LoanTap_Logistic_Regression_BSC

December 23, 2024

#**Context:**

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.

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

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

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

Problem Statement:

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

#Know Your Data

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
#df = pd.read_csv('/kaggle/input/loantap-logisticregression/logistic_regression.
↪csv') #try
df=pd.read_csv('logistic_regression.csv')
```

[4]:
```
df.head()
```

[4]:
```
   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]
```

[5]:
```
df.shape
```

[5]:
```
(396030, 27)
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object
 5   sub_grade             396030 non-null  object
 6   emp_title             373103 non-null  object
```

```
 7   emp_length            377729 non-null  object
 8   home_ownership        396030 non-null  object
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object
 11  issue_d               396030 non-null  object
 12  loan_status           396030 non-null  object
 13  purpose               396030 non-null  object
 14  title                 394274 non-null  object
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

Data Dictionary:

1. loan_amnt: Amount borrower applied for.
2. term: Loan duration (36 or 60 months).
3. int_rate: Interest rate on loan.
4. installment: Monthly repayment amount.
5. grade: LoanTap assigned loan grade (Risk ratings by LoanTap.)
6. sub_grade: LoanTap assigned loan grade (Risk ratings by LoanTap.)
7. emp_title: Borrower's job title.
8. emp_length: Duration of borrower's employment (0-10 years).
9. home_ownership: Borrower's housing situation (own, rent, etc.).
10. annual_inc: Borrower's yearly income.
11. verification_status: Whether borrower's income was verified.
12. issue_d: Loan issuance month.
13. loan_status: Current status of the loan.
14. purpose: Borrower's reason for the loan.
15. title: The loan's title provided by the borrower.
16. dti (Debt-to-Income ratio): Monthly debt vs. monthly income ratio.
17. earliest_cr_line: Date of borrower's oldest credit account.
18. open_acc: Number of borrower's active credit lines.
19. pub_rec: Negative records on borrower's public credit profile.
20. revol_bal: Total credit balance.
21. revol_util: Usage percentage of 'revolving' accounts like credit cards.
22. total_acc: Total number of borrower's credit lines.
23. initial_list_status: Loan's first category ('W' or 'F').
24. application_type: Individual or joint application.

25. mort_acc: Number of borrower's mortgages.
26. pub_rec_bankruptcies: Bankruptcy records for borrower.
27. Address: Borrower's location.

Observations:

1. There are 396030 rows and 27 columns
2. Data contains some missing values
3. Data will require some preprocessing like handling null values, outliers, data types…etc which will be taken care in the following section

#Data Preprocessing

- Data Cleaning (Null Values / Duplicates / Outlier Treatment)
- Feature Engineering
- Data Type Conversion

[7]:
```
df1=df.copy()
```

[9]:
```
# Non-numeric columns
obj_cols = df1.select_dtypes(include='object').columns
obj_cols
```

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

[10]:
```
for _ in obj_cols:
    print()
    print(f'Total Unique Values in {_} column are :- {df1[_].nunique()}')
    print(f'Value counts in {_} column are :-\n {df1[_].
 ↪value_counts(normalize=True)}')
    print()
    print('-'*120)
```

```
Total Unique Values in term column are :- 2
Value counts in term column are :-
 term
36 months    0.762581
60 months    0.237419
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in grade column are :- 7
Value counts in grade column are :-
```

```
 grade
B    0.292953
C    0.267624
A    0.162076
D    0.160402
E    0.079509
F    0.029725
G    0.007712
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in sub_grade column are :- 35
Value counts in sub_grade column are :-
 sub_grade
B3    0.067306
B4    0.064644
C1    0.059748
C2    0.057016
B2    0.056801
B5    0.055766
C3    0.053584
C4    0.051208
B1    0.048436
A5    0.046779
C5    0.046067
D1    0.040383
A4    0.039868
D2    0.035227
D3    0.030864
D4    0.029435
A3    0.026705
A1    0.024566
D5    0.024493
A2    0.024157
E1    0.019991
E2    0.018764
E3    0.015673
E4    0.013537
E5    0.011545
F1    0.008929
F2    0.006984
F3    0.005772
F4    0.004512
F5    0.003528
G1    0.002672
G2    0.001904
```

```
G3    0.001394
G4    0.000944
G5    0.000798
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in emp_title column are :- 173105
Value counts in emp_title column are :-
 emp_title
Teacher                    0.011764
Manager                    0.011391
Registered Nurse           0.004974
RN                         0.004948
Supervisor                 0.004905
                             …
Postman                    0.000003
McCarthy & Holthus, LLC    0.000003
jp flooring                0.000003
Histology Technologist     0.000003
Gracon Services, Inc       0.000003
Name: proportion, Length: 173105, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in emp_length column are :- 11
Value counts in emp_length column are :-
 emp_length
10+ years    0.333681
2 years      0.094848
< 1 year     0.083989
3 years      0.083830
5 years      0.070143
1 year       0.068520
4 years      0.063411
6 years      0.055174
7 years      0.055116
8 years      0.050745
9 years      0.040542
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in home_ownership column are :- 6
Value counts in home_ownership column are :-
```

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

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in verification_status column are :- 3
Value counts in verification_status column are :-
 verification_status
Verified           0.352405
Source Verified    0.331755
Not Verified       0.315840
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in issue_d column are :- 115
Value counts in issue_d column are :-
 issue_d
Oct-2014    0.037487
Jul-2014    0.031838
Jan-2015    0.029556
Dec-2013    0.026811
Nov-2013    0.026503
              …
Jul-2007    0.000066
Sep-2008    0.000063
Nov-2007    0.000056
Sep-2007    0.000038
Jun-2007    0.000003
Name: proportion, Length: 115, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in loan_status column are :- 2
Value counts in loan_status column are :-
 loan_status
Fully Paid     0.803871
Charged Off    0.196129
Name: proportion, dtype: float64
```

```
--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in purpose column are :- 14
Value counts in purpose column are :-
 purpose
debt_consolidation    0.592145
credit_card           0.209628
home_improvement      0.060677
other                 0.053493
major_purchase        0.022195
small_business        0.014395
car                   0.011860
medical               0.010595
moving                0.007207
vacation              0.006191
house                 0.005558
wedding               0.004575
renewable_energy      0.000831
educational           0.000649
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in title column are :- 48816
Value counts in title column are :-
 title
Debt consolidation            0.386716
Credit card refinancing       0.130587
Home improvement              0.038714
Other                         0.032794
Debt Consolidation            0.029441
                                …
Graduation/Travel Expenses    0.000003
Daughter's Wedding Bill       0.000003
gotta move                    0.000003
creditcardrefi                0.000003
Toxic Debt Payoff             0.000003
Name: proportion, Length: 48816, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in earliest_cr_line column are :- 684
Value counts in earliest_cr_line column are :-
 earliest_cr_line
```

```
Oct-2000    0.007618
Aug-2000    0.007411
Oct-2001    0.007313
Aug-2001    0.007282
Nov-2000    0.006909
              …
Jul-1958    0.000003
Nov-1957    0.000003
Jan-1953    0.000003
Jul-1955    0.000003
Aug-1959    0.000003
Name: proportion, Length: 684, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in initial_list_status column are :- 2
Value counts in initial_list_status column are :-
 initial_list_status
f    0.601131
w    0.398869
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in application_type column are :- 3
Value counts in application_type column are :-
 application_type
INDIVIDUAL    0.998205
JOINT         0.001073
DIRECT_PAY    0.000722
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in address column are :- 393700
Value counts in address column are :-
 address
USCGC Smith\r\nFPO AE 70466                         0.000020
USS Johnson\r\nFPO AE 48052                         0.000020
USNS Johnson\r\nFPO AE 05113                        0.000020
USS Smith\r\nFPO AP 70466                           0.000020
USNS Johnson\r\nFPO AP 48052                        0.000018
                                                      …
455 Tricia Cove\r\nAustinbury, FL 00813             0.000003
7776 Flores Fall\r\nFernandezshire, UT 05113        0.000003
```

```
6577 Mia Harbors Apt. 171\r\nRobertshire, OK 22690    0.000003
8141 Cox Greens Suite 186\r\nMadisonstad, VT 05113    0.000003
787 Michelle Causeway\r\nBriannaton, AR 48052         0.000003
Name: proportion, Length: 393700, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------
```

[11]:

```
# Numeric columns
num_cols = df1.select_dtypes(include='number').columns
num_cols
```

[11]:

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

[12]:

```
for _ in num_cols:
    print()
    print(f'Total Unique Values in {_} column are :- {df1[_].nunique()}')
    print(f'Value counts in {_} column are :-\n {df1[_].
 ↪value_counts(normalize=True)}')
    print()
    print('-'*120)
```

```
Total Unique Values in loan_amnt column are :- 1397
Value counts in loan_amnt column are :-
 loan_amnt
10000.0    0.069863
12000.0    0.053950
15000.0    0.050256
20000.0    0.047898
35000.0    0.036805
             …
36225.0    0.000003
950.0      0.000003
37800.0    0.000003
30050.0    0.000003
725.0      0.000003
Name: proportion, Length: 1397, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in int_rate column are :- 566
Value counts in int_rate column are :-
 int_rate
```

```
10.99    0.031339
12.99    0.024321
15.61    0.023609
11.99    0.021670
8.90     0.020248
           …
14.28    0.000003
18.72    0.000003
18.36    0.000003
30.84    0.000003
24.59    0.000003
Name: proportion, Length: 566, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in installment column are :- 55706
Value counts in installment column are :-
 installment
327.34     0.002444
332.10     0.001997
491.01     0.001858
336.90     0.001732
392.81     0.001725
             …
364.37     0.000003
1015.29    0.000003
398.04     0.000003
544.94     0.000003
572.44     0.000003
Name: proportion, Length: 55706, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in annual_inc column are :- 27197
Value counts in annual_inc column are :-
 annual_inc
60000.00    0.038666
50000.00    0.033591
65000.00    0.028617
70000.00    0.026953
40000.00    0.026839
              …
72179.00    0.000003
50416.00    0.000003
46820.80    0.000003
10368.00    0.000003
```

```
31789.88    0.000003
Name: proportion, Length: 27197, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in dti column are :- 4262
Value counts in dti column are :-
 dti
0.00     0.000790
14.40    0.000783
19.20    0.000763
16.80    0.000760
18.00    0.000758
           …
59.18    0.000003
48.37    0.000003
45.71    0.000003
42.38    0.000003
55.53    0.000003
Name: proportion, Length: 4262, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in open_acc column are :- 61
Value counts in open_acc column are :-
 open_acc
9.0     0.092869
10.0    0.089491
8.0     0.088723
11.0    0.082557
7.0     0.079105
          …
55.0    0.000005
76.0    0.000005
58.0    0.000003
57.0    0.000003
90.0    0.000003
Name: proportion, Length: 61, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in pub_rec column are :- 20
Value counts in pub_rec column are :-
 pub_rec
0.0     0.854158
```

```
1.0     0.125594
2.0     0.013827
3.0     0.003841
4.0     0.001331
5.0     0.000598
6.0     0.000308
7.0     0.000141
8.0     0.000086
9.0     0.000030
10.0    0.000028
11.0    0.000020
13.0    0.000010
12.0    0.000010
19.0    0.000005
40.0    0.000003
17.0    0.000003
86.0    0.000003
24.0    0.000003
15.0    0.000003
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in revol_bal column are :- 55622
Value counts in revol_bal column are :-
 revol_bal
0.0         0.005373
5655.0      0.000104
6095.0      0.000096
7792.0      0.000096
3953.0      0.000093
              …
42573.0     0.000003
72966.0     0.000003
105342.0    0.000003
37076.0     0.000003
29244.0     0.000003
Name: proportion, Length: 55622, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in revol_util column are :- 1226
Value counts in revol_util column are :-
 revol_util
0.00      0.005592
53.00     0.001900
```

```
60.00     0.001867
61.00     0.001855
55.00     0.001845
            …
892.30    0.000003
110.10    0.000003
123.00    0.000003
49.63     0.000003
128.10    0.000003
Name: proportion, Length: 1226, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in total_acc column are :- 118
Value counts in total_acc column are :-
 total_acc
21.0     0.036058
22.0     0.036007
20.0     0.035927
23.0     0.035156
24.0     0.035043
           …
110.0    0.000003
129.0    0.000003
135.0    0.000003
104.0    0.000003
103.0    0.000003
Name: proportion, Length: 118, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in mort_acc column are :- 33
Value counts in mort_acc column are :-
 mort_acc
0.0     0.390182
1.0     0.168649
2.0     0.139428
3.0     0.106212
4.0     0.077846
5.0     0.050788
6.0     0.030899
7.0     0.016894
8.0     0.008712
9.0     0.004623
10.0    0.002415
11.0    0.001337
```

```
12.0    0.000737
13.0    0.000408
14.0    0.000299
15.0    0.000170
16.0    0.000103
17.0    0.000061
18.0    0.000050
19.0    0.000042
20.0    0.000036
24.0    0.000028
22.0    0.000020
21.0    0.000011
25.0    0.000011
27.0    0.000008
32.0    0.000006
31.0    0.000006
23.0    0.000006
26.0    0.000006
28.0    0.000003
30.0    0.000003
34.0    0.000003
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------

Total Unique Values in pub_rec_bankruptcies column are :- 9
Value counts in pub_rec_bankruptcies column are :-
 pub_rec_bankruptcies
0.0    0.885928
1.0    0.108194
2.0    0.004670
3.0    0.000887
4.0    0.000207
5.0    0.000081
6.0    0.000018
7.0    0.000010
8.0    0.000005
Name: proportion, dtype: float64

--------------------------------------------------------------------------------
----------------------------------------
```

Converting to Required Data Types

[13]:
```
#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}
df1['emp_length']=df1['emp_length'].replace(d)
```

```
<ipython-input-13-5403edff74f4>:5: FutureWarning: Downcasting behavior in
`replace` is deprecated and will be removed in a future version. To retain the
old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to
the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df1['emp_length']=df1['emp_length'].replace(d)
```

[14]:
```
# Convert earliest credit line & issue date to datetime
df1['earliest_cr_line'] = pd.to_datetime(df1['earliest_cr_line'])
df1['issue_d'] = pd.to_datetime(df1['issue_d'])
```

```
<ipython-input-14-dfb40b390eec>: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.
  df1['earliest_cr_line'] = pd.to_datetime(df1['earliest_cr_line'])
<ipython-input-14-dfb40b390eec>:3: 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.
  df1['issue_d'] = pd.to_datetime(df1['issue_d'])
```

[15]:
```
#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']

df1[cat_cols] = df1[cat_cols].astype('category')
```

[16]:
```
df1.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
```

Feature Engineering / Handling Missing Values

Creation of Flags- If value greater than 1.0 then 1 else 0. This can be done on:

1. Pub_rec
2. Mort_acc
3. Pub_rec_bankruptcies

[17]:
```
#Mean aggregation of mort_acc by total_acc to fill missing values

avg_mort = df1.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
```

[18]:
```
df1['mort_acc'] = df1.apply(lambda x: fill_mort(x['total_acc'],x['mort_acc']),␣
 ↪axis=1)
```

[19]:
```
def pub_rec(number):
    if number == 0.0:
        return 0
    else:
        return 1

def mort_acc(number):
    if number == 0.0:
        return 0
    elif number >= 1.0:
```

```
        return 1

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

[20]:
```
df1['pub_rec']=df1.pub_rec.apply(pub_rec)
df1['mort_acc']=df1.mort_acc.apply(mort_acc)
df1['pub_rec_bankruptcies']=df1.pub_rec_bankruptcies.apply(pub_rec_bankruptcies)
```

[21]:
```
df1['issue_d_year']=df1['issue_d'].dt.year
```

Deriving Zip Code and State from Address

[22]:
```
# Deriving zip code and state from address
df1[['state', 'zip_code']] = df1['address'].apply(lambda x: pd.Series([x[-8:
 ↪-6], x[-5:]]))
```

[23]:
```
#Drop address
df1.drop(["address"], axis = 1, inplace=True)
```

[24]:
```
df1['zip_code'].nunique()
```

[24]:
```
10
```

[25]:
```
df1['zip_code'] = df1['zip_code'].astype('category')
```

[26]:
```
#Filling missing values with 'Unknown' for object dtype
fill_values = {'title': 'Unknown', 'emp_title': 'Unknown'}
df1.fillna(value=fill_values, inplace=True)
```

[27]:
```
df1.isna().sum()
```

[27]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title                   0
emp_length              18301
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                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                    0
pub_rec_bankruptcies      535
issue_d_year                0
state                       0
zip_code                    0
dtype: int64
```

[28]:
```
df1.dropna(inplace=True)
```

[29]:
```
df1.shape
```

[29]:
```
(376929, 29)
```

Check for Duplicate Values

[30]:
```
df1.duplicated().any()
```

[30]:
```
False
```

No Duplicate Records Observed

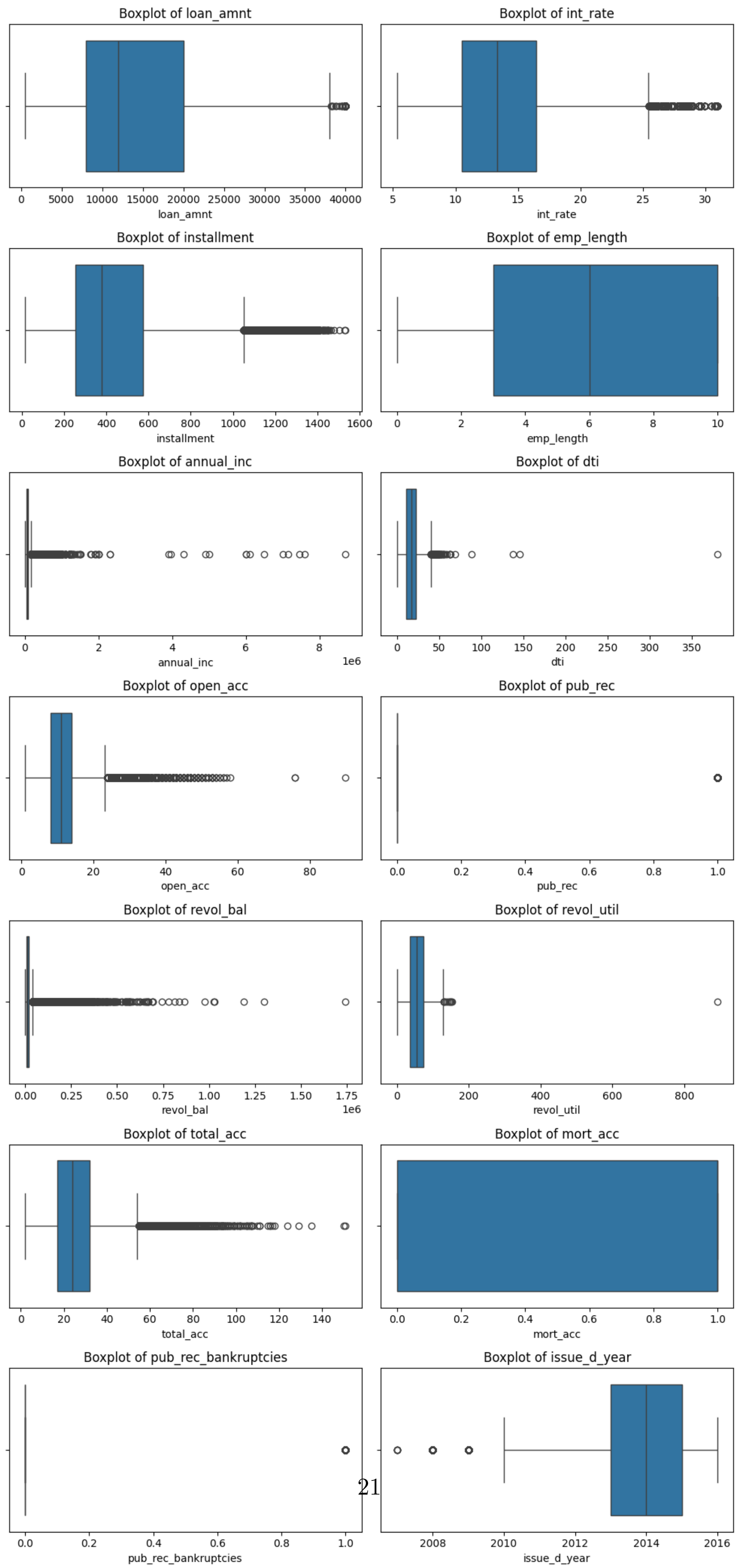
Outlier Treatment

[31]:
```
import seaborn as sns
```

[32]:
```
num_cols = df1.select_dtypes(include='number')
```

[33]:
```
fig = plt.figure(figsize=(10,21))
i=1
for col in num_cols:
  ax = plt.subplot(7,2,i)
  sns.boxplot(x=df1[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
Boxplot of issue_d_year
issue_d_year

[34]:
```
num_cols.columns
```

[34]:
```
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', 'issue_d_year'],
      dtype='object')
```

Removing columns 'pub_rec_bankruptcies','pub_rec','mort_acc' from outlier treatment since they are categorical in nature now

[35]:
```
newnum_cols=['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc']
```

[36]:
```
for col in newnum_cols:
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df1 = df1[(df1[col] >= lower) & (df1[col] <= upper)]
```

[37]:
```
df1.shape
```

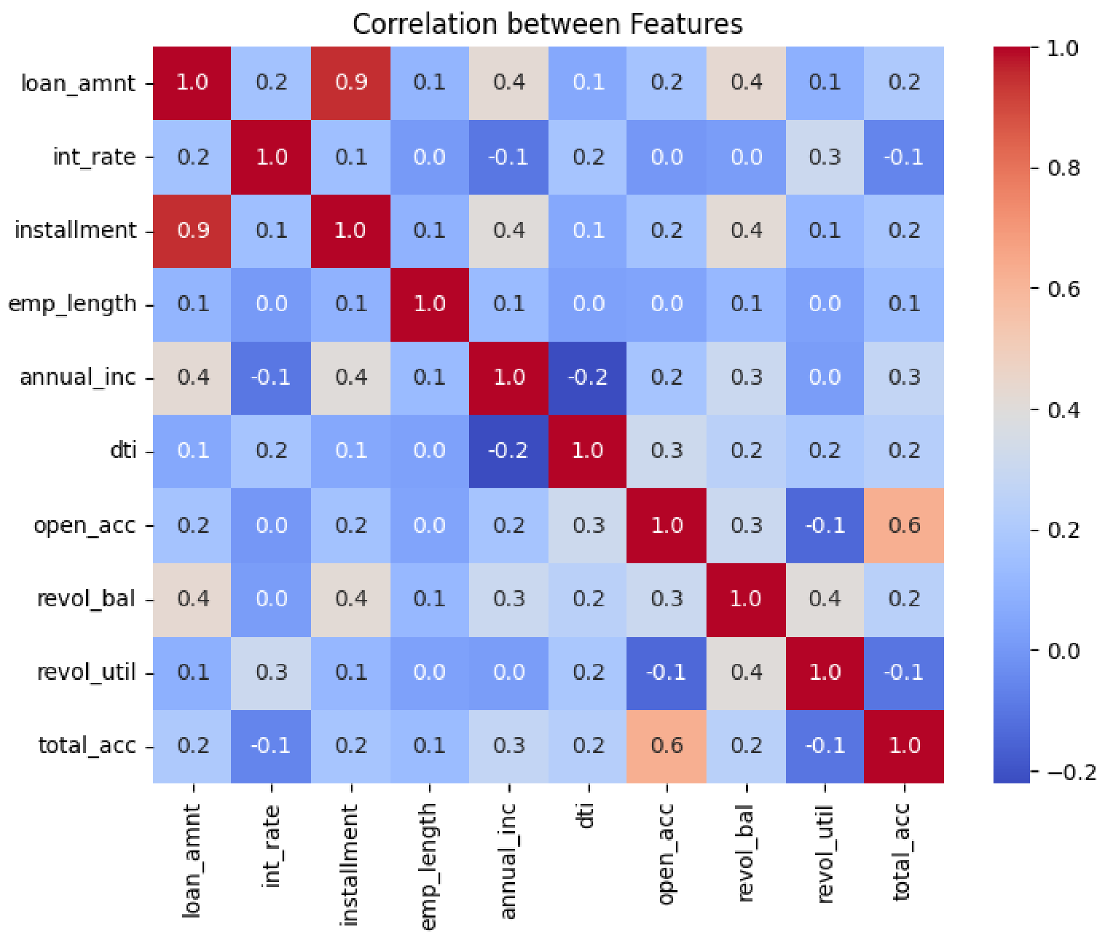
[37]:
```
(318371, 29)
```

Removed Outliers using IQR so that they do not distort model stability and accuracy

1 Exploratory Data Analysis

- Correlation
- Univariate
- Bivariate

[38]:
```
#Correlation between numerical features
plt.figure(figsize=(8,6))
sns.heatmap(df1[newnum_cols].corr(), annot=True, fmt=".1f",cmap='coolwarm')
plt.title('Correlation between Features')
plt.show()
```



Observations:

1. installment and loan_amnt are almost perfectly positive correlated. So one of these can be removed for model building
2. total_acc and open_acc are moderately positive correlated

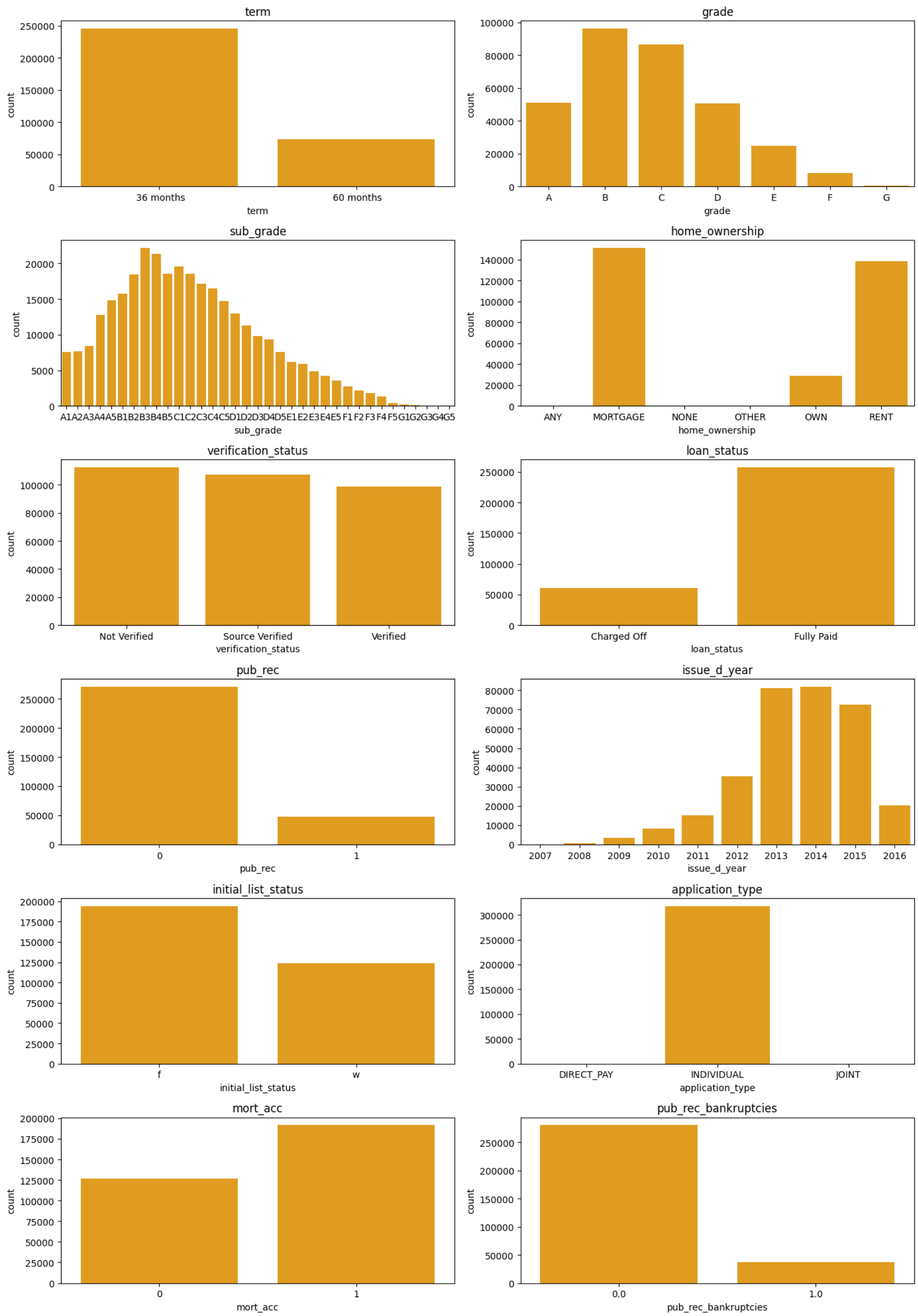
[39]:
```
#Drop installment
df1.drop(columns=['installment'], inplace=True)
```

Distribution of Variables

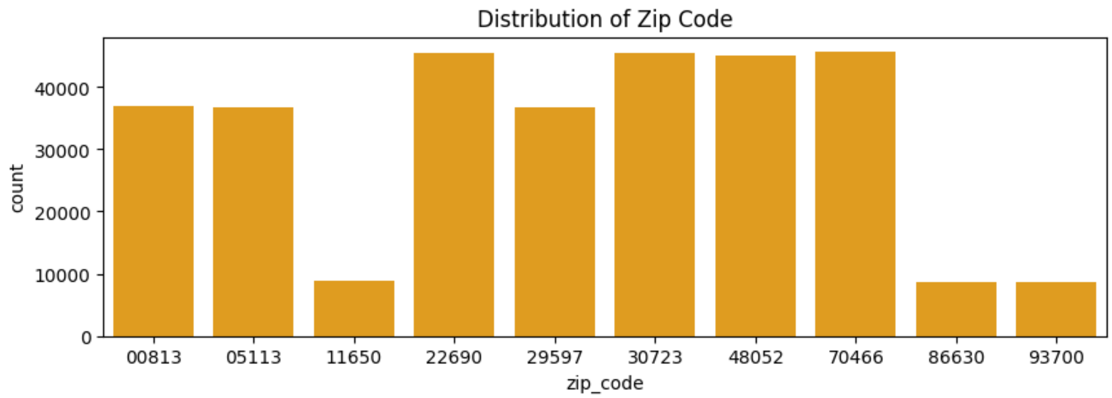
[40]:
```
newcat_cols = ['term', 'grade','sub_grade','home_ownership',
             'verification_status','loan_status','pub_rec','issue_d_year',
             'initial_list_status','application_type','mort_acc','pub_rec_bankruptcies']
```

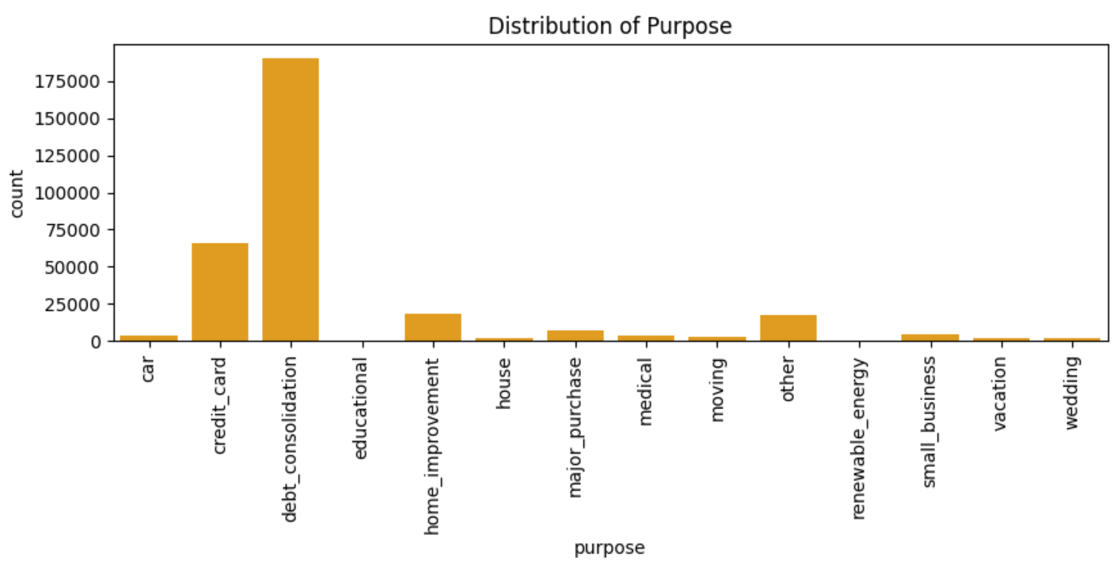
[41]:
```
plt.figure(figsize=(14,20))
i=1
for col in newcat_cols:
```

```
  ax=plt.subplot(6,2,i)
  sns.countplot(x=df1[col],color='orange')
  plt.title(f'{col}')
  i += 1

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


term
count
250000
200000
150000
100000
50000
0
36 months
60 months
term
grade
count
100000
80000
60000
40000
20000
0
A
B
C
D
E
F
G
grade
sub_grade
count
20000
15000
10000
5000
0
A1A2A3A4A5B1B2B3B4B5C1C2C3C4C5D1D2D3D4D5E1E2E3E4E5F1F2F3F4F5G1G2G3G4G5
sub_grade
home_ownership
count
140000
120000
100000
80000
60000
40000
20000
0
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership
verification_status
count
100000
80000
60000
40000
20000
0
Not Verified
Source Verified
Verified
verification_status
loan_status
count
250000
200000
150000
100000
50000
0
Charged Off
Fully Paid
loan_status
pub_rec
count
250000
200000
150000
100000
50000
0
0
1
pub_rec
issue_d_year
count
80000
70000
60000
50000
40000
30000
20000
10000
0
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
issue_d_year
initial_list_status
count
200000
175000
150000
125000
100000
75000
50000
25000
0
f
w
initial_list_status
application_type
count
300000
250000
200000
150000
100000
50000
0
DIRECT_PAY
INDIVIDUAL
JOINT
application_type
mort_acc
count
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
mort_acc
pub_rec_bankruptcies
count
250000
200000
150000
100000
50000
0
0.0
1.0
pub_rec_bankruptcies

[42]:
```
plt.figure(figsize=(10,3))
sns.countplot(x=df1['zip_code'],color='orange')
plt.title('Distribution of Zip Code')

plt.figure(figsize=(10,3))
sns.countplot(x=df1['purpose'],color='orange')
plt.xticks(rotation=90)
plt.title('Distribution of Purpose')

plt.show()
```





Observations:

- Approx. 80% of the loans are of 36 months duration
- Maximum Loans are from B grade followed by C,D,A

- Maximum Home Ownersip belong to MORTGAGE followed by RENT and OWN
- Fully Paid loans are almost 80% of the target variable loan_status
- Almost 90% of the applicants do not have derogatory Public Records
- Initial Listing Status of the loan is more in f category than w
- Almost 99% of the application types are individual
- Most of the applicants have got Mortage Account
- Almost 90% of the applicant have no Public Record Bankrupcies
- Almost 55% of the loans are taken against debt_consolidation followed by Credict card
- 2013 and 2014 were the years with maximum loans funding

Impact of Categorical Columns on Loan Status

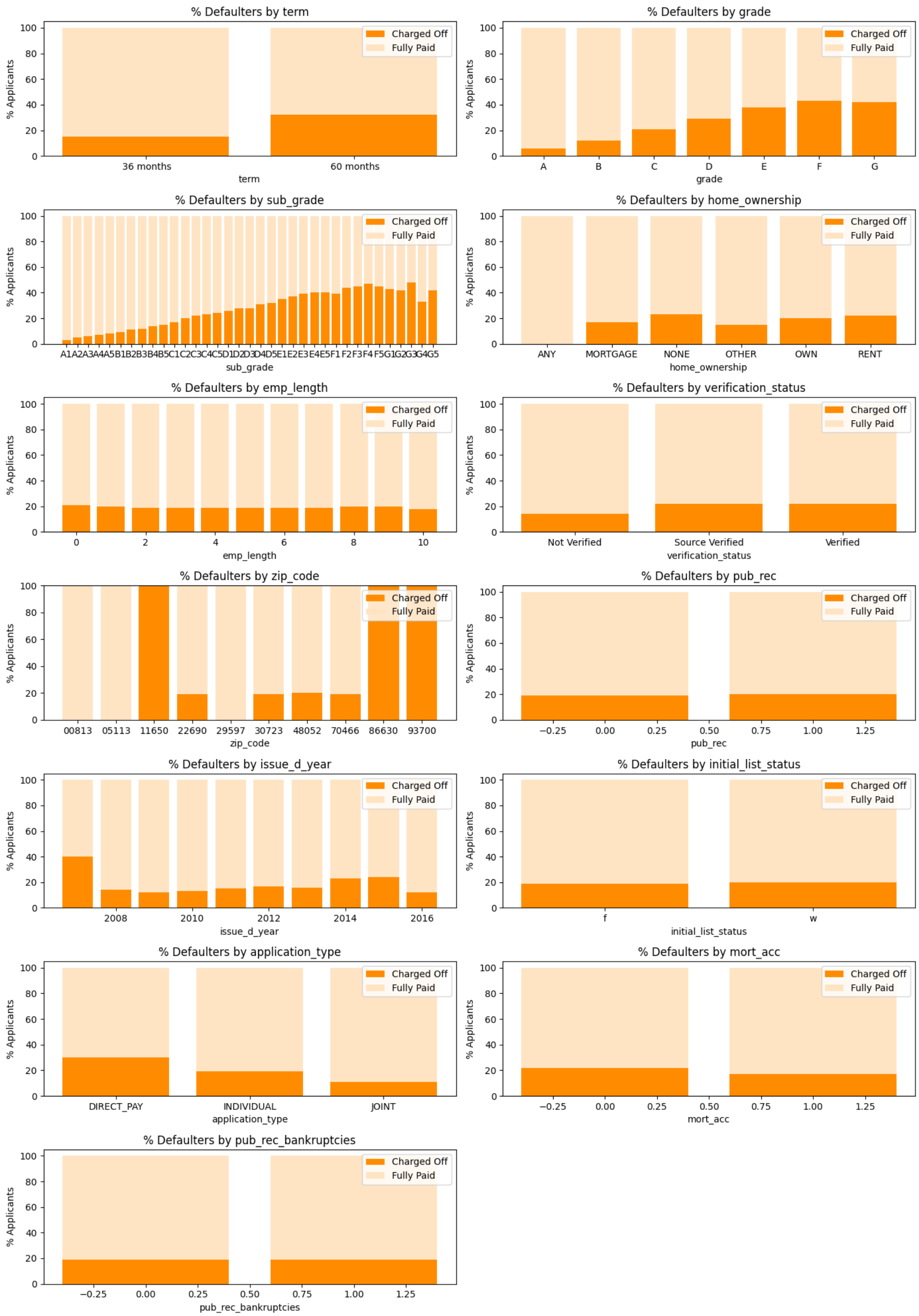
[43]:
```
newcat1_cols = ['term', 'grade','sub_grade','home_ownership','emp_length',
                'verification_status','zip_code','pub_rec','issue_d_year',
               ␣
 ↪'initial_list_status','application_type','mort_acc','pub_rec_bankruptcies']
```

[44]:
```
plt.figure(figsize=(14,20))
i=1
for col in newcat1_cols:
  ax=plt.subplot(7,2,i)

  data = df1.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='darkorange')
  plt.bar(data[col],data['Fully Paid'], color='bisque', 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()
```

```
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
```

```
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
```

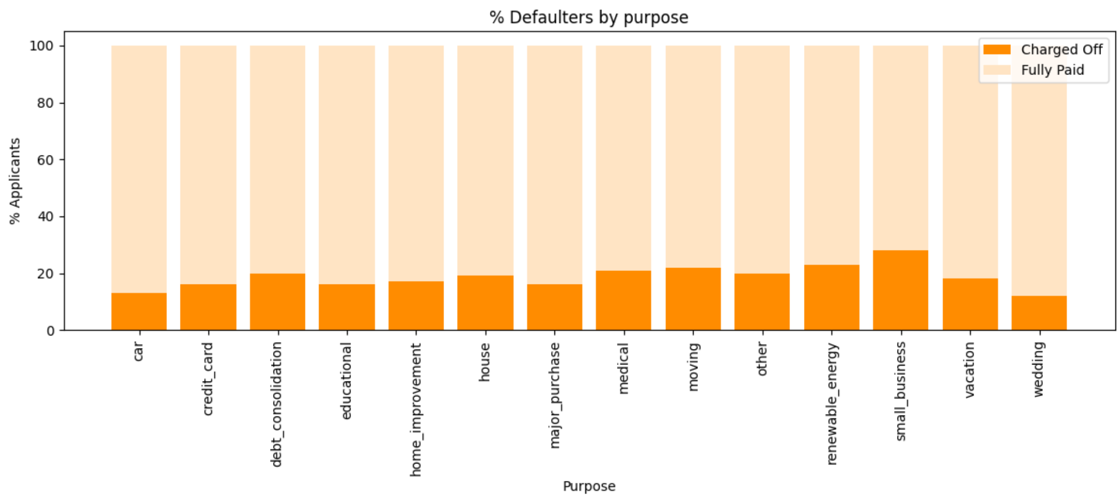
```
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
<ipython-input-44-d3d8d73e1fa8>:6: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  data = df1.pivot_table(index=col, columns='loan_status', aggfunc='count',
values='purpose')
```


% Defaulters by term
% Applicants
Charged Off
Fully Paid
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 emp_length
emp_length
% 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
pub_rec
% Defaulters by issue_d_year
issue_d_year
% Defaulters by initial_list_status
f
w
initial_list_status
% Defaulters by application_type
DIRECT_PAY
INDIVIDUAL
JOINT
application_type
% Defaulters by mort_acc
mort_acc
% Defaulters by pub_rec_bankruptcies
pub_rec_bankruptcies

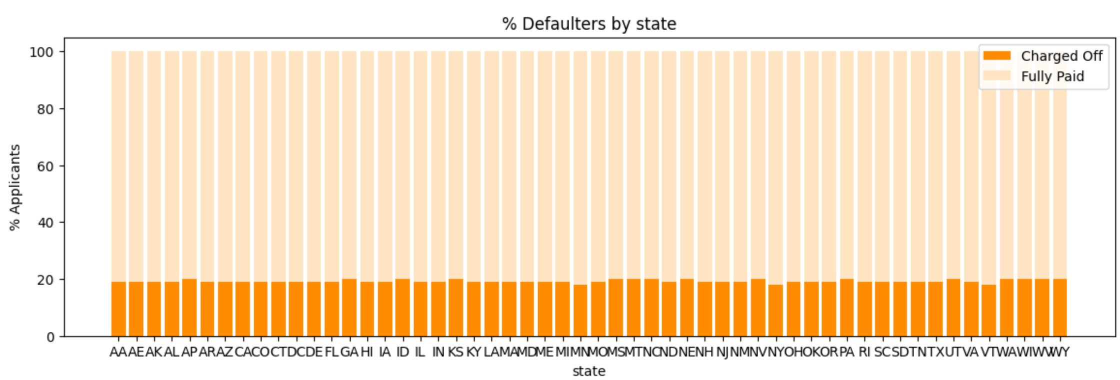
Impact of Purpose and State on Loan Status

[45]:
```python
purpose = df1.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='darkorange')
plt.bar(purpose['purpose'],purpose['Fully Paid'], color='bisque',␣
    ↪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 = df1.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='darkorange')
plt.bar(state['state'],state['Fully Paid'], color='bisque',␣
    ↪bottom=state['Charged Off'])
plt.xlabel('state')
plt.ylabel('% Applicants')
plt.title('% Defaulters by state')
plt.legend(['Charged Off','Fully Paid'])
plt.show()
```

```
<ipython-input-45-b56baed50c3e>:1: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  purpose = df1.pivot_table(index='purpose', columns='loan_status',
aggfunc='count', values='sub_grade')
```




```
<ipython-input-45-b56baed50c3e>:15: FutureWarning: The default value of
observed=False is deprecated and will change to observed=True in a future
version of pandas. Specify observed=False to silence this warning and retain the
current behavior
  state = df1.pivot_table(index='state', columns='loan_status', aggfunc='count',
values='sub_grade')
```



Observations:

- Percent share of default is much higher for long duration loans i.e 60 months
- Defaulters are highest for grade f and g and then decrease with grade. Sub-grade showing similar pattern
- Home Ownership: Charged-off % is high for None category followed by Rent, Own and Mortgage
- Surprisingly 100% defaulters observed for Zip codes 11650, 86630 and 93700. And zip codes with no defaulters at all are 00813, 05113, 29597
- pub_rec, pub_rec_bankruptcies, init_list_status and state have no impact
- In application type, Direct pay has maximum defaulters followed by individual and joint

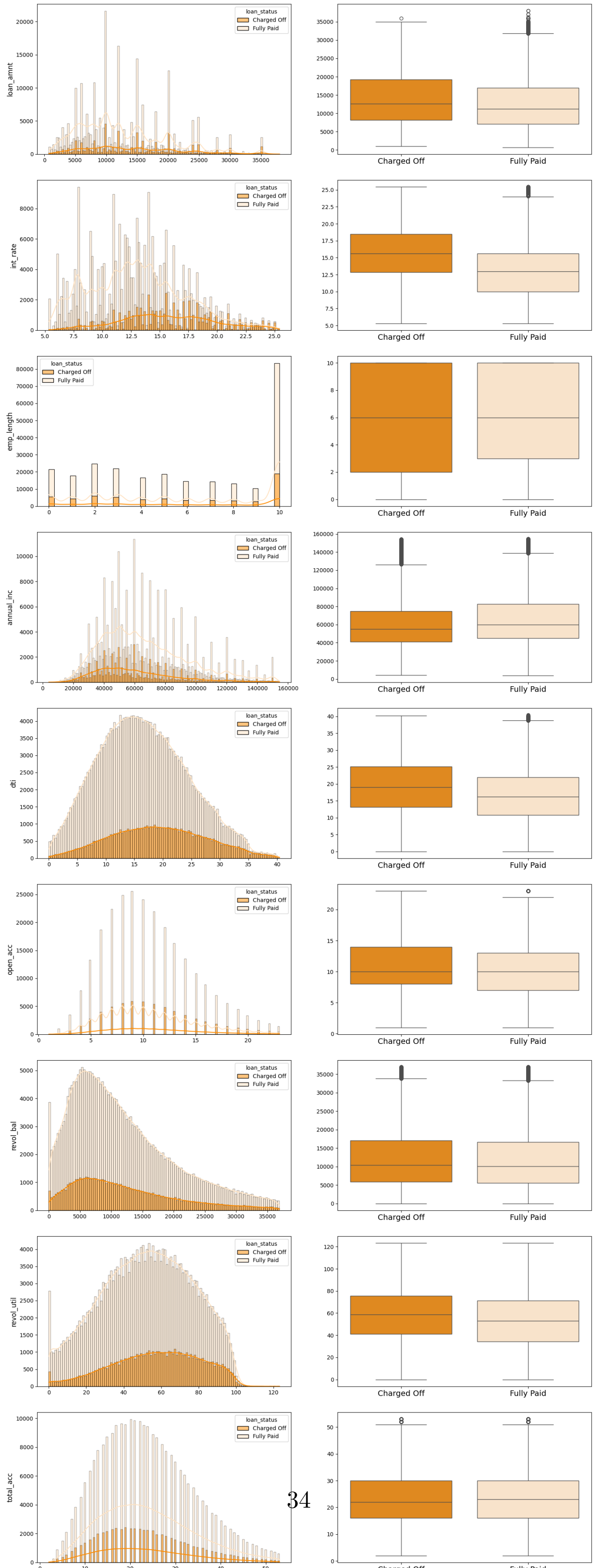
- Applicants with mort_acc 0 have higher charged off % than ones with mort_acc category 1
- Applicants with small_business have high default rate followed by renewable energy and others
- No significant impact of employment length on loan repayments
- 2007 is the year with maximum percent of defaulters followed by 2015 and 2014

Impact of Numerical Features on Loan Status

[46]:
```
newnum1_cols=['loan_amnt', 'int_rate', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc']
```

[47]:
```
import warnings
import matplotlib.colors as mcolors
```

[48]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
fig, ax = plt.subplots(9,2,figsize=(15,40))
i=0
color_dict = {'Fully Paid': mcolors.to_rgba('bisque', 0.5),
              'Charged Off': mcolors.to_rgba('darkorange', 1)}
for col in newnum1_cols:
    sns.boxplot(data=df1, y=col, x='loan_status', ax=ax[i,1],
               palette=('darkorange', 'bisque'))
    sns.histplot(data=df1, x=col, hue='loan_status', ax=ax[i, 0], legend=True,
                palette=color_dict, kde=True, fill=True)
    ax[i,0].set_ylabel(col, fontsize=12)
    ax[i,0].set_xlabel(' ')
    ax[i,1].set_xlabel(' ')
    ax[i,1].set_ylabel(' ')
    ax[i,1].xaxis.set_tick_params(labelsize=14)
    i += 1

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


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

Observations:

- Loan amount, int_rate, dti and open_acc show almost normal distribution
- annual_inc, revol_bal, total_acc are right skewed
- revol_util is left skewed
- Mean loan_amount,int_rate, dti, open_acc, revol_util is slightly higher for charged off
- Mean annual_inc is lower for charged off than fully paid

Basis above analysis, removing some features for further analysis and model building

[49]:
```
# Remove columns which do not have an impact on loan_status
df1.drop(columns=['initial_list_status','state',
                 'emp_title', 'title','earliest_cr_line',
                 'issue_d','sub_grade'], inplace=True)
# Removed issue_d since already extracted year from it and will be taken into
 ↪account
# Removed sub_grade, grade is indicative of same and is part of analysis
```

[50]:
```
df1.drop(columns=['pub_rec','pub_rec_bankruptcies'], inplace=True)
```

Part of Preprocessing

[51]:
```
#Encoding Target Variable
df1['loan_status']=df1['loan_status'].map({'Fully Paid': 0, 'Charged Off':1}).
 ↪astype(int)
```

[52]:
```
df1['term']=df1['term'].map({' 36 months': 36, ' 60 months':60}).astype(int)
```

[53]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 318371 entries, 0 to 396029
Data columns (total 19 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   loan_amnt            318371 non-null  float64
 1   term                 318371 non-null  int64
 2   int_rate             318371 non-null  float64
 3   grade                318371 non-null  category
 4   emp_length           318371 non-null  float64
 5   home_ownership       318371 non-null  category
 6   annual_inc           318371 non-null  float64
 7   verification_status  318371 non-null  category
 8   loan_status          318371 non-null  int64
 9   purpose              318371 non-null  category
 10  dti                  318371 non-null  float64
 11  open_acc             318371 non-null  float64
```

```
 12   revol_bal             318371 non-null  float64
 13   revol_util            318371 non-null  float64
 14   total_acc             318371 non-null  float64
 15   application_type      318371 non-null  category
 16   mort_acc              318371 non-null  int64
 17   issue_d_year          318371 non-null  int32
 18   zip_code              318371 non-null  category
dtypes: category(6), float64(9), int32(1), int64(3)
memory usage: 34.6 MB
```

#Data Preparation for Modeling

- Encoding
- SMOTE
- Scaling

[54]:
```python
from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

[55]:
```python
x = df1.drop(columns=['loan_status'])
x.reset_index(inplace=True, drop=True)
y = df1['loan_status']
y.reset_index(drop=True, inplace=True)
```

One Hot Encoding Categorical Columns

[61]:
```python
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
#importing OnehotEncoder
from sklearn.preprocessing import OneHotEncoder
cat_cols = x.select_dtypes('category').columns

encoder = OneHotEncoder(sparse_output=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()
```

[61]:
```
   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
```

```
   revol_bal  revol_util  total_acc  …  zip_code_00813  zip_code_05113  \
0    36369.0        41.8       25.0  …             0.0             0.0
1    20131.0        53.3       27.0  …             0.0             1.0
2    11987.0        92.2       26.0  …             0.0             1.0
3     5472.0        21.5       13.0  …             1.0             0.0
4    24584.0        69.8       43.0  …             0.0             0.0

   zip_code_11650  zip_code_22690  zip_code_29597  zip_code_30723  \
0             0.0             1.0             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             1.0             0.0             0.0             0.0

   zip_code_48052  zip_code_70466  zip_code_86630  zip_code_93700
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

[5 rows x 55 columns]
```

Train Test Split

[62]:
```
# Split into train, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(x, y, test_size=0.
↪2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
↪test_size=0.25, random_state=42)  # 0.25 * 0.8 = 0.2
```

Check Class Imbalance

[63]:
```
print(f"Before OverSampling, count of label 1: {sum(y_train == 1)}")
print(f"Before OverSampling, count of label 0: {sum(y_train == 0)}")
```

```
Before OverSampling, count of label 1: 36811
Before OverSampling, count of label 0: 154211
```

SMOTE:

(Synthetic Minority Over-sampling Technique) is often used to handle imbalanced datasets, especially when the target variable has significantly fewer instances of one class compared to the other. If our binary classification problem has an imbalanced target variable, applying SMOTE can help improve model performance by generating synthetic samples of the minority class.

[64]:
```
from imblearn.over_sampling import SMOTE
```

[65]:
```
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

[66]:
```
print(f"After OverSampling, count of label 1: {sum(y_train_resampled == 1)}")
print(f"After OverSampling, count of label 0: {sum(y_train_resampled == 0)}")
```

```
After OverSampling, count of label 1: 154211
After OverSampling, count of label 0: 154211
```

Scale Numerical Features

We will perform standard scaling on Numerical features and keep intact One hot encoded features and not perform scaling on them.

Scaling binary variables can make them harder to interpret. In many cases, the binary nature of these variables is crucial for understanding their meaning in the context of the data. Distorting this binary nature can lead to misinterpretations of the data

[67]:
```
numerical_columns=['loan_amnt', 'term', 'int_rate', 'emp_length', 'annual_inc',␣
 ↪'dti',
       'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'issue_d_year']
```

[68]:
```
scaler = StandardScaler()

# Fit the scaler on the resampled training numerical features
scaler.fit(X_train_resampled[numerical_columns])

# Scale the numerical features in the resampled training set
X_train_scaled_numeric = scaler.transform(X_train_resampled[numerical_columns])
# Scale the numerical features in the validation and test sets
X_val_scaled_numeric = scaler.transform(X_val[numerical_columns])
X_test_scaled_numeric = scaler.transform(X_test[numerical_columns])

# Convert scaled numerical features back to DataFrame to align indices
X_train_scaled_numeric_df = pd.DataFrame(X_train_scaled_numeric,␣
 ↪columns=numerical_columns, index=X_train_resampled.index)
X_val_scaled_numeric_df = pd.DataFrame(X_val_scaled_numeric,␣
 ↪columns=numerical_columns, index=X_val.index)
X_test_scaled_numeric_df = pd.DataFrame(X_test_scaled_numeric,␣
 ↪columns=numerical_columns, index=X_test.index)
```

[69]:
```
#Concatenate Scaled Numerical Features with One-Hot Encoded Features:
X_train_non_numeric = X_train_resampled.drop(columns=numerical_columns)
X_val_non_numeric = X_val.drop(columns=numerical_columns)
X_test_non_numeric = X_test.drop(columns=numerical_columns)
```

```
X_train_final = pd.concat([X_train_non_numeric.reset_index(drop=True),␣
 ↪X_train_scaled_numeric_df.reset_index(drop=True)], axis=1)
X_val_final = pd.concat([X_val_non_numeric.reset_index(drop=True),␣
 ↪X_val_scaled_numeric_df.reset_index(drop=True)], axis=1)
X_test_final = pd.concat([X_test_non_numeric.reset_index(drop=True),␣
 ↪X_test_scaled_numeric_df.reset_index(drop=True)], axis=1)
```

[70]:
```
X_train_final.head()
```

[70]:
```
   grade_A  grade_B  grade_C  grade_D  grade_E  grade_F  grade_G  \
0      1.0      0.0      0.0      0.0      0.0      0.0      0.0
1      0.0      0.0      1.0      0.0      0.0      0.0      0.0
2      0.0      0.0      0.0      0.0      1.0      0.0      0.0
3      0.0      0.0      0.0      1.0      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                      0.0                  0.0  …
2                 0.0                      0.0                  0.0  …
3                 0.0                      1.0                  0.0  …
4                 0.0                      0.0                  0.0  …

   int_rate  emp_length  annual_inc       dti  open_acc  revol_bal  \
0 -1.656132   -0.241089    0.424903 -2.328082  0.071498  -1.460465
1 -0.157262   -0.241089    0.789670 -1.250205  0.320259   0.827063
2  1.740010    0.636958    0.716717 -0.720356 -0.923544  -0.793701
3  0.646831    1.222323    0.315473 -0.129472  0.569019   0.758748
4  0.476781    1.222323    1.409774  1.187357  0.569019   2.444935

   revol_util  total_acc  mort_acc  issue_d_year
0   -2.411266   1.305726  1.021892     -2.567030
1    1.105705   0.679894  1.021892     -0.359957
2    0.996346   0.992810 -0.978577     -0.359957
3   -0.661529  -0.363160  1.021892      0.375734
4    1.691866   0.158367 -0.978577     -1.095648

[5 rows x 55 columns]
```

[71]:
```
X_train_final.shape
```

[71]:
```
(308422, 55)
```

[72]:
```
X_test_final.shape
```

[72]:
```
(63675, 55)
```

[73]:
```
X_val_final.shape
```

[73]: (63674, 55)

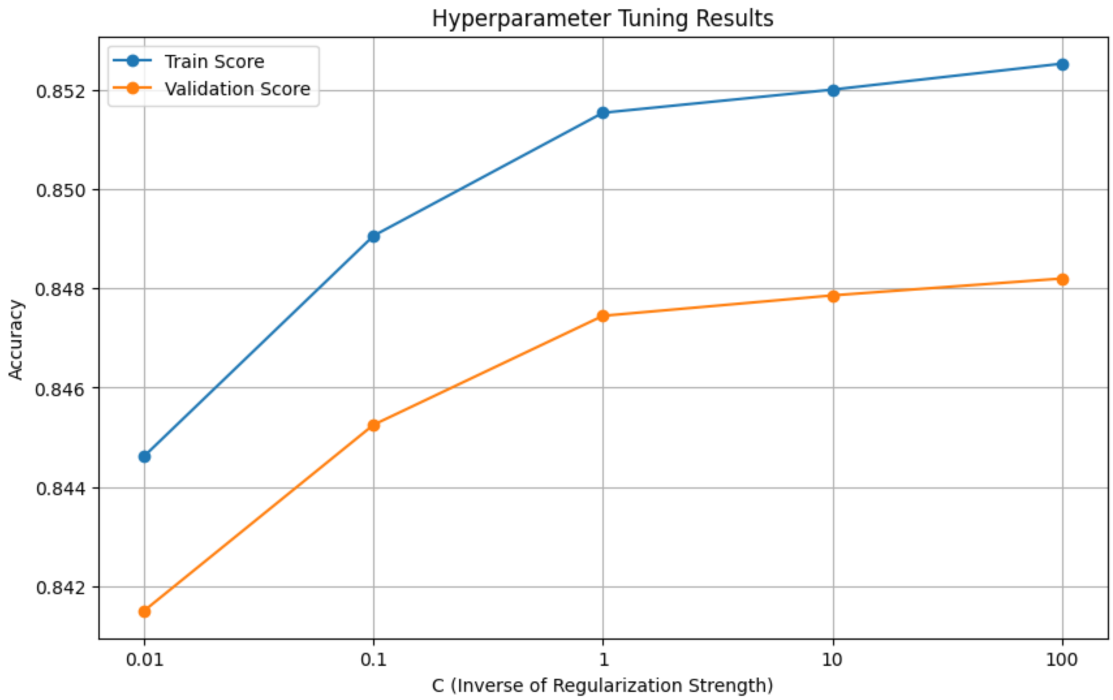
#Logistic Regression Model

- Build the Model
- Tune the Model
- Hyperparameter grid for C (inverse of regularization strength)
- Use GridSearchCV to find the best hyperparameters

[74]:
```
from sklearn.model_selection import GridSearchCV
```

[75]:
```
import warnings
warnings.filterwarnings("ignore")
```

[76]:
```
model = LogisticRegression(max_iter=1000)

# Define the hyperparameter grid for C (inverse of regularization strength)
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(model, param_grid, cv=5,␣
↪return_train_score=True,n_jobs=-1)
grid_search.fit(X_train_final, y_train_resampled)

# Extract mean validation scores for each value of C
results = pd.DataFrame(grid_search.cv_results_)
mean_train_scores = results['mean_train_score']
mean_val_scores = results['mean_test_score']
params = [str(param) for param in param_grid['C']]

plt.figure(figsize=(10, 6))
plt.plot(params, mean_train_scores, marker='o', label='Train Score')
plt.plot(params, mean_val_scores, marker='o', label='Validation Score')
plt.xlabel('C (Inverse of Regularization Strength)')
plt.ylabel('Accuracy')
plt.title('Hyperparameter Tuning Results')
plt.legend()
plt.grid(True)
plt.show()
```



In the above plot it is clearly observed that the accuracy of the model is highest with Hyperparameter C=100.

If we try to tune it further with values w.r.t 100 we can increase accuracy of the model further

[77]:
```
# Get the best model
best_model = grid_search.best_estimator_

# Evaluate on the validation set (Optional, just for reference)
val_score = best_model.score(X_val_final, y_val)
print(f'Validation Score: {val_score}')
```

```
Validation Score: 0.8448346263781135
```

[78]:
```
#Evaluate on train set
train_score = best_model.score(X_train_final, y_train_resampled)
print(f'Train Score: {train_score}')
```

```
Train Score: 0.8521019901304058
```

[79]:
```
# Evaluate on the test set
test_score = best_model.score(X_test_final, y_test)
print(f'Test Score: {test_score}')
```

```
Test Score: 0.8431723596387908
```

Observations:

- The training score is the highest, which is expected since the model is trained on this data.
- The validation score is slightly lower than the training score, which is also expected but close, indicating good generalization.
- The test score is slightly lower than both the training and validation scores but still close, indicating that the model generalizes reasonably well to unseen data.

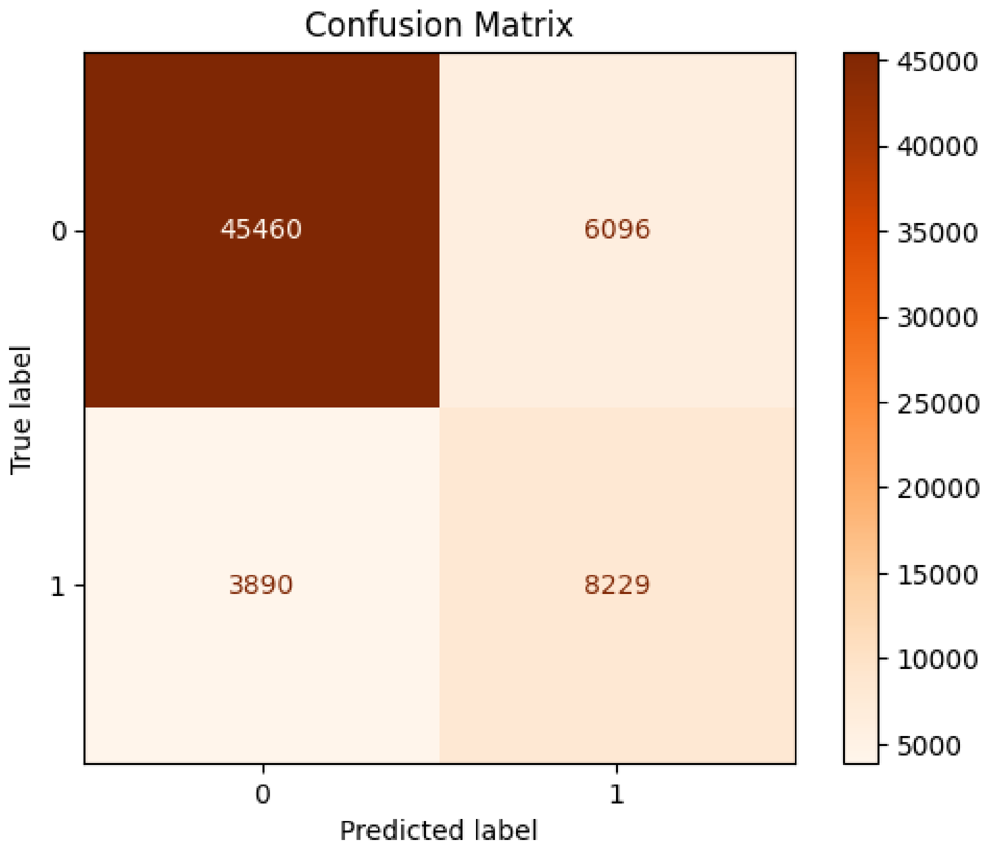
#Confusion Matrix

[80]:
```python
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)
```

[81]:
```python
# Make predictions on the test set
y_pred = best_model.predict(X_test_final)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[45460  6096]
 [ 3890  8229]]
```

[82]:
```python
disp = ConfusionMatrixDisplay(conf_matrix)
cmap = plt.cm.Oranges
disp.plot(cmap=cmap)
plt.title('Confusion Matrix')
plt.show()
```



[83]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90     51556
           1       0.57      0.68      0.62     12119

    accuracy                           0.84     63675
   macro avg       0.75      0.78      0.76     63675
weighted avg       0.86      0.84      0.85     63675
```

Observations:

Precision: Precision is the ratio of true positive predictions to the total number of positive predictions made by the model. In this context:

- Precision for class 0: 0.92 means that out of all instances predicted as class 0, 92% of them were actually class 0.
- Precision for class 1: 0.58 means that out of all instances predicted as class 1, only 58% of them were actually class 1.

Recall (Sensitivity): Recall is the ratio of true positive predictions to the total number of actual positive instances in the data. In this context:

- Recall for class 0: 0.88 means that the model correctly identified 88% of all actual class 0 instances.
- Recall for class 1: 0.68 means that the model correctly identified 68% of all actual class 1 instances.

F1-score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. In this context:

- F1-score for class 0: 0.90 is the harmonic mean of precision and recall for class 0.
- F1-score for class 1: 0.62 is the harmonic mean of precision and recall for class 1.

#Trade Off Analysis

The underwriting process for personal loans at LoanTap involves critical trade-offs between detecting genuine defaulters and avoiding false positives. Here are the key points to consider:

False Positives vs. False Negatives:

False Positives: Approving a loan for a potentially risky borrower. This could lead to non-performing assets (NPAs), which increase financial risk and loss.

False Negatives: Denying a loan to a creditworthy borrower. This results in lost revenue opportunities and a potential decrease in customer satisfaction.

#Interpreting Model Coefficients

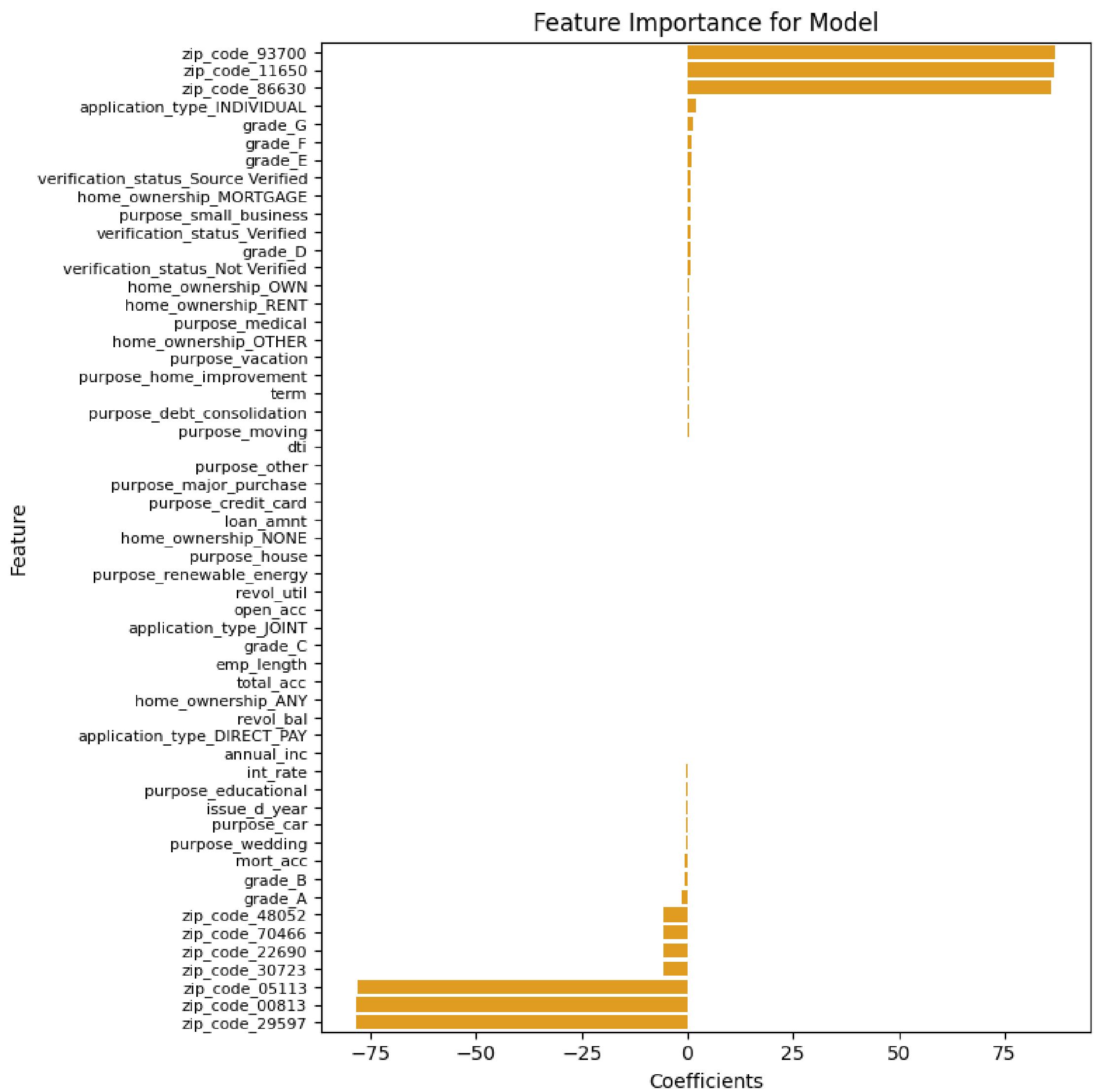
[84]:
```
# Extract coefficients and map them to feature names
coefficients = best_model.coef_[0]
feature_names = X_train_final.columns

# Display coefficients with feature names
coefficients_df = pd.DataFrame({'Feature': feature_names, 'Coefficient':␣
↪coefficients})
coefficients_df = coefficients_df.sort_values(by='Coefficient', ascending=False)
print(coefficients_df)
```

```
                                Feature  Coefficient
42                       zip_code_93700    87.043267
35                       zip_code_11650    86.530461
41                       zip_code_86630    85.898395
31           application_type_INDIVIDUAL     2.003869
6                               grade_G     1.207380
5                               grade_F     1.147732
4                               grade_E     0.914449
14  verification_status_Source Verified     0.820417
8                 home_ownership_MORTGAGE     0.711815
27               purpose_small_business     0.697324
15          verification_status_Verified     0.621075
3                               grade_D     0.551910
```

```
13     verification_status_Not Verified   0.531540
11                  home_ownership_OWN   0.472217
12                 home_ownership_RENT   0.369711
23                     purpose_medical   0.359346
10                home_ownership_OTHER   0.318524
28                    purpose_vacation   0.268989
20            purpose_home_improvement   0.253520
44                                term   0.218905
18              purpose_debt_consolidation   0.210964
24                      purpose_moving   0.207296
48                                 dti   0.189711
25                       purpose_other   0.159045
22              purpose_major_purchase   0.153082
17                 purpose_credit_card   0.144796
9                  home_ownership_NONE   0.120537
43                           loan_amnt   0.116101
21                       purpose_house   0.107677
26            purpose_renewable_energy   0.104508
49                            open_acc   0.092989
51                          revol_util   0.085469
32              application_type_JOINT   0.072153
2                              grade_C   0.049825
46                          emp_length   0.016115
52                           total_acc   0.008539
7                   home_ownership_ANY  -0.019771
50                           revol_bal  -0.068519
30         application_type_DIRECT_PAY  -0.102989
47                          annual_inc  -0.104358
45                            int_rate  -0.163437
19                 purpose_educational  -0.163780
54                        issue_d_year  -0.185444
16                         purpose_car  -0.202012
29                     purpose_wedding  -0.327724
53                            mort_acc  -0.574146
1                              grade_B  -0.588143
0                              grade_A  -1.310121
39                      zip_code_48052  -5.669659
40                      zip_code_70466  -5.706041
36                      zip_code_22690  -5.711699
38                      zip_code_30723  -5.729463
34                      zip_code_05113 -77.977405
33                      zip_code_00813 -78.325294
37                      zip_code_29597 -78.379531
```

[85]:
```
feature_imp = pd.DataFrame({'Columns':X_train_final.columns, 'Coefficients':
 ↪best_model.coef_[0]}).round(2).sort_values('Coefficients', ascending=False)
```

```python
plt.figure(figsize=(8,8))
sns.barplot(y = feature_imp['Columns'],
            x =  feature_imp['Coefficients'],color='orange')
plt.title("Feature Importance for Model")
plt.yticks(fontsize=8)
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



Observation:

- Zip codes- 11650, 93700, 86630 signify strong positive relationship with the Loan Status
- Whereas zip codes 29597,00813,05113 show strong negative relatioship with target variable
- It shows that features such as emp_length, total_acc, revol_bal, annual_inc, int_rate, is-

sue_d_year show no contribution at all. These features should have been dropped for analysis

#ROC Curve & AUC

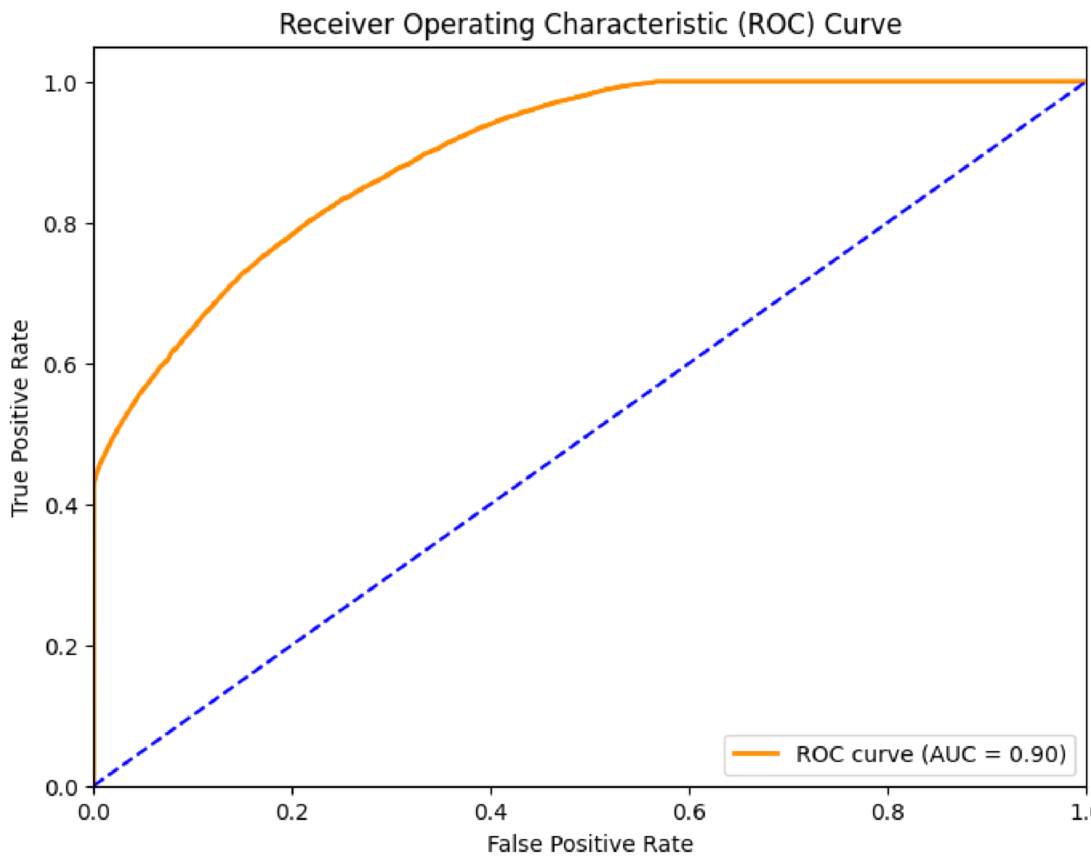
The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model. It helps evaluate and compare different models by illustrating the trade-off between the true positive rate (TPR) and false positive rate (FPR) at various classification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

The 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.

[86]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

[87]:
```
# Make predictions on the test set
y_pred_proba = best_model.predict_proba(X_test_final)[:, 1]

# Compute ROC curve and ROC-AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' %␣
↪roc_auc)
plt.plot([0, 1], [0, 1], color='blue', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



- AUC of 0.90 signifies that the model is able to discriminate well between the positive and the negative 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.

#Precision Recall Curve

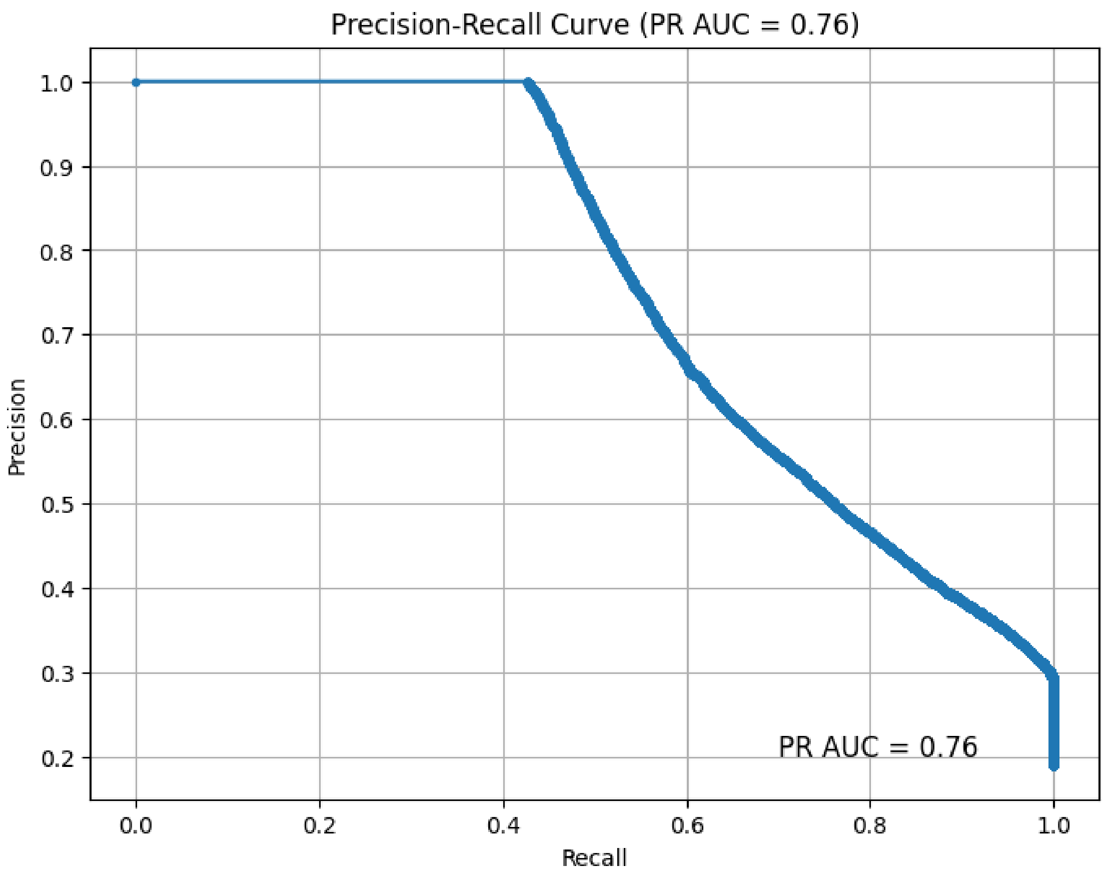
The Precision-Recall (PR) curve is another graphical representation commonly used to evaluate the performance of a binary classification model. It provides insights into the trade-off between precision and recall at various classification thresholds.

[88]:
```
from sklearn.metrics import precision_recall_curve
```

[89]:
```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
```

[90]:
```
pr_auc = auc(recall, precision)

# Plot the precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
```

```python
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (PR AUC = {:.2f})'.format(pr_auc))
plt.grid(True)

# Annotate the PR AUC value on the plot
plt.text(0.7, 0.2, 'PR AUC = {:.2f}'.format(pr_auc), fontsize=12)

plt.show()
```



- Interpretation of PR AUC: A PR AUC value of 0.76 indicates the area under the precision-recall curve. It represents the integral of precision-recall pairs across all possible decision thresholds. Essentially, it measures how well our classifier ranks positive samples with higher confidence scores compared to negative samples across all possible thresholds.
- A higher PR AUC value signifies better performance of the classifier in terms of balancing precision and recall.

#Insights

- Business Insights
- Questionnaire

- Approx. 80% of the loans are of 36 months duration
- Maximum Loans are from B grade followed by C,D,A
- Maximum Home Ownersip belong to MORTGAGE followed by RENT and OWN
- Fully Paid loans are almost 80% of the target variable loan_status
- Almost 90% of the applicants do not have derogatory Public Records
- Initial Listing Status of the loan is more in f category than w
- Almost 99% of the application types are individual
- Most of the applicants have got Mortage Account
- Almost 90% of the applicant have no Public Record Bankrupcies
- Almost 55% of the loans are taken against debt_consolidation followed by Credict card
- 2013 and 2014 were the years with maximum loans funding
- Percent share of default is much higher for long duration loans i.e 60 months
- Defaulters are highest for grade f and g and then decrease with grade. Sub-grade showing similar pattern
- Home Ownership: Charged-off % is high for None category followed by Rent, Own and Mortgage
- Surprisingly 100% defaulters observed for Zip codes 11650, 86630 and 93700. And zip codes with no defaulters at all are 00813, 05113, 29597
- pub_rec, pub_rec_bankruptcies, init_list_status and state have no impact
- In application type, Direct pay has maximum defaulters followed by individual and joint
- Applicants with mort_acc 0 have higher charged off % than ones with mort_acc category 1
- Applicants with small_business have high default rate followed by renewable energy and others
- No significant impact of employment length on loan repayments
- 2007 is the year with maximum percent of defaulters followed by 2015 and 2014
- Mean loan_amount,int_rate, dti, open_acc, revol_util is slightly higher for charged off
- Mean annual_inc is lower for charged off than fully paid
- The test score is slightly lower than both the training and validation scores but still close, indicating that the model generalizes reasonably well to unseen data.
- Precision for class 0: 0.92 means that out of all instances predicted as class 0, 92% of them were actually class 0.
- Precision for class 1: 0.58 means that out of all instances predicted as class 1, only 58% of them were actually class 1.
- Recall for class 0: 0.88 means that the model correctly identified 88% of all actual class 0 instances.
- Recall for class 1: 0.68 means that the model correctly identified 68% of all actual class 1 instances.
- F1-score for class 0: 0.90 is the harmonic mean of precision and recall for class 0.
- F1-score for class 1: 0.62 is the harmonic mean of precision and recall for class 1.
- Zip codes- 11650, 93700, 86630 signify strong positive relationship with the Loan Status
- Whereas zip codes 29597,00813,05113 show strong negative relatioship with target variable
- It shows that features such as emp_length, total_acc, revol_bal, annual_inc, int_rate, issue_d_year show no contribution at all.
- ROC Curve (AUC = 0.90) is observed
- Interpretation of PR AUC: A PR AUC value of 0.76 indicates the area under the precision-recall curve. It represents the integral of precision-recall pairs across all possible decision thresholds.

Questionnaire

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

80.38% of customers have fully paid up their loan

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

They are **close to perfect positive correlation** with value close 1 i.e 0.9

3. The majority of people have home ownership as **Mortgage**.
4. People with grades 'A' are more likely to fully pay their loan. **True**
5. Name the top 2 afforded job titles. **Teacher and Manager**
6. Thinking from a bank's perspective, which metric should our primary focus be on..

- ROC AUC
- Precision
- Recall
- F1 Score

  Recall: It measures the ability of the model to correctly identify all actual defaulters. High recall ensures that the bank catches as many risky borrowers as possible, thereby minimizing the number of approved loans that may default.

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

Financial Losses:

False Positives (Low Precision): Approving loans to individuals who later default can result in financial losses for the bank. These non-performing assets (NPAs) not only reduce profitability but also tie up capital that could have been invested elsewhere.

False Negatives (Low Recall): Rejecting creditworthy applicants due to overly conservative risk assessment can lead to missed revenue opportunities. The bank loses out on potential interest income and customer relationships.

Reputation Damage:

False Positives: Approving loans to individuals who subsequently default can damage the bank's reputation. It may erode trust among customers and investors, affecting brand perception and market credibility.

False Negatives: Rejecting creditworthy applicants unfairly can lead to dissatisfaction among customers. Negative word-of-mouth, social media backlash, and complaints to regulatory authorities can tarnish the bank's reputation.

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

Zip Code followed by Application Type and Verification status

9. Will the results be affected by geographical location?

Yes, Zip code as part of geographical location highly affected the results

#Recommendations & Feedback

Risk Mitigation:

- **Segment-Based Strategy**: Focus on higher grades (A, B, C) for initial rollouts while continuously monitoring performance. As the model proves effective, gradually extend to lower grades (D, E, F, G) with cautious parameters.
- **Loan Caps and Conditional Approvals**: Implement loan caps for high-risk segments and conditional approvals where additional guarantees or higher interest rates are applied.
- **Geographical Risk Assessment**: Given the strong relationship between certain zip codes and default rates, incorporate geographical risk factors into the model. Focus on high-risk zip codes with stricter criteria.

Enhancing Loan Approval Process:

- **Verification Process**: Strengthen the verification process for critical features like income, employment status, and home ownership to reduce misinformation.
- **Real-Time Monitoring**: Implement real-time credit monitoring for borrowers to identify early signs of financial distress and intervene before defaults occur.

Feedback Loop

1. **Continuous Monitoring**:

Performance Metrics: Continuously track key performance metrics such as precision, recall, F1-score, and AUC-ROC to evaluate model effectiveness.

Regular Audits: Conduct periodic audits of approved and denied loans to assess the model's decisions against actual outcomes.

2. **Iterative Improvements**:

Model Retraining: Regularly retrain the model with new data to capture changes in borrower behavior and economic conditions.

User Feedback: Incorporate feedback from loan officers and customers to identify areas of improvement in the model and process.

3. **Dynamic Risk Adjustments**:

Economic Indicators: Monitor macroeconomic indicators such as unemployment rates and economic growth to adjust lending criteria dynamically.

Anomaly Detection: Use anomaly detection techniques to identify and investigate unusual patterns in loan applications and repayments.

[ ]:

```

```