```
Name: address, dtype: object
```

[104]:
```
# Deriving zip code and state from address
df[['state', 'zip_code']] = df['address'].apply(lambda x: pd.Series([x[-8:-6],␣
↪x[-5:]]))
```

[105]:
```
df[['state', 'zip_code']].head()
```

[105]:

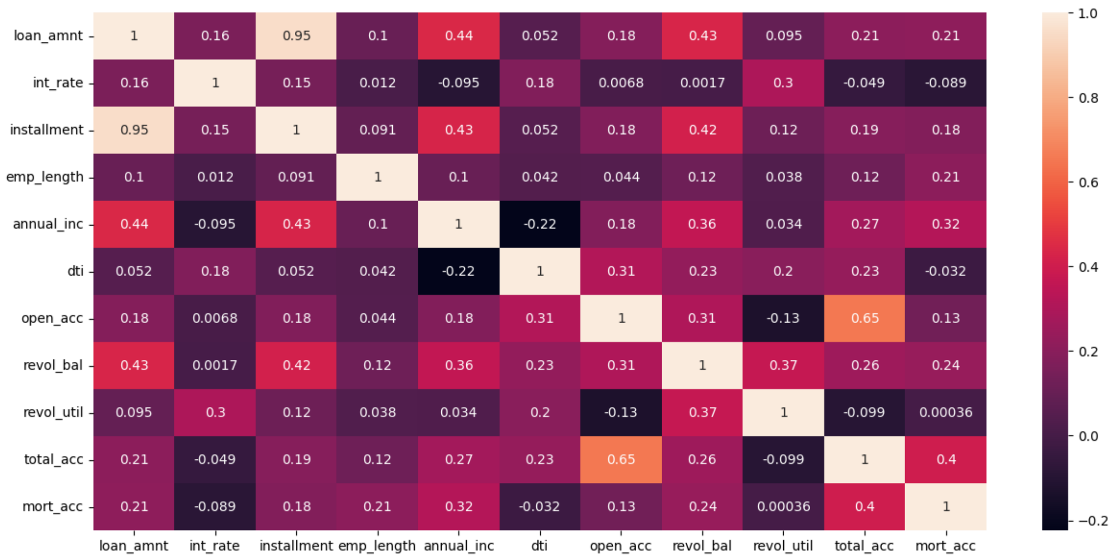
	state	zip_code
0	OK	22690
1	SD	05113
2	WV	05113
3	MA	00813
4	VA	11650

[106]:
```
#Drop address
df.drop(["address"], axis = 1, inplace=True)
```

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

0.1 Exploratory Data Analysis

[110]:
```
# prompt: Correlation between numerical features

corr = df[num_cols].corr()
plt.figure(figsize=(15,7))
sns.heatmap(corr, annot=True)
plt.show()
```



- loan_amnt and installment are perfectly correlated
- total_acc is highly correlated with open_acc
- total_acc is moderately correlated with mort_acc

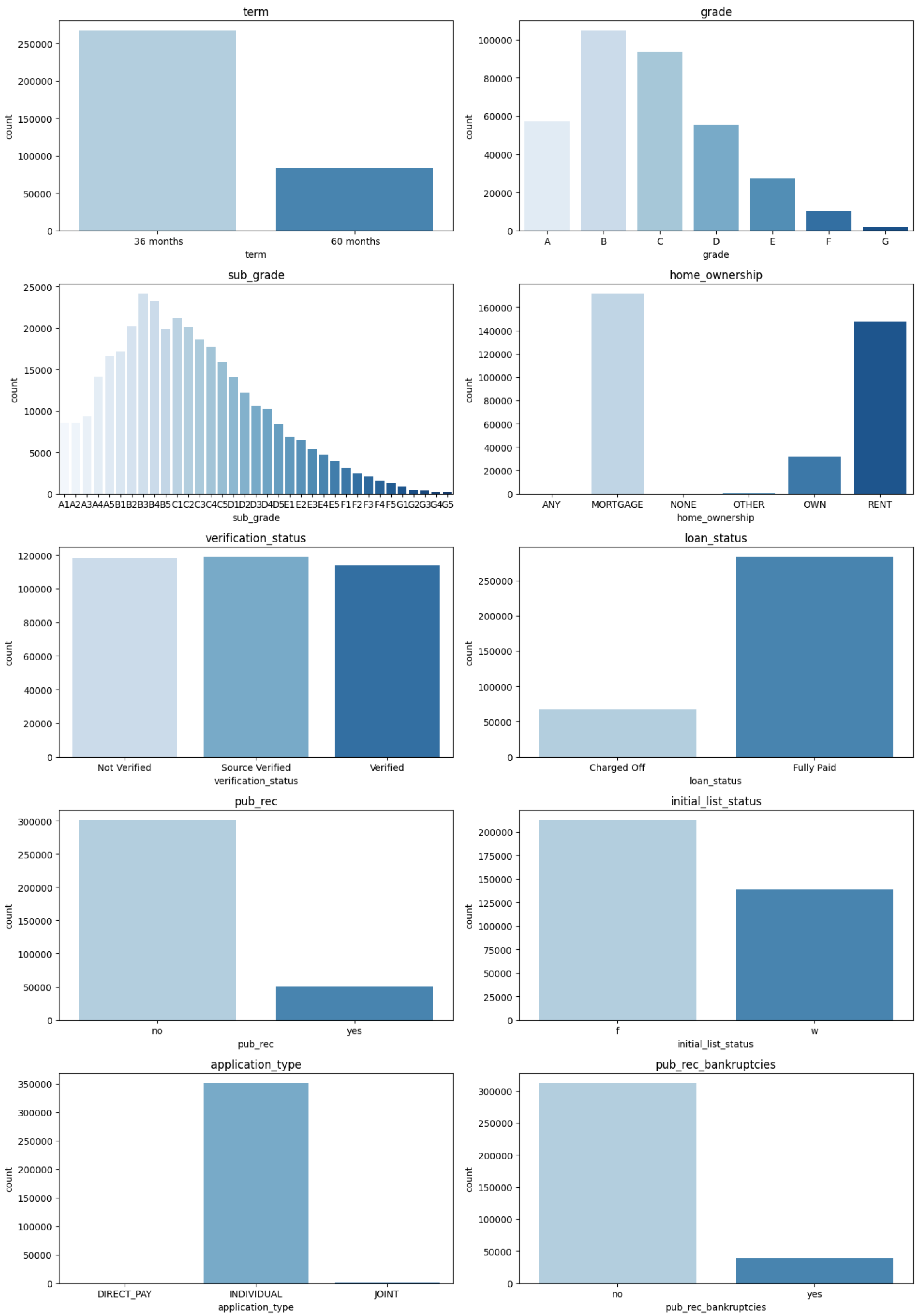
We can remove some of these correlated features to avoid multicolinearity

[111]:
```
#Drop installment
df.drop(columns=['installment'], inplace=True)
```

[112]:
```
#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()
```

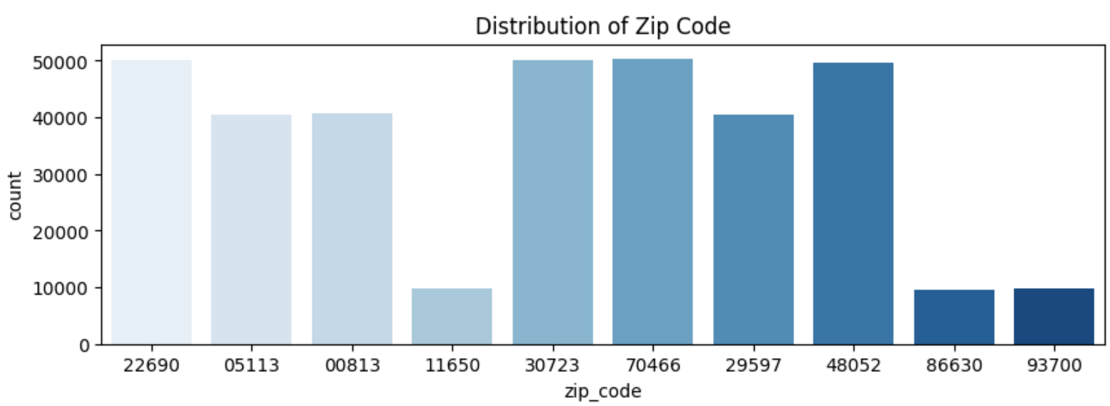
```
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:
```

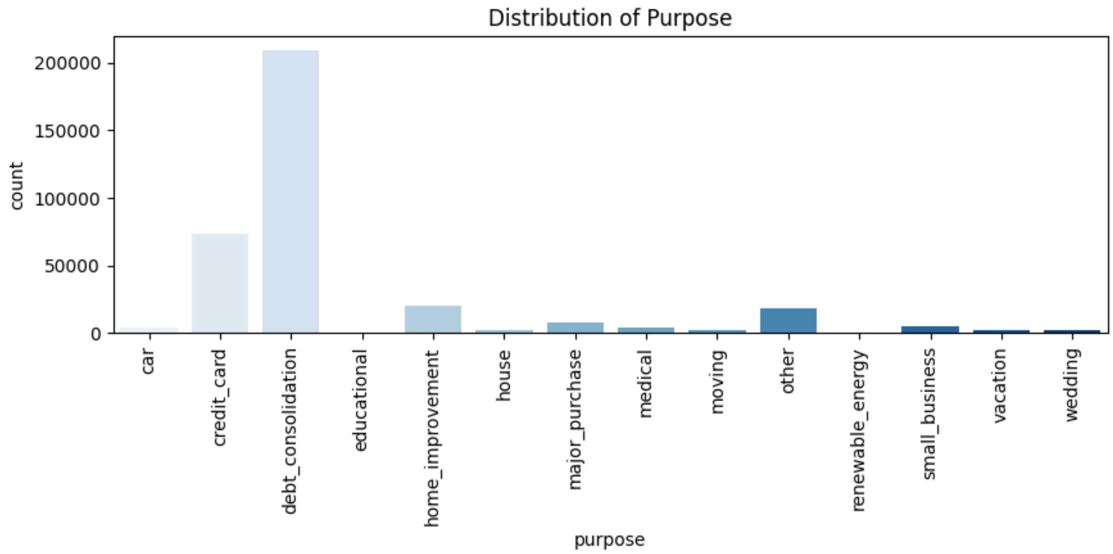
```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
<ipython-input-112-b8e89bce3ab2>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df[col], palette='Blues')
```


term
count
36 months
60 months
grade
A
B
C
D
E
F
G
sub_grade
A1A2A3A4A5B1B2B3B4B5C1C2C3C4C5D1D2D3D4D5E1 E2 E3 E4 E5 F1 F2 F3 F4 F5 G1G2G3G4G5
home_ownership
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
verification_status
Not Verified
Source Verified
Verified
loan_status
Charged Off
Fully Paid
pub_rec
no
yes
initial_list_status
f
w
application_type
DIRECT_PAY
INDIVIDUAL
JOINT
pub_rec_bankruptcies
no
yes

[113]:
```
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()
```

```
<ipython-input-113-7325fb61dba9>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df['zip_code'], palette='Blues')
<ipython-input-113-7325fb61dba9>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=df['purpose'], palette='Blues')
```



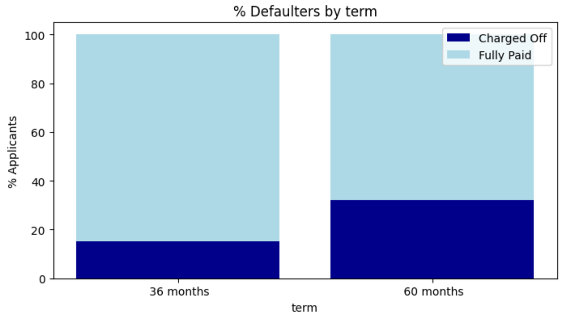


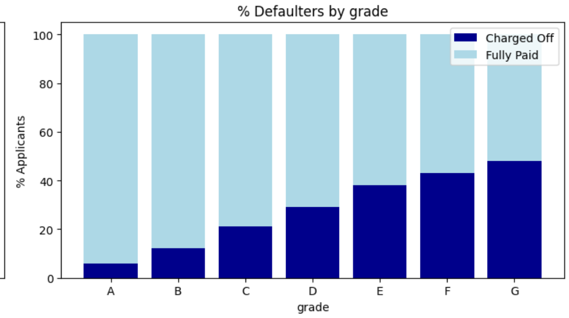
Observations:

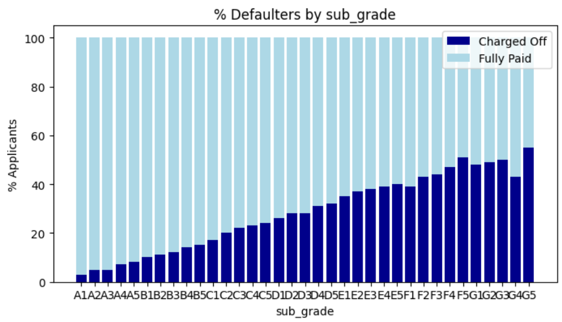
- Almost 80% loans are of 36 months term
- Maximum loans (30%) fall in B grade, followed by C,A & D respectively
- The type of home ownership for 50% cases is mortgage
- The target variable (loan status) is imbalanced in the favour of fully-paid * * loans. Defaulters are approx 25% of fully paid instances.
- 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

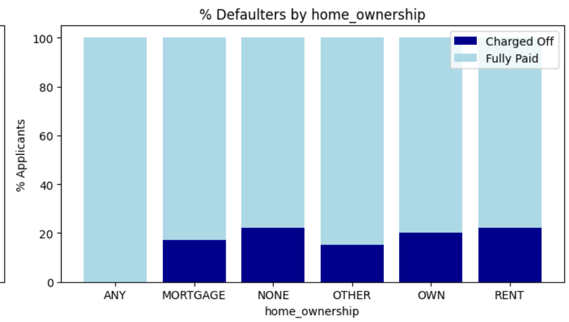
[114]:
```
# Impact of categorical factors on loan status

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')
```

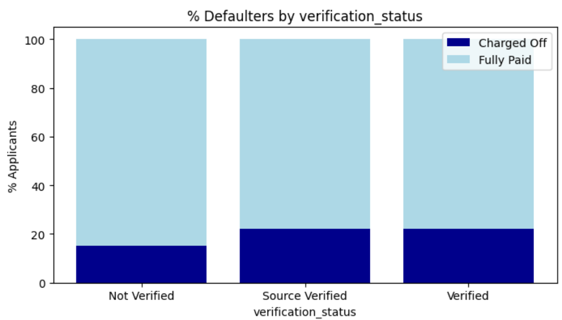
```
    plt.bar(data[col],data['Fully Paid'], color='#add8e6', bottom=data['Charged␣
 ↪Off'])
    plt.xlabel(f'{col}')
    plt.ylabel('% Applicants')
    plt.title(f'% Defaulters by {col}')
    plt.legend(['Charged Off','Fully Paid'])
    i += 1

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

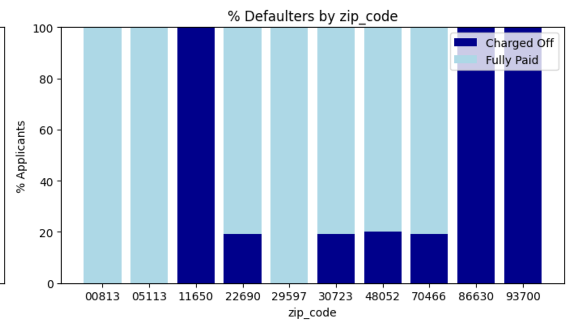

% Defaulters by term
Charged Off
Fully Paid
% Applicants
36 months
60 months
term

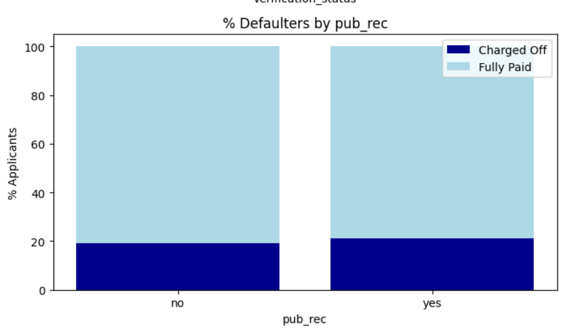

% Defaulters by grade
Charged Off
Fully Paid
% Applicants
A
B
C
D
E
F
G
grade

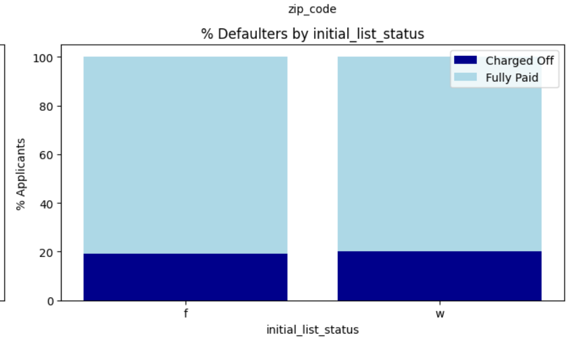

% Defaulters by sub_grade
Charged Off
Fully Paid
% Applicants
A1A2A3A4A5B1B2B3B4B5C1C2C3C4C5D1D2D3D4D5E1E2E3E4E5F1F2F3F4F5G1G2G3G4G5
sub_grade

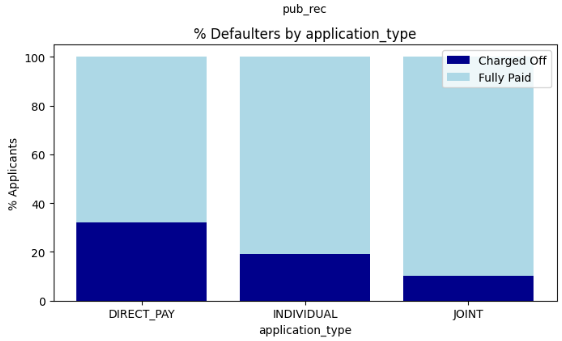

% Defaulters by home_ownership
Charged Off
Fully Paid
% Applicants
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership

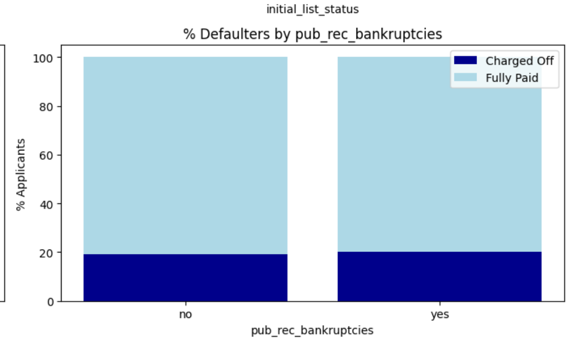

% Defaulters by verification_status
Charged Off
Fully Paid
% Applicants
Not Verified
Source Verified
Verified
verification_status


% Defaulters by zip_code
Charged Off
Fully Paid
% Applicants
00813
05113
11650
22690
29597
30723
48052
70466
86630
93700
zip_code


% Defaulters by pub_rec
Charged Off
Fully Paid
% Applicants
no
yes
pub_rec

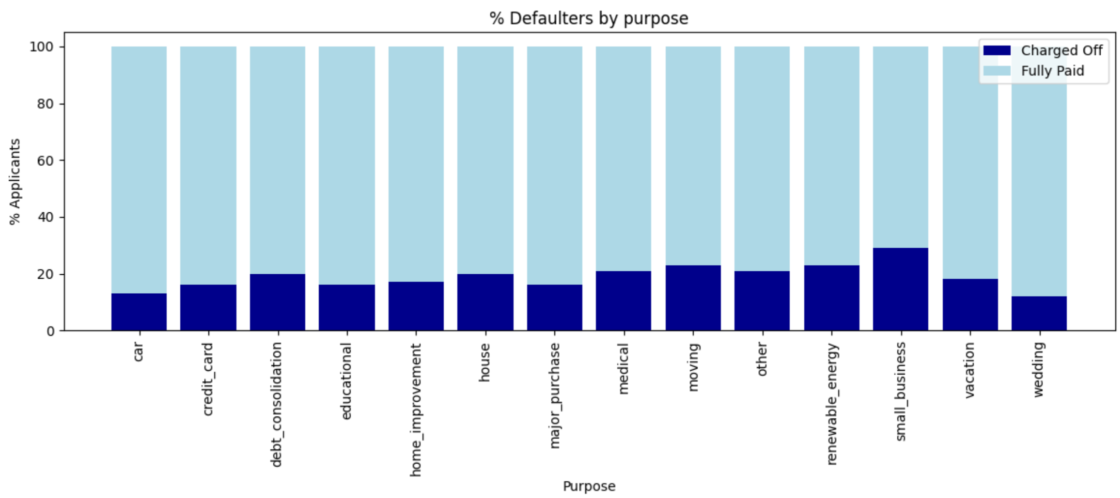

% Defaulters by initial_list_status
Charged Off
Fully Paid
% Applicants
f
w
initial_list_status

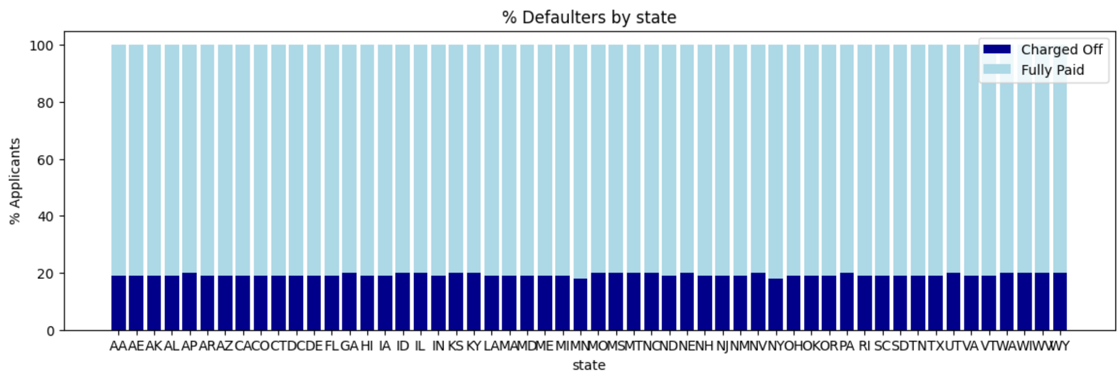

% Defaulters by application_type
Charged Off
Fully Paid
% Applicants
DIRECT_PAY
INDIVIDUAL
JOINT
application_type


% Defaulters by pub_rec_bankruptcies
Charged Off
Fully Paid
% Applicants
no
yes
pub_rec_bankruptcies

[115]:

```
# 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:

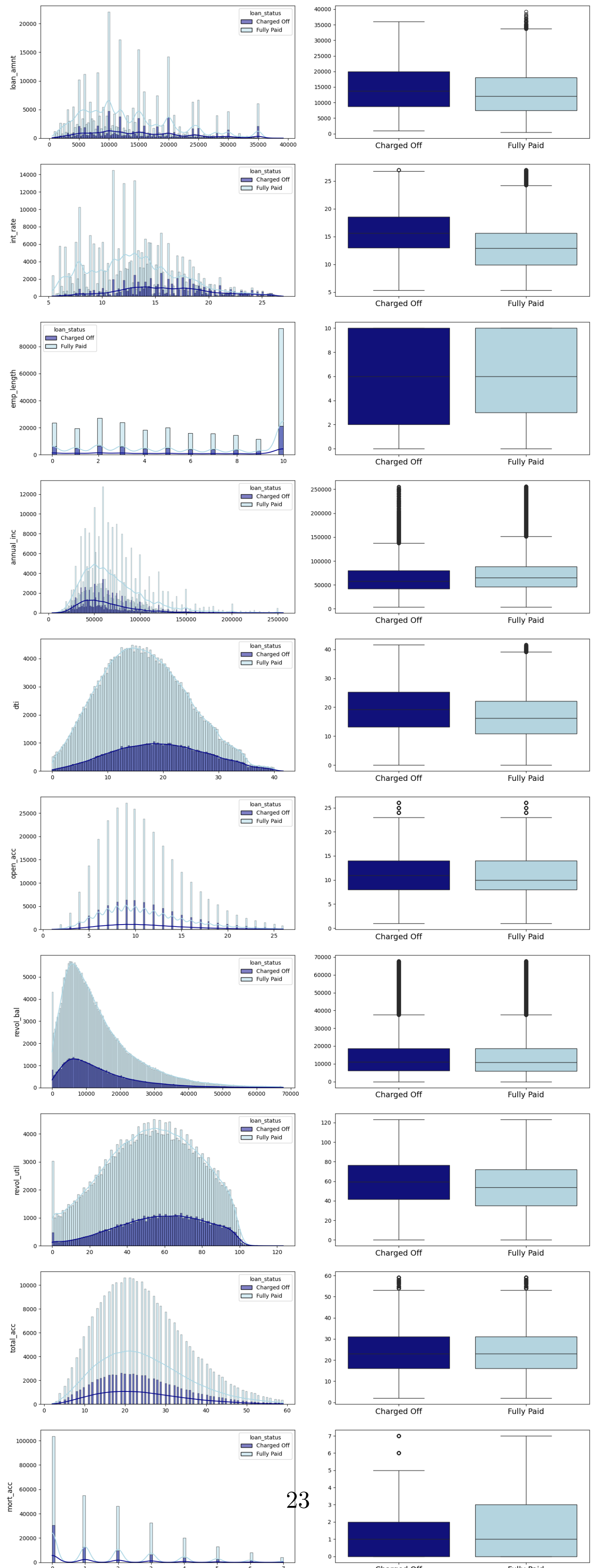
- The % of defaulters is much higher for longer (60-month) term
- As expected, grade/sub-grade has the maximum impact on loan_status with highest grade having maximum defaulters
- Zip codes such as 11650, 86630 and 93700 have 100% defaulters
- We can remove initial_list_status and state as they have 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

[117]:
```
# Impact of numerical features on loan_status
import matplotlib
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()
```

```
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
<ipython-input-117-c69223389652>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
```


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

[118]:
```
# 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
```

0.2 Data Pre-Processing

[119]:
```
# Encoding Target Variable

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

[120]:
```
x = df.drop(columns=['loan_status'])
x.reset_index(inplace=True, drop=True)
y = df['loan_status']
y.reset_index(drop=True, inplace=True)
```

[121]:
```
# 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)
```

0.2.1 One Hot Encoding of Categorical Features

[122]:
```
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()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868:
FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will
```

```
be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
```

[122]:

```
   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  …  purpose_medical  purpose_moving  \
0        0    36369.0        41.8  …              0.0             0.0
1        0    20131.0        53.3  …              0.0             0.0
2        0    11987.0        92.2  …              0.0             0.0
3        0     5472.0        21.5  …              0.0             0.0
4        0    24584.0        69.8  …              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
0                          1.0                     0.0
1                          1.0                     0.0
2                          1.0                     0.0
3                          1.0                     0.0
4                          1.0                     0.0

[5 rows x 47 columns]
```

0.2.2 Train-Test Split

[123]:

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

[124]:

```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[124]: ((280676, 47), (280676,), (70169, 47), (70169,))

0.2.3 Scaling Numeric Features

[125]:
```
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)
```

[126]:
```
x_train.tail()
```

[126]:
```
        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  …  purpose_medical  purpose_moving  \
280671      0.0   0.104275    0.271695  …              0.0             0.0
280672      0.0   0.224536    0.670722  …              0.0             0.0
280673      0.0   0.249454    0.622871  …              0.0             1.0
280674      0.0   0.080701    0.039740  …              0.0             0.0
280675      1.0   0.213775    0.543390  …              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
280671                          1.0                     0.0
280672                          1.0                     0.0
280673                          1.0                     0.0
280674                          1.0                     0.0
280675                          1.0                     0.0

[5 rows x 47 columns]
```

0.2.4 Oversampling with SMOTE

[127]:
```
# 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
```

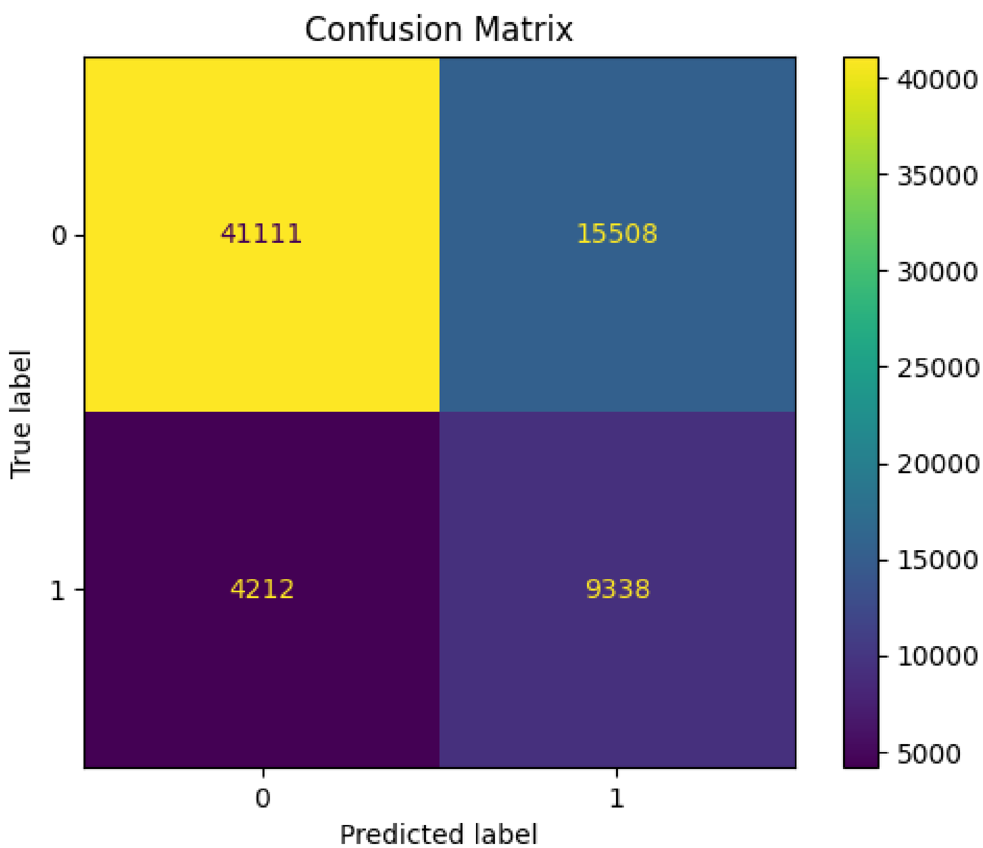
0.3 Logistic Regression

[130]:
```
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()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Train Accuracy : 0.72
Train F1 Score: 0.49
Train Recall Score: 0.69
Train Precision Score: 0.38

Test Accuracy : 0.72
Test F1 Score: 0.49
Test Recall Score: 0.69
Test Precision Score: 0.38
```



0.3.1 Classification Report

[131]:
```
print(classification_report(y_test, test_preds))
```

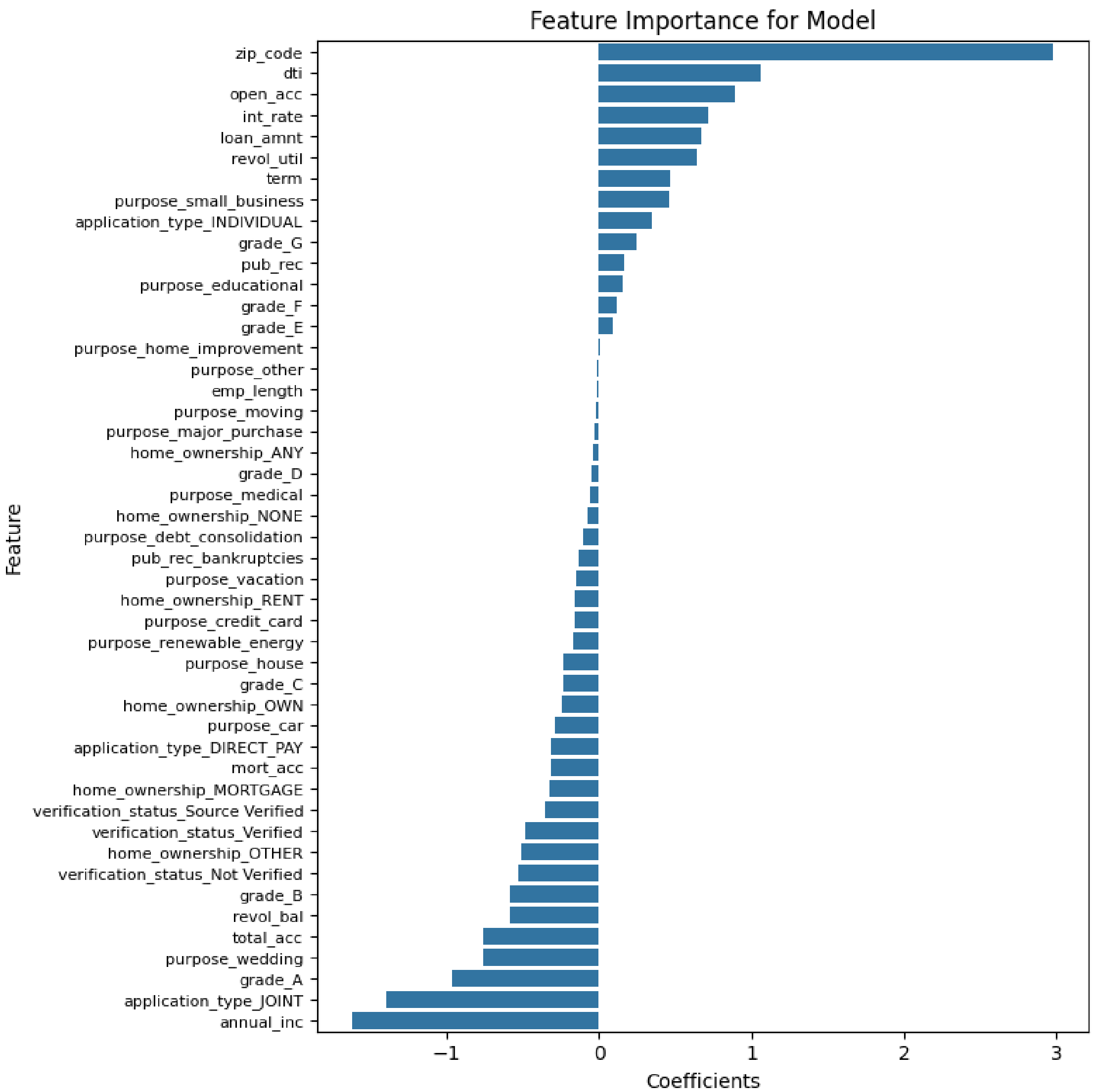
```
              precision    recall  f1-score   support
```

```
             0       0.91      0.73      0.81     56619
             1       0.38      0.69      0.49     13550

      accuracy                           0.72     70169
     macro avg       0.64      0.71      0.65     70169
  weighted avg       0.80      0.72      0.74     70169
```

- It can be observed that the recall score is very high (our model is able to identify 73% 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 49% even though accuracy is 80%

0.3.2 Feature Importance

[132]:
```
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()
```

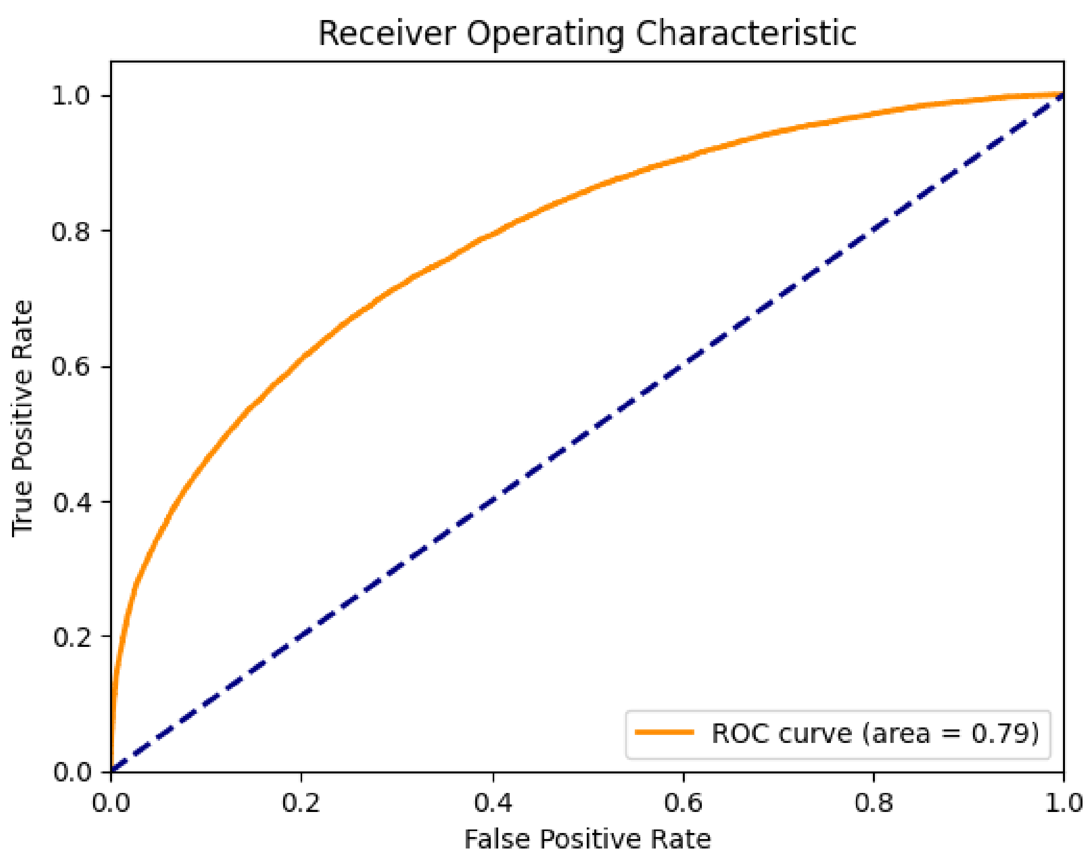


- The model has assigned large weightage to zip_code features followed by dti, open_acc, loan_amnt
- Similarly, large negative coefficients are assigned to a few zip code, followed by annual income and joint application type

0.3.3 ROC Curve & AUC

[133]:
```
# 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)
```

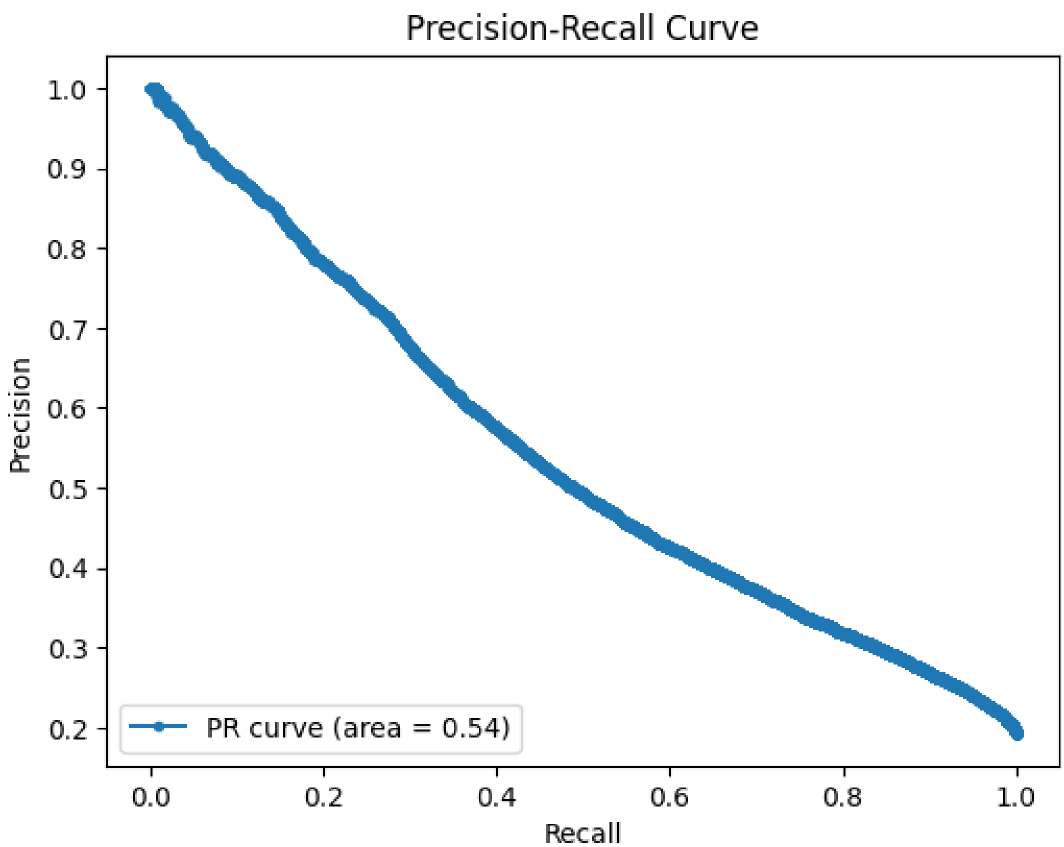
```
# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# 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()
```



Insights * An AU-ROC value of 0.79 suggests that, on average, the model has a good ability to distinguish between the positive and negative classes, but there is room for improvement.

[134]:

```
# 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()
```



Insights * From the values mentioned above it can be observed that the model is performing as expected and no further hypertuning can improve the preformance. * The low precision value for class 0 can be due to the imbalance of data for the same, if more real time data for class 0 can be provided, the model can be trained better and the performance might increase. * Also since the data consists of a lot of categorical columns a different ML model might prove better in predicting the outcome than Logistic Regression. * The model's precision value of 0.91 signifies that it accurately predicts the likelihood of loan repayment in 91% of cases. * The model's precision value of 0.38 for charged-off loans indicates that, among the instances predicted as charged off, only 38% were correctly classified, emphasizing a lower accuracy in predicting this specific class. * The model's sensitivity value of 0.73 for loan repayment signifies that it accurately identifies 73% of the instances where loans are repaid, demonstrating its ability to effectively capture a significant portion of the actual loan repayment cases. * The model's sensitivity value of 0.69 for charged-off loans signifies that it correctly identifies 69% of the actual charged-off instances, reflecting its ability to capture a substantial portion of the relevant cases for this class. * The features that heavily affected the models outcome are * grade - LoanTap assigned loan grade (Risk ratings by LoanTap) * pub_rec - Negative records on borrower's public credit profile. * From the analysis performed it can also be observed that the applicants for regions with pincodes('11650'm '86630' and '93700') have not made any loan repayment. It can be inferred that either * The data is missing w.r.t. loan repayment for these regions or * The applicants from regions with pincodes('11650'm '86630' and '93700') are highly unlikely to repay the loan granted by LoanTap. * LoanTap should carefully review the applicants belonging to above regions.