💷💳LoanTap - ML CaseStudy💳💷

Logistic Regression

Analysed by : **SUCHI**

💳Introduction:

Loantap is a leading financial technology company based in India, specializing in providing flexible and innovative loan products to individuals and businesses. With a focus on customer-centric solutions, Loantap leverages technology to offer hassle-free borrowing experiences, including personal loans, salary advances, and flexible EMI options. Their commitment to transparency, speed, and convenience has established them as a trusted partner for borrowers seeking efficient financial solutions.

- LoanTap is at the forefront of offering tailored financial solutions to millennials.
- Their innovative approach seeks to harness data science for refining their credit underwriting process.
- The focus here is the Personal Loan segment. A deep dive into the dataset can reveal patterns in borrower behavior and creditworthiness.
- Analyzing this dataset can provide crucial insights into the financial behaviors, spending habits, and potential risk associated with each borrower.
- The insights gained can optimize loan disbursal, balancing customer outreach with risk management.

🔹 Our Task:

- > As a data scientist at LoanTap, you are tasked with analyzing the dataset to determine the creditworthiness of potential borrowers. Your ultimate objective is to build a logistic regression model, evaluate its performance, and provide actionable insights for the underwriting process.

📃 Features of the dataset:

- Column Profiling:

| Feature | Description | |:--------|:------------| | loan_amnt | The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value| | term | The number of payments on the loan. Values are in months and can be either 36 or 60| | int_rate | Interest Rate on the loan| | installment | The monthly payment owed by the borrower if the loan originates| | grade | LoanTap assigned loan grade| | sub_grade | LoanTap assigned loan subgrade| | emp_title |The job title supplied by the Borrower when applying for the loan| | emp_length | Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years| | home_ownership | The home ownership status provided by the borrower during registration or obtained from the credit report| | annual_inc | The self-reported annual income provided by the borrower during registration| | verification_status | Indicates if income was verified by LoanTap, not verified, or if the income source was verified| | issue_d | The month which the loan was funded| | loan_status | Current status of the loan - Target Variable| | purpose | A category provided by the borrower for the loan request| | title | The loan title provided by the borrower| | dti | A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower’s self-reported monthly income| | earliest_cr_line |The month the borrower's earliest reported credit line was opened| | open_acc | The number of open credit lines in the borrower's credit file| | pub_rec | Number of derogatory public records| | revol_bal | Total credit revolving balance| | revol_util | Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit| | total_acc | The total number of credit lines currently in the borrower's credit file| | initial_list_status | The initial listing status of the loan| Possible values are – W, F| | application_type | Indicates whether the loan is an individual application or a joint application with two co-borrowers| | mort_acc | Number of mortgage accounts| | pub_rec_bankruptcies | Number of public record bankruptcies| | Address| Address of the individual|

🕵️ **Exploratory Data Analysis**

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

```
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import ttest_ind,chi2_contingency

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc, precision_recall_curve, average_precision
    ConfusionMatrixDisplay, RocCurveDisplay,f1_score,recall_score,precision_
)

from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE

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

In [ ]:
```
lt_data =pd.read_csv('loantap-data.csv')
df = lt_data.copy()
df.head()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	e
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	

5 rows × 27 columns

In [ ]:
```
pd.set_option('display.max_columns', None)
```

🧐Exploration of data :

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

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

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394274 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

```
In [ ]: df.columns
```

```
Out[ ]: Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grad
e',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_typ
e',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

📝 Statistical Summary

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

Out[ ]:

	count	mean	std	min	25%
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00
int_rate	396030.0	13.639400	4.472157	5.32	10.49
installment	396030.0	431.849698	250.727790	16.08	250.33
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00
dti	396030.0	17.379514	18.019092	0.00	11.28
open_acc	396030.0	11.311153	5.137649	0.00	8.00
pub_rec	396030.0	0.178191	0.530671	0.00	0.00
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00
revol_util	395754.0	53.791749	24.452193	0.00	35.80
total_acc	396030.0	25.414744	11.886991	2.00	17.00
mort_acc	358235.0	1.813991	2.147930	0.00	0.00
pub_rec_bankruptcies	395495.0	0.121648	0.356174	0.00	0.00

In [ ]:

```
df.describe(include='object').T
```

Out[ ]:

	count	unique	top	freq
term	396030	2	36 months	302005
grade	396030	7	B	116018
sub_grade	396030	35	B3	26655
emp_title	373103	173105	Teacher	4389
emp_length	377729	11	10+ years	126041
home_ownership	396030	6	MORTGAGE	198348
verification_status	396030	3	Verified	139563
issue_d	396030	115	Oct-2014	14846
loan_status	396030	2	Fully Paid	318357
purpose	396030	14	debt_consolidation	234507
title	394274	48816	Debt consolidation	152472
earliest_cr_line	396030	684	Oct-2000	3017
initial_list_status	396030	2	f	238066
application_type	396030	3	INDIVIDUAL	395319
address	396030	393700	USCGC Smith\r\nFPO AE 70466	8

📜📜Duplicate Detection

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

Out[ ]:

loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_le

🏷️Insights

- The dataset does not contain any duplicates.

🔸 ❓Null Detection

In [ ]:
```
df.isna().any()[df.isna().any()]
```

Out[ ]:
```
emp_title               True
emp_length              True
title                   True
revol_util              True
mort_acc                True
pub_rec_bankruptcies    True
dtype: bool
```

In [ ]:
```
df.isna().sum().sort_values(ascending=False)
```

Out[ ]:
```
mort_acc                37795
emp_title               22927
emp_length              18301
title                    1756
pub_rec_bankruptcies      535
revol_util                276
loan_amnt                   0
dti                         0
application_type            0
initial_list_status         0
total_acc                   0
revol_bal                   0
pub_rec                     0
open_acc                    0
earliest_cr_line            0
purpose                     0
term                        0
loan_status                 0
issue_d                     0
verification_status         0
annual_inc                  0
home_ownership              0
sub_grade                   0
grade                       0
installment                 0
int_rate                    0
address                     0
dtype: int64
```

In [ ]:

```
def missing_data(df):
    total_missing_df = df.isnull().sum().sort_values(ascending =False)
    percent_missing_df = (df.isnull().sum()/df.isna().count()*100).sort_valu
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis
    return missing_data_df

missing_pct = missing_data(df)
missing_pct[missing_pct['Total']>0]
```

Out[ ]:

	Total	Percent
mort_acc	37795	9.543469
emp_title	22927	5.789208
emp_length	18301	4.621115
title	1756	0.443401
pub_rec_bankruptcies	535	0.135091
revol_util	276	0.069692

🏷️Insight

1. emp_title has 5.78% missing values
2. emp_length has 4.62% missing values
3. title has 0.44% missing values
4. revol_until has 0.06% missing values
5. mort_acc has 9.54% missing values
6. pub_rec_bankruptcies has 0.13% missing values

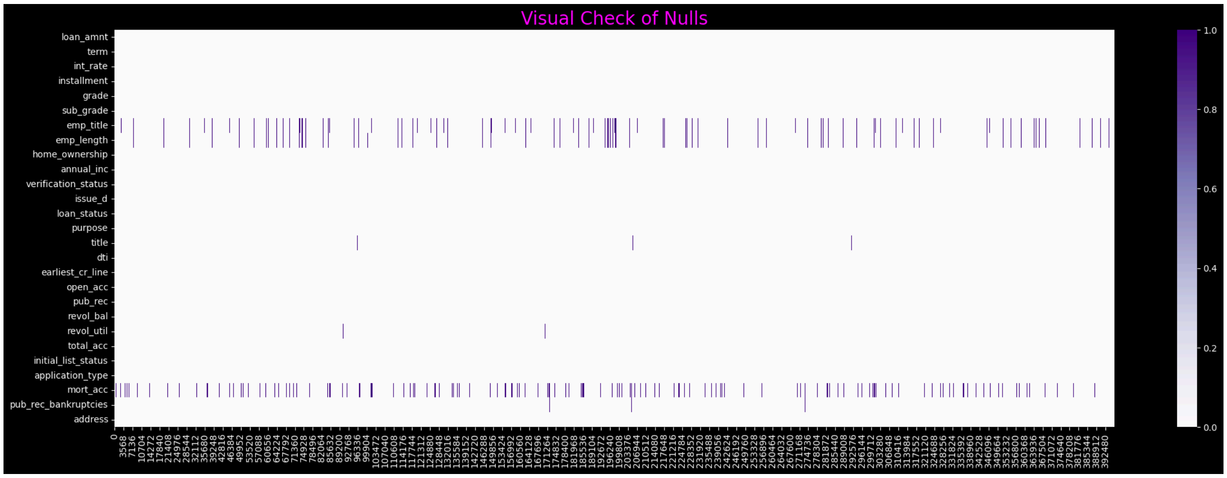
> Action

> Following columns has missing values

- Since ML algorithm do not work on columns which has missing values so we need to impute these missing values.

In [ ]:

```
plt.figure(figsize=(25,8))
plt.style.use('dark_background')
sns.heatmap(df.isnull().T,cmap='Purples')
plt.title('Visual Check of Nulls',fontsize=20,color='magenta')
plt.show()
```



In [ ]:

```
df.isna().sum().sum()
# since there are 81590 rows are null , we cant drop na ...
```

Out[ ]: 81590

In [ ]:

```
#checking the unique values for columns
for _ in df.columns:
    print()
    print(f'Total Unique Values in {_} column are :- {df[_].nunique()}')
    print(f'Unique Values in {_} column are :-\n {df[_].unique()}')
    print(f'Value_counts of {_} column :-\n {df[_].value_counts()}')
    print()
    print('-'*120)
```

```
Total Unique Values in loan_amnt column are :- 1397
Unique Values in loan_amnt column are :-
 [10000.  8000. 15600. ... 36275. 36475.   725.]
Value_counts of loan_amnt column :-
 loan_amnt
10000.0    27668
12000.0    21366
15000.0    19903
20000.0    18969
35000.0    14576
           ...
36225.0        1
950.0          1
37800.0        1
30050.0        1
725.0          1
Name: count, Length: 1397, dtype: int64

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

Total Unique Values in term column are :- 2
Unique Values in term column are :-
 [' 36 months' ' 60 months']
Value_counts of term column :-
 term
 36 months    302005
 60 months     94025
Name: count, dtype: int64

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

Total Unique Values in int_rate column are :- 566
Unique Values in int_rate column are :-
 [11.44 11.99 10.49  6.49 17.27 13.33  5.32 11.14 10.99 16.29 13.11 14.64
  9.17 12.29  6.62  8.39 21.98  7.9   6.97  6.99 15.61 11.36 13.35 12.12
  9.99  8.19 18.75  6.03 14.99 16.78 13.67 13.98 16.99 19.91 17.86 21.49
 12.99 18.54  7.89 17.1  18.25 11.67  6.24  8.18 12.35 14.16 17.56 18.55
 22.15 10.39 15.99 16.07 24.99  9.67 19.19 21.   12.69 10.74  6.68 19.22
 11.49 16.55 19.97 24.7  13.49 18.24 16.49 25.78 25.83 18.64  7.51 13.99
 15.22 15.31  7.69 19.53 10.16  7.62  9.75 13.68 15.88 14.65  6.92 23.83
 10.75 18.49 20.31 17.57 27.31 19.99 22.99 12.59 10.37 14.33 13.53 22.45
 24.5  17.99  9.16 12.49 11.55 17.76 28.99 23.1  20.49 22.7  10.15  6.89
 19.52  8.9  14.3   9.49 25.99 24.08 13.05 14.98 16.59 11.26 25.89 14.48
 21.99 23.99  5.99 14.47 11.53  8.67  8.59 10.64 23.28 25.44  9.71 16.2
 19.24 24.11 15.8  15.96 14.49 18.99  5.79 19.29 14.54 14.09  9.25 19.05
 17.77 18.92 20.75 10.65 18.85 10.59 12.85 11.39 13.65 13.06  7.12 20.99
 13.61 12.73 14.46 16.24 25.49  7.39 10.78 20.8   7.88 15.95 12.39 21.18
 21.97 15.77  6.39 10.   12.53 13.43  7.49 25.57 21.48 18.39 11.47  7.26
 15.68 19.04 14.31 24.24  5.42 23.43 19.47  6.54 23.32 17.58 14.72  7.66
  9.76 13.23 13.48 12.42  9.8  11.71 14.27 21.15 22.95  8.49 17.74 15.59
 13.72  9.45  7.29 15.1  11.86 19.72 14.35 11.22 15.62 15.81 12.41 28.67
 11.48 13.66  9.91 23.76 17.14 18.84 12.23  6.17  8.94 14.22 19.03 25.29
  8.99  9.88 15.58 27.49  8.07 22.47 19.2  13.44 22.4  12.79 18.2  13.18
  7.24 14.84  5.93 15.28 13.85 25.28  8.    9.62 12.05 15.7  20.2  13.57
```

```
 21.67  7.4  25.8  12.68 11.83  7.37 11.11 14.85 16.   11.12 23.63  6.
  7.99  7.91 14.83 21.7  26.06 16.77 27.34 12.21  7.68 15.27 19.69  9.63
  7.14 20.5  16.02 12.84  7.74 15.33 19.79 22.2  18.62 17.49 16.89 15.21
 14.79 18.67  9.32 15.41 15.65 23.5  22.9  11.34 22.11 19.48 14.75 28.14
 13.22 23.4  23.13 28.18 12.88 22.06 24.49 16.45 21.6  28.49  8.38  6.76
 10.83 13.79  8.88 17.88 17.97 14.26  6.91 13.47  8.6  27.88  8.63 10.25
 14.91 12.74 10.96 25.88  7.43 16.4  20.25 24.89 12.87 20.16 14.17 12.18
 17.51 13.92 20.53 26.77 10.62 26.49 16.32 12.61 21.36 14.61 15.37 20.3
 14.59 16.7  19.89 10.95 18.17 18.21 17.93 22.39 24.83 13.8  19.42 23.7
  7.59 13.17 18.09 13.04 25.69  9.07 15.23 14.42 23.33 16.69 10.36 14.96
 10.38 26.24 24.2  12.98 20.85 13.36 26.57 23.52 22.78 13.16 15.13 25.11
 13.55 10.51 11.78  7.05 11.46 21.28 12.09 16.35  8.7  26.99 14.11 26.14
 16.82 23.26 18.79 10.28 19.36 18.3  17.06 17.19  7.75 17.34 20.89 22.35
 19.66 13.62 22.74 11.89 23.59  8.24 20.62 11.97 15.2  20.48 12.36 10.71
 25.09 20.11 27.79 29.49 11.58 19.13 11.66 13.75 30.74  9.38 27.99 11.59
  9.64 25.65  9.96 19.41 14.18 10.08 17.43 24.74 14.74 17.04 15.57 30.49
 17.8  10.91 14.82 29.96 12.92 12.22 15.45 11.72 10.2  14.7  20.69 15.05
 24.33 14.93 10.33 16.95 28.88 11.03 28.34 21.22 18.07  9.33 12.17 19.74
 20.9  20.03 17.39 29.67 12.04 23.22 10.01 22.48 24.76 13.3  20.77 10.14
 14.5  30.94  8.32 13.24 21.59 21.27 24.52 11.54 10.46 13.87 30.99  9.51
  9.83 19.39 12.86 30.79 21.74 11.09 16.11 17.26 22.85 18.91 18.43  9.2
 21.14 12.62 21.21 29.99 14.88 13.12 30.89 16.08 12.54 28.69 12.8  11.28
 23.91 22.94 19.16 20.86 11.63 19.82 11.41 21.82 12.72 20.4   9.7  18.72
 18.36 14.25 13.84 18.78 17.15 15.25 16.63 16.15 11.91 14.07  9.01 15.01
 21.64 15.83 18.53  7.42 12.67 15.76 16.33 30.84 13.93 14.12 14.28 20.17
 24.59 20.52 17.03 17.9  14.67 15.38 17.46 14.62 14.38 24.4  22.64 17.54
 17.44 15.07]
Value_counts of int_rate column :-
 int_rate
10.99    12411
12.99     9632
15.61     9350
11.99     8582
8.90      8019
         ...
14.28        1
18.72        1
18.36        1
30.84        1
24.59        1
Name: count, Length: 566, dtype: int64

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

Total Unique Values in installment column are :- 55706
Unique Values in installment column are :-
 [329.48 265.68 506.97 ... 343.14 118.13 572.44]
Value_counts of installment column :-
 installment
327.34     968
332.10     791
491.01     736
336.90     686
392.81     683
          ...
```

```
364.37       1
1015.29      1
398.04       1
544.94       1
572.44       1
Name: count, Length: 55706, dtype: int64

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

Total Unique Values in grade column are :- 7
Unique Values in grade column are :-
 ['B' 'A' 'C' 'E' 'D' 'F' 'G']
Value_counts of grade column :-
 grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64

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

Total Unique Values in sub_grade column are :- 35
Unique Values in sub_grade column are :-
 ['B4' 'B5' 'B3' 'A2' 'C5' 'C3' 'A1' 'B2' 'C1' 'A5' 'E4' 'A4' 'A3' 'D1'
 'C2' 'B1' 'D3' 'D5' 'D2' 'E1' 'E2' 'E5' 'F4' 'E3' 'D4' 'G1' 'F5' 'G2'
 'C4' 'F1' 'F3' 'G5' 'G4' 'F2' 'G3']
Value_counts of sub_grade column :-
 sub_grade
B3    26655
B4    25601
C1    23662
C2    22580
B2    22495
B5    22085
C3    21221
C4    20280
B1    19182
A5    18526
C5    18244
D1    15993
A4    15789
D2    13951
D3    12223
D4    11657
A3    10576
A1     9729
D5     9700
A2     9567
E1     7917
E2     7431
```

```
E3    6207
E4    5361
E5    4572
F1    3536
F2    2766
F3    2286
F4    1787
F5    1397
G1    1058
G2     754
G3     552
G4     374
G5     316
Name: count, dtype: int64

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

Total Unique Values in emp_title column are :- 173105
Unique Values in emp_title column are :-
 ['Marketing' 'Credit analyst ' 'Statistician' ...
 "Michael's Arts & Crafts" 'licensed bankere' 'Gracon Services, Inc']
Value_counts of emp_title column :-
 emp_title
Teacher                    4389
Manager                    4250
Registered Nurse           1856
RN                         1846
Supervisor                 1830
                           ...
Postman                       1
McCarthy & Holthus, LLC       1
jp flooring                   1
Histology Technologist        1
Gracon Services, Inc          1
Name: count, Length: 173105, dtype: int64

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

Total Unique Values in emp_length column are :- 11
Unique Values in emp_length column are :-
 ['10+ years' '4 years' '< 1 year' '6 years' '9 years' '2 years' '3 years'
 '8 years' '7 years' '5 years' '1 year' nan]
Value_counts of emp_length column :-
 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

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

Total Unique Values in home_ownership column are :- 6
Unique Values in home_ownership column are :-
 ['RENT' 'MORTGAGE' 'OWN' 'OTHER' 'NONE' 'ANY']
Value_counts of home_ownership column :-
 home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64

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

Total Unique Values in annual_inc column are :- 27197
Unique Values in annual_inc column are :-
 [117000.    65000.    43057.   ...  36111.    47212.    31789.88]
Value_counts of annual_inc column :-
 annual_inc
60000.00    15313
50000.00    13303
65000.00    11333
70000.00    10674
40000.00    10629
            ...
72179.00        1
50416.00        1
46820.80        1
10368.00        1
31789.88        1
Name: count, Length: 27197, dtype: int64

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

Total Unique Values in verification_status column are :- 3
Unique Values in verification_status column are :-
 ['Not Verified' 'Source Verified' 'Verified']
Value_counts of verification_status column :-
 verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64

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

```
Total Unique Values in issue_d column are :- 115
Unique Values in issue_d column are :-
 ['Jan-2015' 'Nov-2014' 'Apr-2013' 'Sep-2015' 'Sep-2012' 'Oct-2014'
 'Apr-2012' 'Jun-2013' 'May-2014' 'Dec-2015' 'Apr-2015' 'Oct-2012'
 'Jul-2014' 'Feb-2013' 'Oct-2015' 'Jan-2014' 'Mar-2016' 'Apr-2014'
 'Jun-2011' 'Apr-2010' 'Jun-2014' 'Oct-2013' 'May-2013' 'Feb-2015'
 'Oct-2011' 'Jun-2015' 'Aug-2013' 'Feb-2014' 'Dec-2011' 'Mar-2013'
 'Jun-2016' 'Mar-2014' 'Nov-2013' 'Dec-2014' 'Apr-2016' 'Sep-2013'
 'May-2016' 'Jul-2015' 'Jul-2013' 'Aug-2014' 'May-2008' 'Mar-2010'
 'Dec-2013' 'Mar-2012' 'Mar-2015' 'Sep-2011' 'Jul-2012' 'Dec-2012'
 'Sep-2014' 'Nov-2012' 'Nov-2015' 'Jan-2011' 'May-2012' 'Feb-2016'
 'Jun-2012' 'Aug-2012' 'Jan-2016' 'May-2015' 'Oct-2016' 'Aug-2015'
 'Jul-2016' 'May-2009' 'Aug-2016' 'Jan-2012' 'Jan-2013' 'Nov-2010'
 'Jul-2011' 'Mar-2011' 'Feb-2012' 'May-2011' 'Aug-2010' 'Nov-2016'
 'Jul-2010' 'Sep-2010' 'Dec-2010' 'Feb-2011' 'Jun-2009' 'Aug-2011'
 'Dec-2016' 'Mar-2009' 'Jun-2010' 'May-2010' 'Nov-2011' 'Sep-2016'
 'Oct-2009' 'Mar-2008' 'Nov-2008' 'Dec-2009' 'Oct-2010' 'Sep-2009'
 'Oct-2007' 'Aug-2009' 'Jul-2009' 'Nov-2009' 'Jan-2010' 'Dec-2008'
 'Feb-2009' 'Oct-2008' 'Apr-2009' 'Feb-2010' 'Apr-2011' 'Apr-2008'
 'Aug-2008' 'Jan-2009' 'Feb-2008' 'Aug-2007' 'Sep-2008' 'Dec-2007'
 'Jan-2008' 'Sep-2007' 'Jun-2008' 'Jul-2008' 'Jun-2007' 'Nov-2007'
 'Jul-2007']
Value_counts of issue_d column :-
 issue_d
Oct-2014    14846
Jul-2014    12609
Jan-2015    11705
Dec-2013    10618
Nov-2013    10496
            ...
Jul-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: count, Length: 115, dtype: int64

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

Total Unique Values in loan_status column are :- 2
Unique Values in loan_status column are :-
 ['Fully Paid' 'Charged Off']
Value_counts of loan_status column :-
 loan_status
Fully Paid     318357
Charged Off     77673
Name: count, dtype: int64

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

Total Unique Values in purpose column are :- 14
Unique Values in purpose column are :-
 ['vacation' 'debt_consolidation' 'credit_card' 'home_improvement'
 'small_business' 'major_purchase' 'other' 'medical' 'wedding' 'car'
```

```
 'moving' 'house' 'educational' 'renewable_energy']
Value_counts of purpose column :-
 purpose
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
car                     4697
medical                 4196
moving                  2854
vacation                2452
house                   2201
wedding                 1812
renewable_energy         329
educational              257
Name: count, dtype: int64

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

Total Unique Values in title column are :- 48816
Unique Values in title column are :-
 ['Vacation' 'Debt consolidation' 'Credit card refinancing' ...
 'Credit buster ' 'Loanforpayoff' 'Toxic Debt Payoff']
Value_counts of title column :-
 title
Debt consolidation            152472
Credit card refinancing        51487
Home improvement               15264
Other                          12930
Debt Consolidation             11608
                                ...
Graduation/Travel Expenses         1
Daughter's Wedding Bill            1
gotta move                         1
creditcardrefi                     1
Toxic Debt Payoff                  1
Name: count, Length: 48816, dtype: int64

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

Total Unique Values in dti column are :- 4262
Unique Values in dti column are :-
 [26.24 22.05 12.79 ... 40.56 47.09 55.53]
Value_counts of dti column :-
 dti
0.00     313
14.40    310
19.20    302
16.80    301
18.00    300
        ...
59.18      1
```

```
48.37      1
45.71      1
42.38      1
55.53      1
Name: count, Length: 4262, dtype: int64

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

Total Unique Values in earliest_cr_line column are :- 684
Unique Values in earliest_cr_line column are :-
 ['Jun-1990' 'Jul-2004' 'Aug-2007' 'Sep-2006' 'Mar-1999' 'Jan-2005'
 'Aug-2005' 'Sep-1994' 'Jun-1994' 'Dec-1997' 'Dec-1990' 'May-1984'
 'Apr-1995' 'Jan-1997' 'May-2001' 'Mar-1982' 'Sep-1996' 'Jan-1990'
 'Mar-2000' 'Jan-2006' 'Oct-2006' 'Jan-2003' 'May-2008' 'Oct-2003'
 'Jun-2004' 'Jan-1999' 'Apr-1994' 'Apr-1998' 'Jul-2007' 'Apr-2002'
 'Oct-2007' 'Jun-2009' 'May-1997' 'Jul-2006' 'Sep-2003' 'Aug-1992'
 'Dec-1988' 'Feb-2002' 'Jan-1992' 'Aug-2001' 'Dec-2010' 'Oct-1999'
 'Sep-2004' 'Aug-1994' 'Jul-2003' 'Apr-2000' 'Dec-2004' 'Jun-1995'
 'Dec-2003' 'Jul-1994' 'Oct-1990' 'Dec-2001' 'Apr-1999' 'Feb-1995'
 'May-2003' 'Oct-2002' 'Mar-2004' 'Aug-2003' 'Oct-2000' 'Nov-2004'
 'Mar-2010' 'Mar-1996' 'May-1994' 'Jun-1996' 'Nov-1986' 'Jan-2001'
 'Jan-2002' 'Mar-2001' 'Sep-2012' 'Apr-2006' 'May-1998' 'Dec-2002'
 'Nov-2003' 'Oct-2005' 'May-1990' 'Jun-2003' 'Jun-2001' 'Jan-1998'
 'Oct-1978' 'Feb-2001' 'Jun-2006' 'Aug-1993' 'Apr-2001' 'Nov-2001'
 'Feb-2003' 'Jun-1993' 'Sep-1992' 'Nov-1992' 'Jun-1983' 'Oct-2001'
 'Jul-1999' 'Sep-1997' 'Nov-1993' 'Feb-1993' 'Apr-2007' 'Nov-1999'
 'Nov-2005' 'Dec-1992' 'Mar-1986' 'May-1989' 'Dec-2000' 'Mar-1991'
 'Mar-2005' 'Jun-2010' 'Dec-1998' 'Sep-2001' 'Nov-2000' 'Jan-1994'
 'Aug-2002' 'Jan-2011' 'Aug-2008' 'Jun-2005' 'Nov-1997' 'May-1996'
 'Apr-2010' 'May-1993' 'Sep-2005' 'Jun-1992' 'Apr-1986' 'Aug-1996'
 'Aug-1997' 'Jul-2005' 'May-2011' 'Sep-2002' 'Jan-1989' 'Aug-1999'
 'Feb-1992' 'Sep-1999' 'Jul-2001' 'May-1980' 'Oct-2008' 'Nov-2007'
 'Apr-1997' 'Jun-1986' 'Sep-1998' 'Jun-1982' 'Oct-1981' 'Feb-1994'
 'Dec-1984' 'Nov-1991' 'Nov-2006' 'Aug-2000' 'Oct-2004' 'Jun-2011'
 'Apr-1988' 'May-2004' 'Aug-1988' 'Mar-1994' 'Aug-2004' 'Dec-2006'
 'Nov-1998' 'Oct-1997' 'Mar-1989' 'Feb-1988' 'Jul-1982' 'Nov-1995'
 'Mar-1997' 'Oct-1994' 'Jul-1998' 'Jun-2002' 'May-1991' 'Oct-2011'
 'Sep-2007' 'Jan-2007' 'Jan-2010' 'Mar-1987' 'Feb-1997' 'Oct-1986'
 'Mar-2002' 'Jul-1993' 'Mar-2007' 'Aug-1989' 'Oct-1995' 'May-2007'
 'Dec-1993' 'Jun-1989' 'Apr-2004' 'Jun-1997' 'Apr-1996' 'Apr-1992'
 'Oct-1998' 'Mar-1983' 'Mar-1985' 'Oct-1993' 'Feb-2000' 'Apr-2003'
 'Oct-1985' 'Jul-1985' 'May-1978' 'Sep-2010' 'Oct-1996' 'Sep-2009'
 'Jun-1999' 'Jan-2000' 'Sep-1987' 'Aug-1998' 'Jan-1995' 'Jul-1988'
 'May-2000' 'Jun-1981' 'Feb-1998' 'Nov-1996' 'Aug-1967' 'Dec-1999'
 'Aug-2006' 'Nov-2009' 'Jul-2000' 'Mar-1988' 'Jul-1992' 'Jul-1991'
 'Mar-1990' 'May-1986' 'Jun-1991' 'Dec-1987' 'Jul-1996' 'Jul-1997'
 'Aug-1990' 'Jan-1988' 'Dec-2005' 'Mar-2003' 'Feb-1999' 'Nov-1990'
 'Jun-2000' 'Dec-1996' 'Jan-2004' 'May-1999' 'Sep-1972' 'Jul-1981'
 'Sep-1993' 'Feb-2009' 'Nov-2002' 'Nov-1969' 'Jan-1993' 'May-2005'
 'Sep-1982' 'Apr-1990' 'Feb-1996' 'Mar-1993' 'Apr-1978' 'Jul-1995'
 'May-1995' 'Apr-1991' 'Mar-1998' 'Aug-1991' 'Jul-2002' 'Oct-1989'
 'Apr-1984' 'Dec-2009' 'Sep-2000' 'Jan-1982' 'Jun-1998' 'Jan-1996'
 'Nov-1987' 'May-2010' 'Jul-1989' 'Jun-1987' 'Oct-1987' 'Aug-1995'
 'Feb-2004' 'Oct-1991' 'Dec-1989' 'Oct-1992' 'Feb-2005' 'Apr-1993'
 'Dec-1985' 'Sep-1979' 'Feb-2007' 'Nov-1989' 'Apr-2005' 'Mar-1978'
```

'Sep-1985'	'Nov-1994'	'Jun-2008'	'Apr-1987'	'Dec-1983'	'Dec-2007'
'May-1979'	'May-1992'	'Jul-1990'	'Mar-1995'	'Feb-2006'	'Feb-1985'
'Sep-1989'	'Aug-2009'	'Nov-2008'	'Nov-1981'	'Jan-2008'	'Aug-1987'
'Nov-1985'	'Dec-1965'	'Sep-1995'	'Jan-1986'	'Oct-2009'	'May-2002'
'Aug-1980'	'Sep-1977'	'Sep-1988'	'Oct-1984'	'May-1988'	'Aug-1984'
'Nov-1988'	'May-1974'	'Nov-1982'	'Oct-1983'	'Sep-1991'	'Feb-1984'
'Feb-1991'	'Jan-1981'	'Jun-1985'	'Dec-1976'	'Dec-1994'	'Dec-1980'
'Sep-1984'	'Jun-2007'	'Aug-1979'	'Sep-2008'	'Apr-1983'	'Mar-2006'
'Jun-1984'	'Jul-1984'	'Jan-1985'	'Dec-1995'	'Apr-2008'	'Mar-2008'
'Jan-1983'	'Dec-1986'	'Jun-1979'	'Dec-1975'	'Nov-1983'	'Jul-1986'
'Nov-1977'	'Dec-1982'	'May-1985'	'Feb-1983'	'Aug-1982'	'Oct-1980'
'Mar-1979'	'Jan-1978'	'Mar-1984'	'May-1983'	'Jul-2008'	'Apr-1982'
'Jul-1983'	'Feb-1990'	'Dec-2008'	'Jul-1975'	'Dec-1971'	'Feb-2008'
'Mar-2011'	'Feb-1987'	'Feb-1989'	'Aug-1985'	'Jul-2010'	'Apr-1989'
'Feb-1980'	'May-2006'	'Nov-2010'	'Apr-2009'	'Feb-2010'	'May-1976'
'Feb-1981'	'Jan-2012'	'Oct-1988'	'Nov-1984'	'May-1982'	'Oct-1975'
'Jun-1988'	'May-1972'	'Apr-2013'	'Sep-1990'	'Oct-1982'	'Feb-2013'
'Mar-1992'	'Aug-1981'	'Feb-2011'	'Nov-1974'	'Feb-1978'	'Sep-1983'
'Jul-2011'	'Nov-1979'	'Aug-1983'	'Apr-1985'	'Jul-2009'	'Jan-1971'
'Jul-1987'	'Aug-1978'	'Aug-2010'	'Oct-1976'	'Aug-1986'	'Jan-1991'
'Dec-1991'	'May-2009'	'Aug-2011'	'Jun-1964'	'Jan-1974'	'May-1981'
'Jun-1972'	'Jun-1978'	'Sep-1986'	'Jan-1987'	'Jan-1975'	'Feb-1982'
'Jan-1980'	'Feb-1977'	'Sep-1980'	'Nov-1978'	'Jul-1974'	'Jun-1970'
'Jan-1984'	'Nov-1980'	'May-1987'	'Sep-1970'	'Jan-1976'	'Feb-1986'
'Oct-2010'	'Apr-1979'	'Oct-1979'	'Jan-1979'	'Sep-2011'	'Jul-1979'
'Sep-1975'	'Mar-1981'	'Aug-1971'	'Apr-1980'	'Apr-1977'	'Jan-1965'
'Nov-1976'	'Nov-1970'	'Nov-2011'	'Nov-1973'	'Sep-1981'	'Jul-1980'
'Mar-2012'	'Dec-1974'	'Mar-1977'	'Dec-1977'	'May-2012'	'Dec-1979'
'Jan-2009'	'Jan-1970'	'Dec-2011'	'Feb-1979'	'Mar-1976'	'Jan-1973'
'Oct-1973'	'Mar-1969'	'Oct-1977'	'Mar-1975'	'Aug-1977'	'Jun-1969'
'Oct-1963'	'Nov-1960'	'Aug-1970'	'Feb-1975'	'Sep-1974'	'May-1966'
'Apr-1972'	'Apr-1973'	'Apr-2012'	'May-1975'	'Sep-1966'	'Feb-1969'
'Feb-2012'	'Jan-1961'	'Aug-1973'	'Feb-1972'	'Apr-1975'	'Jul-1978'
'Oct-1970'	'Mar-1980'	'Sep-1976'	'Apr-2011'	'Nov-2012'	'Aug-1976'
'Jun-1975'	'Apr-1981'	'Mar-2009'	'Jun-1977'	'Apr-1971'	'Sep-1969'
'Jun-2012'	'Apr-1976'	'Feb-1965'	'Jul-1977'	'Jun-1976'	'Mar-1973'
'Oct-1972'	'Dec-1978'	'Nov-1967'	'Sep-1967'	'Nov-1971'	'Jun-1980'
'May-1964'	'Feb-1971'	'May-1970'	'Apr-1970'	'Mar-1971'	'Apr-1969'
'Jan-1963'	'Jun-1974'	'Oct-1974'	'May-1977'	'Dec-1981'	'Jan-1969'
'Feb-1976'	'Mar-1970'	'Aug-1968'	'Feb-1970'	'Jun-1971'	'Jun-1963'
'Jun-2013'	'Mar-1972'	'Aug-2012'	'Jan-1967'	'Feb-1968'	'Dec-1969'
'Jan-1977'	'Jul-1970'	'Feb-1973'	'Mar-1974'	'Feb-1974'	'Dec-1960'
'Jul-1972'	'Jul-1973'	'Sep-1964'	'Jul-1965'	'Oct-1958'	'Jul-2012'
'Jun-1973'	'Sep-1978'	'Nov-1975'	'Jul-1963'	'Jan-1964'	'Dec-1968'
'May-1958'	'Sep-1973'	'May-1971'	'Dec-1972'	'Aug-1965'	'Jul-1976'
'Oct-2012'	'May-1973'	'Apr-1955'	'Apr-1966'	'Jan-1968'	'Nov-1968'
'Oct-1969'	'Mar-2013'	'Jan-2013'	'Jul-1967'	'Oct-1965'	'Jan-1966'
'Aug-1972'	'Jul-1969'	'May-1965'	'Jan-1953'	'Aug-1974'	'May-1968'
'Aug-1969'	'May-2013'	'Oct-1967'	'Aug-1975'	'Apr-1974'	'Sep-1971'
'Apr-1968'	'Jul-1971'	'Jan-1972'	'Nov-1965'	'Dec-1970'	'Dec-1973'
'Nov-1972'	'Oct-1959'	'Oct-1962'	'Apr-1967'	'Oct-1971'	'Nov-1963'
'Oct-1968'	'Dec-1962'	'Jun-1960'	'Jan-1960'	'Sep-2013'	'May-1969'
'Dec-1966'	'Feb-1967'	'Dec-1967'	'Aug-1961'	'Sep-1968'	'Oct-1964'
'Aug-1966'	'Jul-1966'	'Apr-1964'	'Sep-1962'	'Jul-2013'	'Jun-1967'
'Apr-1965'	'Jun-1966'	'Jan-1955'	'Jan-1962'	'Feb-1964'	'Aug-1958'
'Jul-1968'	'May-1967'	'Dec-1959'	'Sep-1963'	'Dec-2012'	'Dec-1963'

```
 'Jan-1944' 'Jun-1965' 'May-1962' 'Mar-1967' 'Mar-1968' 'Jan-1956'
 'Sep-1965' 'Dec-1951' 'Aug-2013' 'Jun-1968' 'Mar-1965' 'Oct-1957'
 'Nov-1966' 'Dec-1958' 'Feb-1957' 'Feb-1963' 'Mar-1963' 'Jan-1959'
 'May-1955' 'Feb-1966' 'Nov-1950' 'Mar-1964' 'Jan-1958' 'Nov-1964'
 'Sep-1961' 'Apr-1963' 'Jul-1964' 'Nov-1955' 'Jun-1957' 'Dec-1964'
 'Nov-1953' 'Apr-1961' 'Mar-1966' 'Oct-1960' 'Jul-1959' 'Jul-1961'
 'Jan-1954' 'Dec-1956' 'Mar-1962' 'Jul-1960' 'Sep-1959' 'Dec-1950'
 'Oct-1966' 'Apr-1960' 'Jul-1958' 'Nov-1954' 'Nov-1957' 'Jun-1962'
 'May-1963' 'Jul-1955' 'Oct-1950' 'Dec-1961' 'Aug-1951' 'Oct-2013'
 'Aug-1964' 'Apr-1962' 'Jun-1955' 'Jul-1962' 'Jan-1957' 'Nov-1958'
 'Jul-1951' 'Nov-1959' 'Apr-1958' 'Mar-1960' 'Sep-1957' 'Nov-1961'
 'Sep-1960' 'May-1959' 'Jun-1959' 'Feb-1962' 'Sep-1956' 'Aug-1960'
 'Feb-1961' 'Jan-1948' 'Aug-1963' 'Oct-1961' 'Aug-1962' 'Aug-1959']
Value_counts of earliest_cr_line column :-
 earliest_cr_line
Oct-2000    3017
Aug-2000    2935
Oct-2001    2896
Aug-2001    2884
Nov-2000    2736
            ...
Jul-1958       1
Nov-1957       1
Jan-1953       1
Jul-1955       1
Aug-1959       1
Name: count, Length: 684, dtype: int64

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

Total Unique Values in open_acc column are :- 61
Unique Values in open_acc column are :-
 [16. 17. 13.  6.  8. 11.  5. 30.  9. 15. 12. 10. 18.  7.  4. 14. 20. 19.
 21. 23.  3. 26. 42. 22. 25. 28.  2. 34. 24. 27. 31. 32. 33.  1. 29. 36.
 40. 35. 37. 41. 44. 39. 49. 48. 38. 51. 50. 43. 46.  0. 47. 57. 53. 58.
 52. 54. 45. 90. 56. 55. 76.]
Value_counts of open_acc column :-
 open_acc
9.0     36779
10.0    35441
8.0     35137
11.0    32695
7.0     31328
        ...
55.0        2
76.0        2
58.0        1
57.0        1
90.0        1
Name: count, Length: 61, dtype: int64

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

Total Unique Values in pub_rec column are :- 20
```

```
Unique Values in pub_rec column are :-
 [ 0.  1.  2.  3.  4.  6.  5.  8.  9. 10. 11.  7. 19. 13. 40. 17. 86. 12.
 24. 15.]
Value_counts of pub_rec column :-
 pub_rec
0.0     338272
1.0      49739
2.0       5476
3.0       1521
4.0        527
5.0        237
6.0        122
7.0         56
8.0         34
9.0         12
10.0        11
11.0         8
13.0         4
12.0         4
19.0         2
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: count, dtype: int64

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

Total Unique Values in revol_bal column are :- 55622
Unique Values in revol_bal column are :-
 [ 36369.  20131.  11987. ...  34531. 151912.  29244.]
Value_counts of revol_bal column :-
 revol_bal
0.0         2128
5655.0        41
6095.0        38
7792.0        38
3953.0        37
            ...
42573.0        1
72966.0        1
105342.0       1
37076.0        1
29244.0        1
Name: count, Length: 55622, dtype: int64

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

Total Unique Values in revol_util column are :- 1226
Unique Values in revol_util column are :-
 [ 41.8   53.3   92.2  ...  56.26 111.4  128.1 ]
Value_counts of revol_util column :-
 revol_util
```

```
0.00      2213
53.00      752
60.00      739
61.00      734
55.00      730
          ...
892.30       1
110.10       1
123.00       1
49.63        1
128.10       1
Name: count, Length: 1226, dtype: int64

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

Total Unique Values in total_acc column are :- 118
Unique Values in total_acc column are :-
 [ 25.  27.  26.  13.  43.  23.  15.  40.  37.  61.  35.  22.  20.  36.
  38.   7.  18.  10.  17.  29.  16.  21.  34.   9.  14.  59.  41.  19.
  12.  30.  56.  24.  28.   8.  52.  31.  44.  39.  50.  11.  62.  32.
   5.  33.  46.  42.   6.  49.  45.  57.  48.  67.  47.  51.  58.   3.
  55.  63.  53.   4.  71.  69.  54.  64.  81.  72.  60.  68.  65.  73.
  78.  84.   2.  76.  75.  79.  87.  77. 104.  89.  70. 105.  97.  66.
 108.  74.  80.  82.  91.  93. 106.  90.  85.  88.  83. 111.  86. 101.
 135.  92.  94.  95.  99. 102. 129. 110. 124. 151. 107. 118. 150. 115.
 117.  96.  98. 100. 116. 103.]
Value_counts of total_acc column :-
 total_acc
21.0     14280
22.0     14260
20.0     14228
23.0     13923
24.0     13878
         ...
110.0        1
129.0        1
135.0        1
104.0        1
103.0        1
Name: count, Length: 118, dtype: int64

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

Total Unique Values in initial_list_status column are :- 2
Unique Values in initial_list_status column are :-
 ['w' 'f']
Value_counts of initial_list_status column :-
 initial_list_status
f    238066
w    157964
Name: count, dtype: int64

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

```
Total Unique Values in application_type column are :- 3
Unique Values in application_type column are :-
 ['INDIVIDUAL' 'JOINT' 'DIRECT_PAY']
Value_counts of application_type column :-
 application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64

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

Total Unique Values in mort_acc column are :- 33
Unique Values in mort_acc column are :-
 [ 0.  3.  1.  4.  2.  6.  5. nan 10.  7. 12. 11.  8.  9. 13. 14. 22. 34.
 15. 25. 19. 16. 17. 32. 18. 24. 21. 20. 31. 28. 30. 23. 26. 27.]
Value_counts of mort_acc column :-
 mort_acc
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
19.0        15
20.0        13
24.0        10
22.0         7
21.0         4
25.0         4
27.0         3
32.0         2
31.0         2
23.0         2
26.0         2
28.0         1
30.0         1
34.0         1
Name: count, dtype: int64

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

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

Total Unique Values in pub_rec_bankruptcies column are :- 9
Unique Values in pub_rec_bankruptcies column are :-
 [ 0.  1.  2.  3. nan  4.  5.  6.  7.  8.]
Value_counts of pub_rec_bankruptcies column :-
 pub_rec_bankruptcies
0.0    350380
1.0     42790
2.0      1847
3.0       351
4.0        82
5.0        32
6.0         7
7.0         4
8.0         2
Name: count, dtype: int64

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

Total Unique Values in address column are :- 393700
Unique Values in address column are :-
 ['0174 Michelle Gateway\r\nMendozaberg, OK 22690'
 '1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113'
 '87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113' ...
 '953 Matthew Points Suite 414\r\nReedfort, NY 70466'
 '7843 Blake Freeway Apt. 229\r\nNew Michael, FL 29597'
 '787 Michelle Causeway\r\nBriannaton, AR 48052']
Value_counts of address column :-
 address
USCGC Smith\r\nFPO AE 70466                          8
USS Johnson\r\nFPO AE 48052                          8
USNS Johnson\r\nFPO AE 05113                         8
USS Smith\r\nFPO AP 70466                            8
USNS Johnson\r\nFPO AP 48052                         7
                                                    ..
455 Tricia Cove\r\nAustinbury, FL 00813              1
7776 Flores Fall\r\nFernandezshire, UT 05113         1
6577 Mia Harbors Apt. 171\r\nRobertshire, OK 22690    1
8141 Cox Greens Suite 186\r\nMadisonstad, VT 05113    1
787 Michelle Causeway\r\nBriannaton, AR 48052        1
Name: count, Length: 393700, dtype: int64

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

🔴Null Treatment:

In [ ]:

```
df.loc[df['revol_util'].isna(),'revol_util'] = 0.0
df.loc[df['mort_acc'].isna(),'mort_acc'] = 0.0
df.loc[df['pub_rec_bankruptcies'].isna(),'pub_rec_bankruptcies'] = 0.0
df.loc[df['emp_title'].isna(),'emp_title'] = 'No Employee Title'
```

```python
df.loc[df['title'].isna(),'title'] = 'Unavailable'
df['emp_length'] = df['emp_length'].fillna('< 1 year')
```

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

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

In [ ]:
```python
df.describe().T
```

Out[ ]:

	count	mean	std	min	25%
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00
int_rate	396030.0	13.639400	4.472157	5.32	10.49
installment	396030.0	431.849698	250.727790	16.08	250.33
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00
dti	396030.0	17.379514	18.019092	0.00	11.28
open_acc	396030.0	11.311153	5.137649	0.00	8.00
pub_rec	396030.0	0.178191	0.530671	0.00	0.00
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00
revol_util	396030.0	53.754260	24.484857	0.00	35.80
total_acc	396030.0	25.414744	11.886991	2.00	17.00
mort_acc	396030.0	1.640873	2.111249	0.00	0.00
pub_rec_bankruptcies	396030.0	0.121483	0.355962	0.00	0.00

In [ ]:

```
df.describe(include='object').T
```

Out[ ]:

	count	unique	top	freq
term	396030	2	36 months	302005
grade	396030	7	B	116018
sub_grade	396030	35	B3	26655
emp_title	396030	173106	No Employee Title	22927
emp_length	396030	11	10+ years	126041
home_ownership	396030	6	MORTGAGE	198348
verification_status	396030	3	Verified	139563
issue_d	396030	115	Oct-2014	14846
loan_status	396030	2	Fully Paid	318357
purpose	396030	14	debt_consolidation	234507
title	396030	48817	Debt consolidation	152472
earliest_cr_line	396030	684	Oct-2000	3017
initial_list_status	396030	2	f	238066
application_type	396030	3	INDIVIDUAL	395319
address	396030	393700	USCGC Smith\r\nFPO AE 70466	8

🗒️Feature Engineering

In [ ]:
```
df['pub_rec'] = [1 if i > 1 else 0 for i in df['pub_rec']]
df['mort_acc'] = [1 if i > 1 else 0 for i in df['mort_acc']]
df['pub_rec_bankruptcies'] = [1 if i > 1 else 0 for i in df['pub_rec_bankrup
```

In [ ]:
```
df.sample()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_t
60136	35000.0	36 months	12.29	1167.36	C	C1	Superinten of Maintena

In [ ]:
```
#Split issue_date into month and year
df[['issue_month', 'issue_year']] = df['issue_d'].str.split('-', expand=True
df.drop(['issue_d'], axis=1, inplace=True)
```

In [ ]:
```
#Split er_cr_line date into month and year
df[['er_cr_line_m', 'er_cr_line_y']] = df['earliest_cr_line'].str.split('-',
df.drop(['earliest_cr_line'], axis=1, inplace=True)
```

In [ ]:
```
df['address']
```

Out[ ]:
```
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
                                 ...
396025     12951 Williams Crossing\r\nJohnnyville, DC 30723
396026    0114 Fowler Field Suite 028\r\nRachelborough, ...
396027    953 Matthew Points Suite 414\r\nReedfort, NY 7...
396028    7843 Blake Freeway Apt. 229\r\nNew Michael, FL...
396029       787 Michelle Causeway\r\nBriannaton, AR 48052
Name: address, Length: 396030, dtype: object
```

In [ ]:
```
#Split address into State and Zip code
import re
df[['state','zipcode']] = df['address'].str.extract(r'([A-Z]{2}) (\d{5})')
df.drop(['address'], axis=1, inplace=True)
```

In [ ]:
```
df['state'].nunique() , df['zipcode'].nunique()
```

Out[ ]:
```
(54, 10)
```

In [ ]:
```
df['state'].isna().sum() , df['zipcode'].isna().sum()
```

Out[ ]:
```
(0, 0)
```

In [ ]:
```
df['emp_length_yrs'] = df['emp_length'].str.extract('(\d+)')
df.drop(['emp_length'], axis=1, inplace=True)
```

In [ ]:
```
df['term'] = df['term'].str.split().str[0].astype('object')
```

In [ ]:
```python
df.sample()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	l
43629	16425.0	36	17.57	590.27	D	D2	Insurance Consultant	

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

Out[ ]: (396030, 30)

In [ ]:
```python
# List of categorical columns
cat_cols = df.select_dtypes(include='object')

# List of numerical columns
num_cols = df.select_dtypes(exclude='object')
```

In [ ]:
```python
cat_cols.sample(3)
```

Out[ ]:

	term	grade	sub_grade	emp_title	home_ownership	verification_s
297504	60	E	E3	Admin Supervisor	RENT	Ve
348683	60	C	C3	BUSINESS CONSULTANT	MORTGAGE	Ve
361643	60	G	G2	Supply Chain Manager	RENT	Source Ve

In [ ]:
```python
num_cols.sample(3)
```

Out[ ]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_re
225076	14400.0	14.09	492.79	50000.0	34.70	13.0	
313487	16000.0	16.55	393.79	70000.0	27.93	6.0	
99821	6000.0	10.49	194.99	100000.0	28.20	14.0	

In [ ]:
```python
num_cols.skew()
```

Out[ ]:
```
loan_amnt                  0.777285
int_rate                   0.420669
installment                0.983598
annual_inc                41.042725
dti                      431.051225
open_acc                   1.213019
pub_rec                    6.812303
revol_bal                 11.727515
revol_util                -0.074238
total_acc                  0.864328
mort_acc                   0.412225
pub_rec_bankruptcies      12.936099
dtype: float64
```

💡Insights

- Features are Right skewed

Action

- Need to apply log transformations in order to normalise them

---

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

In [ ]:
```
df1.sample()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title
241830	8000.0	36	9.99	258.1	B	B3	No Employee Title

🔖Q1. What percentage of customers have fully paid their Loan Amount?

In [ ]:
```
df['loan_status'].value_counts(normalize=True)*100
```

Out[ ]:
```
loan_status
Fully Paid     80.387092
Charged Off    19.612908
Name: proportion, dtype: float64
```

💡Insights:

- Target variable distribution is 80%-20%. Data is `significantly imbalanced`

📊 Graphical Analysis:

uni / bi / multi variate Analysis

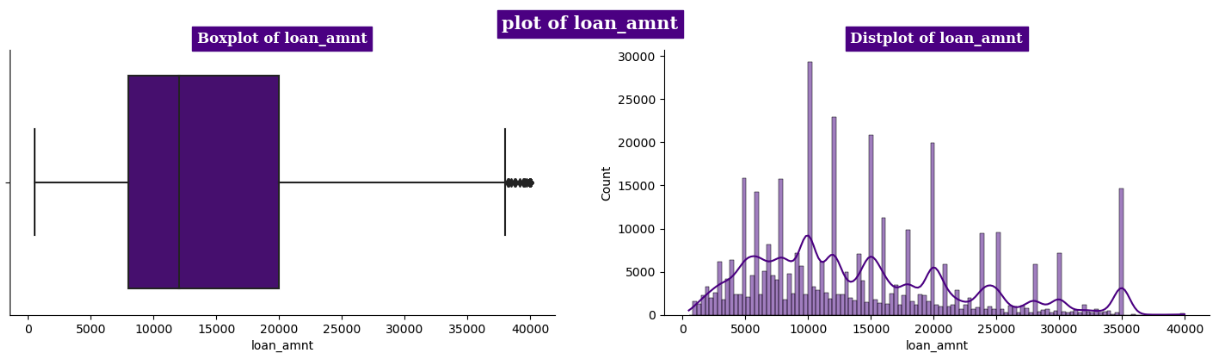
In [ ]:
```
cp = ['indigo','m','darkviolet','magenta','mediumorchid','violet','purple','
```

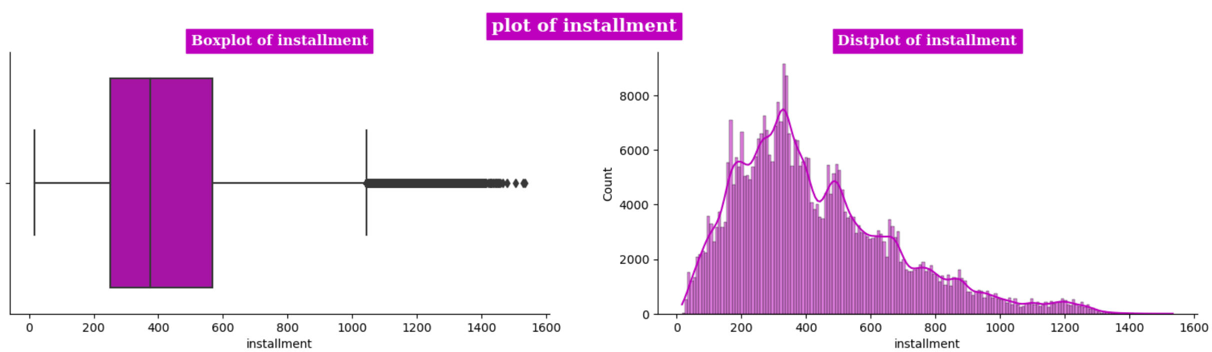
In [ ]:
```
num_cols.iloc[:,[0,2,3,4,5,6,8,9,10]].sample()
```

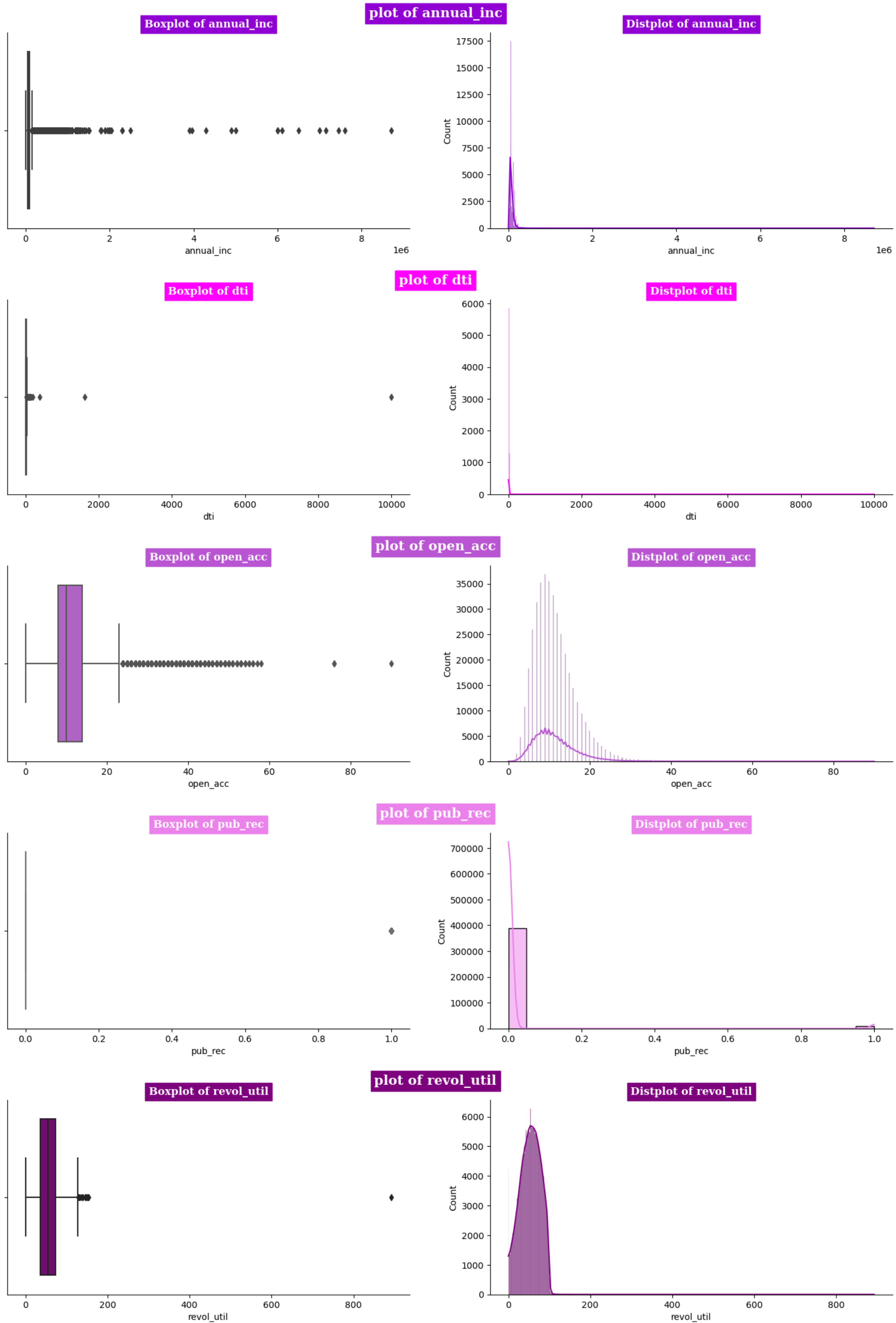
Out[ ]:

	loan_amnt	installment	annual_inc	dti	open_acc	pub_rec	revol_ut
52024	10000.0	317.54	105000.0	25.98	16.0	0	55.

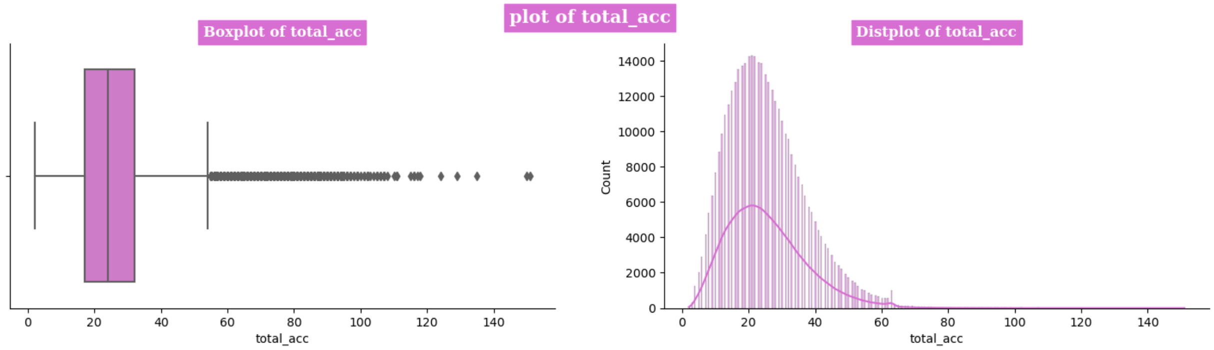
In [ ]:
```
plt.style.use('default')
plt.style.use('seaborn-bright')
outlier_graphical_cols = num_cols.iloc[:,[0,2,3,4,5,6,8,9,10]]
for _,col in enumerate(outlier_graphical_cols.columns):
    plt.figure(figsize=(18,4))
    plt.suptitle(f'plot of {col}',fontsize=15,fontfamily='serif',fontweight=
    plt.subplot(121)
    sns.boxplot(x=df[col],color=cp[_])
    plt.title(f'Boxplot of {col}',fontsize=12,fontfamily='serif',fontweight=
    plt.subplot(122)
    sns.histplot(x=df[col], kde=True,color=cp[_])
    plt.title(f'Distplot of {col}',fontsize=12,fontfamily='serif',fontweight
    sns.despine()
    plt.show()
```

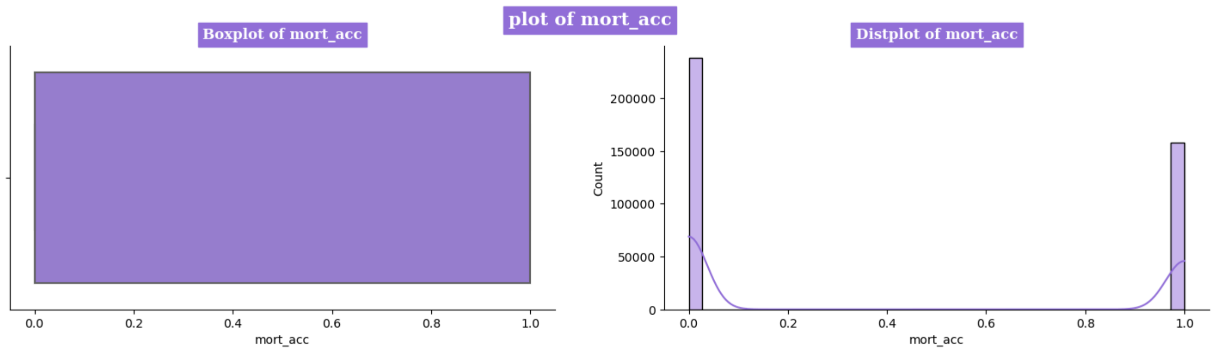






plot of annual_inc
Boxplot of annual_inc
annual_inc
1e6
Distplot of annual_inc
Count
annual_inc
1e6
plot of dti
Boxplot of dti
dti
Distplot of dti
Count
dti
plot of open_acc
Boxplot of open_acc
open_acc
Distplot of open_acc
Count
open_acc
plot of pub_rec
Boxplot of pub_rec
pub_rec
Distplot of pub_rec
Count
pub_rec
plot of revol_util
Boxplot of revol_util
revol_util
Distplot of revol_util
Count
revol_util





💡Insights:

1. The analysis suggests a prevalence of outliers, prompting further investigation into outlier detection techniques.
2. Among the numerical features, Potential outliers may still be present.
3. Notably, features such as Pub_rec, Mort_acc, and Pub_rec_bankruptcies display a sparse distribution of unique values, indicating the potential benefit of generating binary features from these variables.

In [ ]:

```
#Countplots of various categorical features w.r.t. to target variable loan_s
plt.figure(figsize=(16,17))
plt.suptitle('Countplots of various categorical features w.r.t. to target va
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolo
plt.subplot(321)
sns.countplot(data=df, x='loan_status',palette=cp)
plt.title('Loan Status Counts',fontsize=12,fontfamily='serif',fontweight='bo
plt.subplot(322)
sns.countplot(data=df, x='loan_status', hue='term',palette=cp)
plt.title('Term wise loan status count',fontsize=12,fontfamily='serif',fontw
plt.subplot(323)
sns.countplot(data=df, x='home_ownership', hue='loan_status',palette=cp)
plt.title('Loan Status Vs Home Ownership',fontsize=12,fontfamily='serif',fon
plt.subplot(324)
sns.countplot(data=df, x='verification_status', hue='loan_status',palette=cp
plt.title('Loan Status Vs Verification Status',fontsize=12,fontfamily='serif
plt.subplot(325)
sns.countplot(data=df, x='issue_month', hue='loan_status',palette=cp)
plt.title('Loan Status Vs issue_month',fontsize=12,fontfamily='serif',fontwe
plt.subplot(326)
sns.countplot(data=df, x='zipcode', hue='loan_status',palette=cp)
plt.title('Loan Status Vs zipcode',fontsize=12,fontfamily='serif',fontweight
```

```
sns.despine()
plt.show()
```

In [ ]:

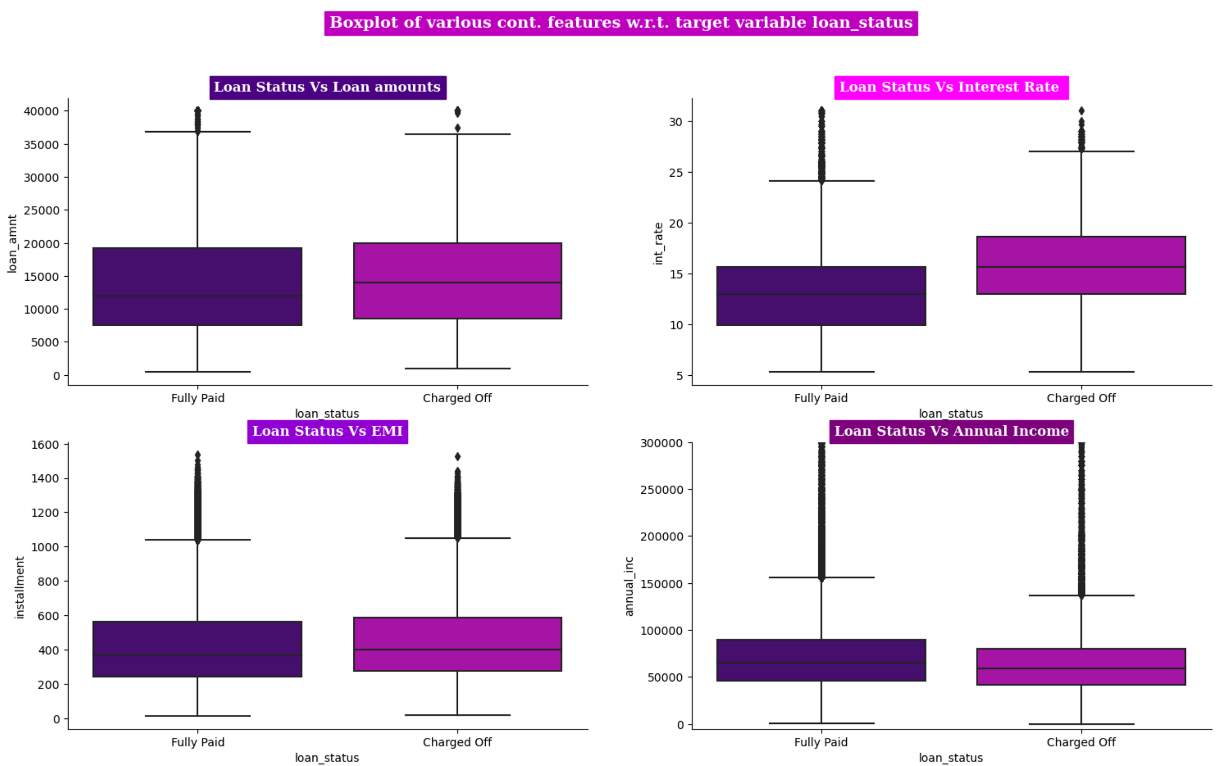
```
zip_codes = ["11650", "86630", "93700"]
states = df[df['zipcode'].isin(zip_codes)]['state']

for zip_code, state in zip(zip_codes, states):
    print(f"Zip code: {zip_code}, State: {state}")
```

```
Zip code: 11650, State: VA
Zip code: 86630, State: MI
Zip code: 93700, State: MD
```

🔍Observations:

- It's been observed that loans haven't been completely repaid in zip codes 11650, 86630, and 93700.
- Loans haven't been repaid by borrowers residing in 'VA', 'MI', and 'MD'.

In [ ]:
```
#Boxplot of various cont. features w.r.t. target variable loan_status
plt.figure(figsize=(18,10))
plt.suptitle('Boxplot of various cont. features w.r.t. target variable loan_
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolo
plt.subplot(221)
sns.boxplot(data=df, x='loan_status', y='loan_amnt',palette=cp)
plt.title('Loan Status Vs Loan amounts',fontsize=12,fontfamily='serif',fontw
plt.subplot(222)
sns.boxplot(data=df, x='loan_status', y='int_rate',palette=cp)
plt.title('Loan Status Vs Interest Rate ',fontsize=12,fontfamily='serif',for
plt.subplot(223)
sns.boxplot(data=df, x='loan_status', y='installment',palette=cp)
plt.title('Loan Status Vs EMI',fontsize=12,fontfamily='serif',fontweight='bo
plt.subplot(224)
sns.boxplot(data=df, x='loan_status', y='annual_inc',palette=cp)
plt.ylim(bottom=-5000, top=300000)
plt.title('Loan Status Vs Annual Income',fontsize=12,fontfamily='serif',font
sns.despine()
plt.show()
```



🔍Observations:

- Charged Off customers exhibit a notably higher median interest rate compared to Fully Paid customers.
- The median annual income of Charged Off customers is lower than that of Fully Paid customers.

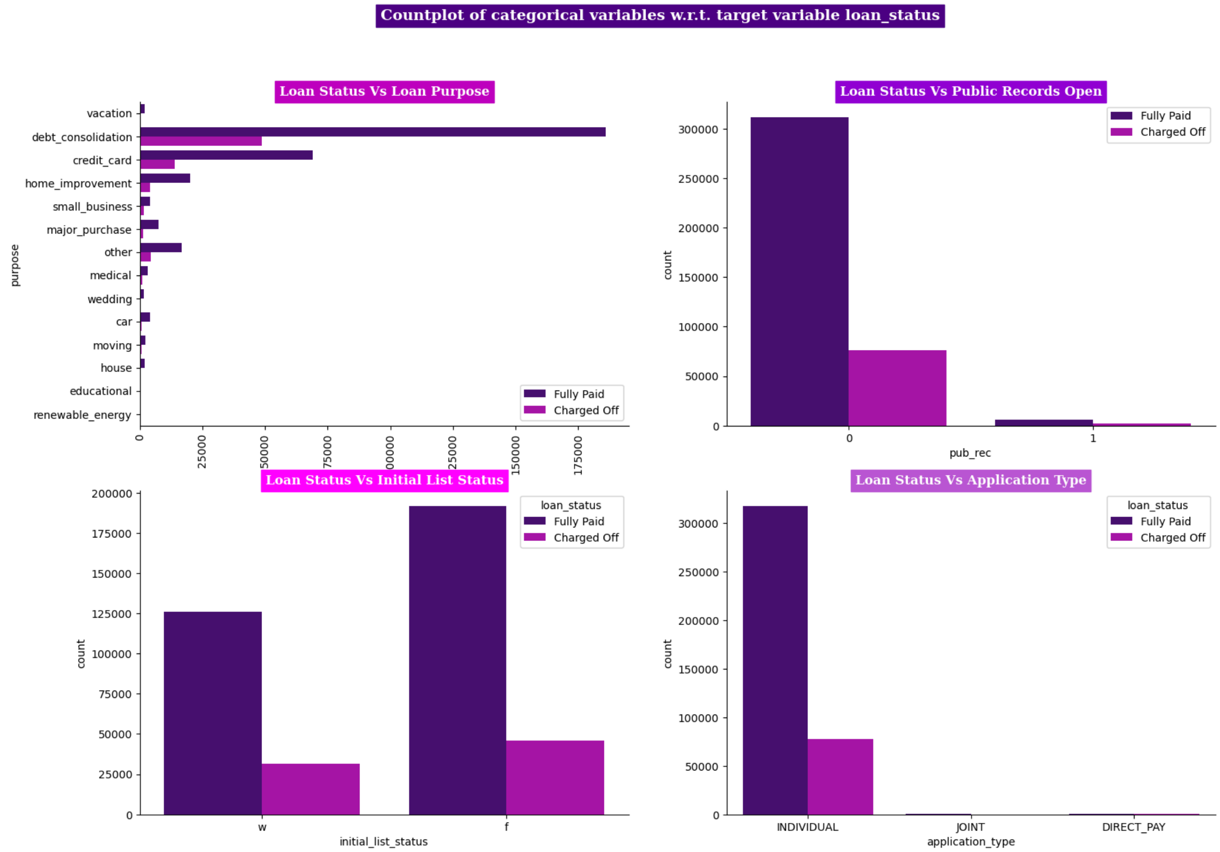
- Charged Off customers tend to have a higher median EMI compared to Fully Paid customers.
- The median loan amount for Charged Off customers surpasses that of Fully Paid customers.

In [ ]:
```
df.sample()
```

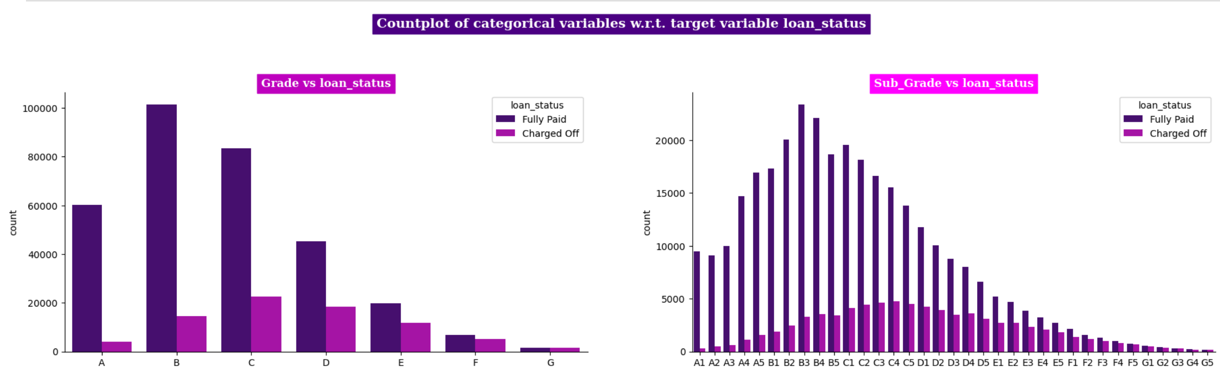
Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title
166449	25000.0	60	15.8	605.3	C	C3	Raytheon

In [ ]:
```
#Countplot of categorical variables w.r.t. target variable loan_status
plt.figure(figsize=(18,12))
plt.suptitle('Countplot of categorical variables w.r.t. target variable loar
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolc
plt.subplot(221)
sns.countplot(data=df, y='purpose', hue='loan_status',palette=cp)
plt.xticks(rotation=90)
plt.title('Loan Status Vs Loan Purpose',fontsize=12,fontfamily='serif',fontw
plt.legend(loc=4)
plt.subplot(222)
sns.countplot(data=df, x='pub_rec',hue='loan_status',palette=cp)
plt.title('Loan Status Vs Public Records Open',fontsize=12,fontfamily='serif
plt.legend(loc=1)
plt.subplot(223)
sns.countplot(data=df, x='initial_list_status', hue='loan_status',palette=cp
plt.title('Loan Status Vs Initial List Status',fontsize=12,fontfamily='serif
plt.subplot(224)
sns.countplot(data=df, x='application_type',hue='loan_status',palette=cp)
plt.title('Loan Status Vs Application Type',fontsize=12,fontfamily='serif',f
sns.despine()
plt.show()
```




```
In [ ]: plt.figure(figsize=(22,11))
        plt.suptitle('Countplot of categorical variables w.r.t. target variable loar
                     fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolc
        plt.subplot(221)
        grade = sorted(df.grade.unique().tolist())
        sns.countplot(x='grade', data=df, hue='loan_status', order=grade,palette=cp)
        plt.title('Grade vs loan_status',fontsize=12,fontfamily='serif',fontweight='
        plt.subplot(222)
        sub_grade = sorted(df.sub_grade.unique().tolist())
        sns.countplot(x='sub_grade', data=df, hue='loan_status', order=sub_grade,pal
        plt.title('Sub_Grade vs loan_status',fontsize=12,fontfamily='serif',fontweig
        sns.despine()
        plt.show()
```



🔍Observations:

- Top 2 loan purpose categories are Debit Consolidation and Credit Card

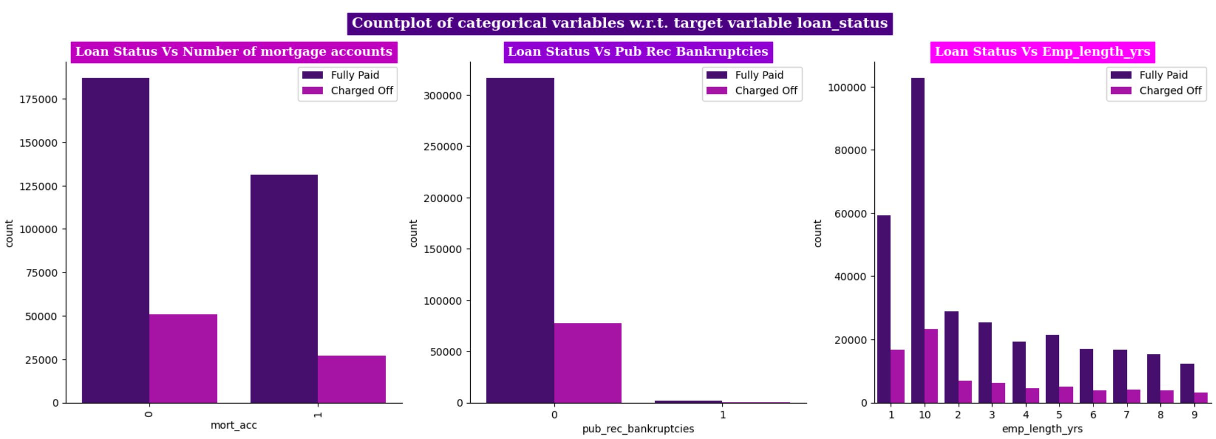
- Topmost loan type application is INDIVIDUAL
- The distribution of open_acc appears to be relatively normal when visualized graphically.
- Charged Off and Fully Paid categories exhibit similar distributions.

In [ ]:
```
df.sample()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_titl
67006	28000.0	36	14.27	960.65	C	C2	ADP Tota Souc (Specialt Manufacturin

In [ ]:
```
#Countplot for various categorical features w.r.t. target variable loan_stat

plt.figure(figsize=(20,6))
plt.suptitle('Countplot of categorical variables w.r.t. target variable loar
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolc
plt.subplot(131)
sns.countplot(data=df, x='mort_acc',hue='loan_status',palette=cp)
plt.xticks(rotation=90)
plt.title('Loan Status Vs Number of mortgage accounts',fontsize=12,fontfamil
plt.legend(loc=1)
plt.subplot(132)
sns.countplot(data=df, x='pub_rec_bankruptcies',hue='loan_status',palette=cp
plt.title('Loan Status Vs Pub Rec Bankruptcies',fontsize=12,fontfamily='seri
plt.legend(loc=1)
plt.subplot(133)
order = sorted(df.emp_length_yrs.unique().tolist())
sns.countplot(data=df, x='emp_length_yrs',hue='loan_status',order=order,pale
plt.title('Loan Status Vs Emp_length_yrs',fontsize=12,fontfamily='serif',for
plt.legend(loc=1)
sns.despine()
plt.show()
```



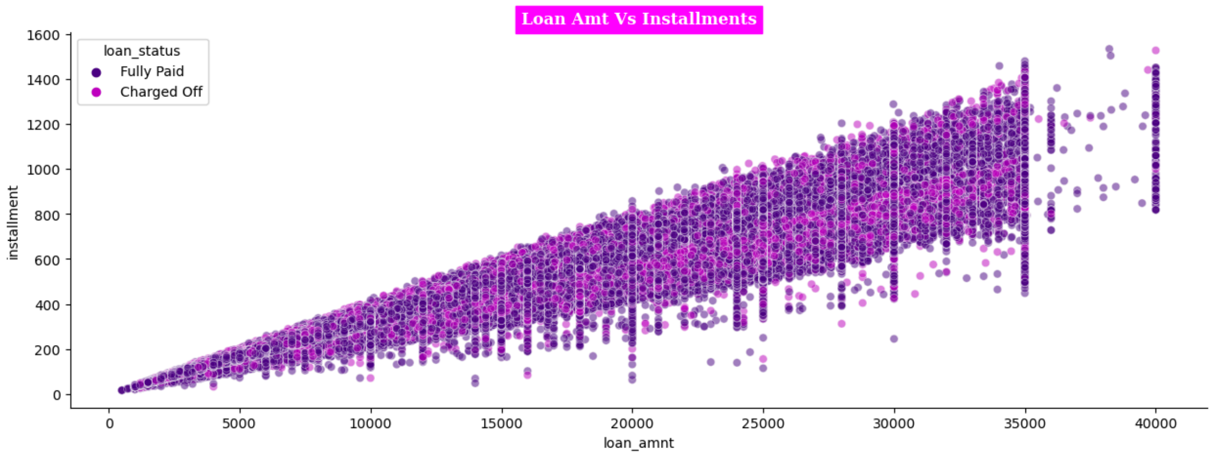
🔖Q2. Comment about the correlation between Loan Amount and Installment features.

```
In [ ]: df[['loan_amnt', 'installment']].corr()
```

Out[ ]:

	loan_amnt	installment
loan_amnt	1.000000	0.953929
installment	0.953929	1.000000

```
In [ ]: plt.figure(figsize = (15,5))
        sns.scatterplot(data = df, x = 'loan_amnt', y = 'installment', alpha = 0.5,
        plt.title('Loan Amt Vs Installments',fontsize=12,fontfamily='serif',fontweig
        sns.despine()
        plt.show()
```



💡Insights:

The correlation coefficient measures the strength and direction of the linear relationship between two variables. In this case, the correlation coefficient between 'loan_amnt' and 'installment' is quite high, approximately 0.95, indicating a strong positive linear relationship between these two variables.

- **Loan Terms**: Understanding the relationship between loan amount and installment payments is crucial for setting appropriate loan terms. Lenders can adjust loan terms such as interest rates and repayment periods based on the borrower's ability to handle installment payments associated with different loan amounts.

- **Potential Multicollinearity**: When building predictive models, it's essential to be cautious of multicollinearity between highly correlated predictor variables. Multicollinearity can lead to unstable estimates and difficulties in interpreting the model coefficients. Therefore, it might be necessary to address multicollinearity through techniques such as variable selection or regularization.

🔖Q3. The majority of people have home ownership as _______.

In [ ]:
```
(df['home_ownership'].value_counts(normalize=True)*100).to_frame()
```

Out[ ]:

home_ownership	proportion
MORTGAGE	50.084085
RENT	40.347953
OWN	9.531096
OTHER	0.028281
NONE	0.007828
ANY	0.000758

💡Insights:

- Mortgage holders comprise the majority with approximately `50.08%`, indicating that a significant portion of individuals own homes through `Mortgage` agreements.
- `Renters` constitute a substantial portion, accounting for around `40.35%` of home ownership types. This suggests a sizable demographic of individuals who opt for renting rather than owning a home.

🔖Q4. People with grades ‘A’ are more likely to fully pay their loan. (T/F)

In [ ]:
```
pd.crosstab(df['grade'],df['loan_status'], normalize = 'index')
```

Out[ ]:

grade \ loan_status	Charged Off	Fully Paid
A	0.062879	0.937121
B	0.125730	0.874270
C	0.211809	0.788191
D	0.288678	0.711322
E	0.373634	0.626366
F	0.427880	0.572120
G	0.478389	0.521611

💡Insights:

> - `True`. **Grade 'A' borrowers demonstrate a significantly high likelihood of fully repaying their loans, with approximately 93.71% of loans being fully paid**. This suggests that borrowers with the highest credit rating are more inclined to fulfill their loan obligations successfully.

- The proportion of charged-off loans for grade 'A' borrowers is relatively low, standing at approximately 6.29%. This indicates a low default rate among borrowers with the highest credit rating, emphasizing their creditworthiness and reliability in loan repayment.

🔖Q5. Name the top 2 afforded job titles.

In [ ]:
```
df[df['emp_title'] != 'No Employee Title']['emp_title'].value_counts().to_fr
```

Out[ ]:

emp_title	count
Teacher	4389
Manager	4250
Registered Nurse	1856
RN	1846
Supervisor	1830

In [ ]:
```
df.groupby('emp_title')['loan_status'].count().sort_values(ascending=False).
```

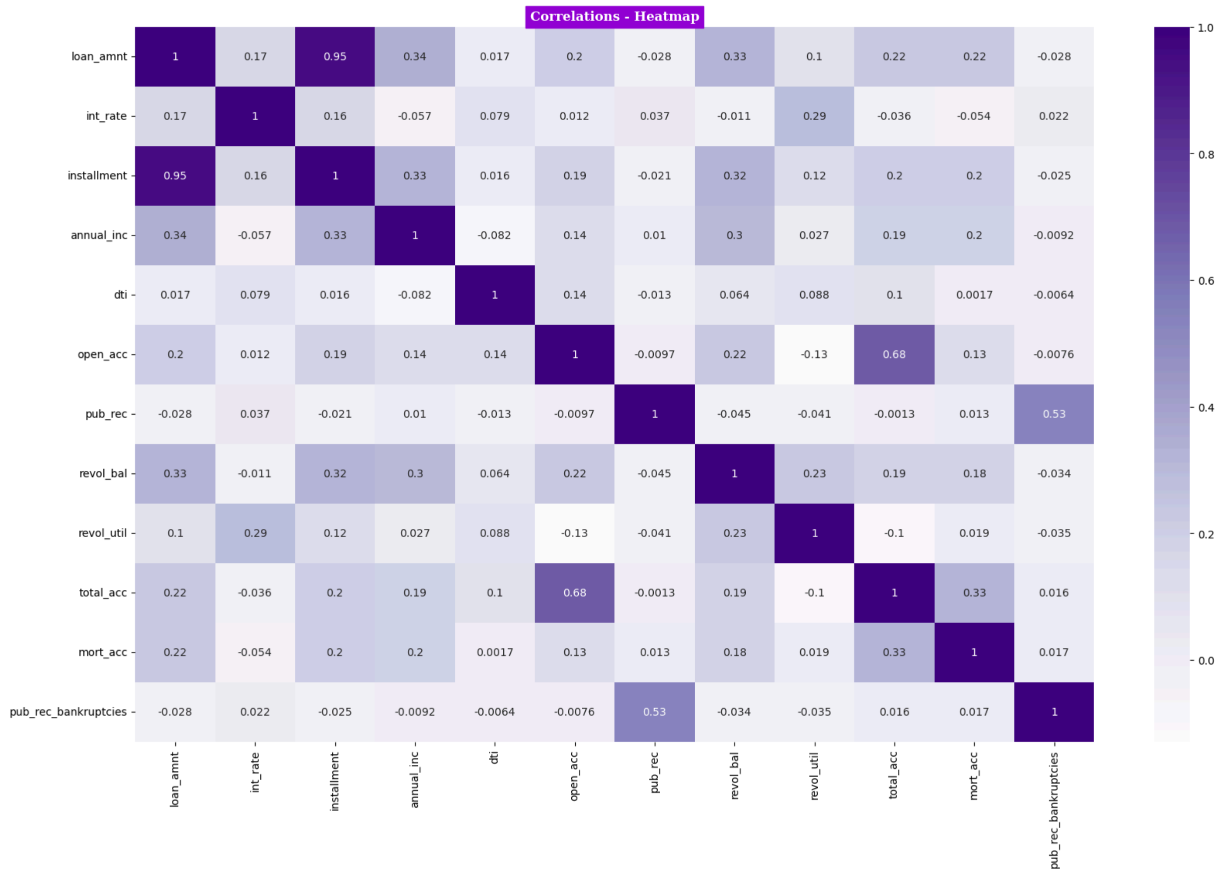
Out[ ]:

emp_title	loan_status
Teacher	4389
Manager	4250
Registered Nurse	1856
RN	1846
Supervisor	1830

💡Insights:

- The Most afforded job titles are `Teachers & Managers`.

In [ ]:
```python
plt.figure(figsize=(20,12))
sns.heatmap(num_cols.corr(), annot=True, cmap='Purples')
plt.title('Correlations - Heatmap',fontsize=12,fontfamily='serif',fontweight
plt.show()
```



💡Observations:

- There exists a strong correlation between loan_amnt and installment, indicating that higher loan amounts correspond to larger installment payments.
- The variables total_acc and open_acc exhibit a significant correlation.
- There is a notable correlation between pub_rec_bankruptcies and pub_rec.

🔍Outlier Treatment:

In [ ]:
```python
numerical_cols = df.select_dtypes(include=np.number).columns
numerical_cols
```

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

In [ ]:
```python
# outlier treatment
def remove_outliers_zscore(df, threshold=2): #(considering 2 std.dev away fr
```

```
    """
    Remove outliers from a DataFrame using the Z-score method.

    Parameters:
        df (DataFrame): The input DataFrame.
        threshold (float): The Z-score threshold for identifying outliers.
                           Observations with a Z-score greater than this thr
                           will be considered as outliers.

    Returns:
        DataFrame: The DataFrame with outliers removed.
    """
    # Calculate Z-scores for numerical columns
    z_scores = (df[numerical_cols] - df[numerical_cols].mean()) / df[numeric

    # Identify outliers
    outliers = np.abs(z_scores) > threshold

    # Keep non-outliers for numerical columns
    df_cleaned = df[~outliers.any(axis=1)]

    return df_cleaned

cleaned_df = remove_outliers_zscore(df1)
print(cleaned_df.shape)
```

```
(311392, 30)
```

In [ ]:

```
def clip_outliers_zscore(df, threshold=2):
    """
    Clip outliers in a DataFrame using the Z-score method.

    Parameters:
        df (DataFrame): The input DataFrame.
        threshold (float): The Z-score threshold for identifying outliers.
                           Observations with a Z-score greater than this thr
                           will be considered as outliers.

    Returns:
        DataFrame: The DataFrame with outliers clipped.
    """
    # Calculate Z-scores for numerical columns
    z_scores = (df[numerical_cols] - df[numerical_cols].mean()) / df[numeric

    # Clip outliers
    clipped_values = df[numerical_cols].clip(df[numerical_cols].mean() - thr
                                             df[numerical_cols].mean() + thr
                                             axis=1)

    # Assign clipped values to original DataFrame
    df_clipped = df.copy()
    df_clipped[numerical_cols] = clipped_values

    return df_clipped
```

```
clipped_df = clip_outliers_zscore(df1)
print(clipped_df.shape)
```

```
(396030, 30)
```

In [ ]:
```
data = cleaned_df.copy()
cp_data = clipped_df.copy()
data.sample()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title
110850	14000.0	36	11.67	462.8	B	B4	Manager

In [ ]:
```
data['pub_rec_bankruptcies'].value_counts() , data['pub_rec'].value_counts()
```

Out[ ]:
```
(pub_rec_bankruptcies
 0    311392
 Name: count, dtype: int64,
 pub_rec
 0    311392
 Name: count, dtype: int64)
```

In [ ]:
```
cp_data['pub_rec_bankruptcies'].value_counts() , cp_data['pub_rec'].value_co
```

Out[ ]:
```
(pub_rec_bankruptcies
 0.000000    393705
 0.158662      2325
 Name: count, dtype: int64,
 pub_rec
 0.000000    388011
 0.301947      8019
 Name: count, dtype: int64)
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(311392, 30)
```

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 311392 entries, 0 to 396029
Data columns (total 30 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             311392 non-null  float64
 1   term                  311392 non-null  object 
 2   int_rate              311392 non-null  float64
 3   installment           311392 non-null  float64
 4   grade                 311392 non-null  object 
 5   sub_grade             311392 non-null  object 
 6   emp_title             311392 non-null  object 
 7   home_ownership        311392 non-null  object 
 8   annual_inc            311392 non-null  float64
 9   verification_status   311392 non-null  object 
 10  loan_status           311392 non-null  object 
 11  purpose               311392 non-null  object 
 12  title                 311392 non-null  object 
 13  dti                   311392 non-null  float64
 14  open_acc              311392 non-null  float64
 15  pub_rec               311392 non-null  int64  
 16  revol_bal             311392 non-null  float64
 17  revol_util            311392 non-null  float64
 18  total_acc             311392 non-null  float64
 19  initial_list_status   311392 non-null  object 
 20  application_type      311392 non-null  object 
 21  mort_acc              311392 non-null  int64  
 22  pub_rec_bankruptcies  311392 non-null  int64  
 23  issue_month           311392 non-null  object 
 24  issue_year            311392 non-null  object 
 25  er_cr_line_m          311392 non-null  object 
 26  er_cr_line_y          311392 non-null  object 
 27  state                 311392 non-null  object 
 28  zipcode               311392 non-null  object 
 29  emp_length_yrs        311392 non-null  object 
dtypes: float64(9), int64(3), object(18)
memory usage: 73.6+ MB
```

Manual encoding:

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

data['initial_list_status']=data.initial_list_status.map({'w':0, 'f':1})
```

In [ ]:
```
data.head()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	hon
0	10000.0	36	11.44	329.48	B	B4	Marketing	
1	8000.0	36	11.99	265.68	B	B5	Credit analyst	
2	15600.0	36	10.49	506.97	B	B3	Statistician	
3	7200.0	36	6.49	220.65	A	A2	Client Advocate	
4	24375.0	60	17.27	609.33	C	C5	Destiny Management Inc.	

✅Feature selection - done by hypothesis testing & VIF(multicolinearity)

> Find VIF after modelling and remove features with high VIF (>5):

```
def calc_vif(X):
    # Calculating the VIF
    vif=pd.DataFrame()
    vif['Feature']=X.columns
    vif['VIF']=[variance_inflation_factor(X.values,i) for i in
range(X.shape[1])]
    vif['VIF']=round(vif['VIF'],2)
    vif=vif.sort_values(by='VIF',ascending=False)
    return vif
```

In [ ]:

```
cat_cols = data.select_dtypes(include=['object']).columns.tolist()
for col in cat_cols:
    chi2, p, dof, expected = chi2_contingency(pd.crosstab(data[col], data['l
    if p > 0.05:
        print('>>>>>>> Independent feature - Not Significant:',col,' >> p va
```

```
>>>>>>> Independent feature - Not Significant: emp_title  >> p value: 0.5367
121560200798
>>>>>>> Independent feature - Not Significant: title  >> p value: 1.0
>>>>>>> Independent feature - Not Significant: er_cr_line_m  >> p value: 0.2
722117086158036
>>>>>>> Independent feature - Not Significant: state  >> p value: 0.76047808
977373
```

In [ ]:

```
## dropping cols based on correlation(heatmap,hypothesis testing)
lt = data.drop(columns=['emp_title','title','sub_grade','er_cr_line_m','er_c
                        'state','issue_month','issue_year','pub_rec','pub_re
lt.shape
```

Out[ ]:

```
(311392, 19)
```

In [ ]:

```
lt.sample()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	home_ownership	annu
382545	25000.0	36	7.9	782.26	A	MORTGAGE	13

In [ ]:
```
#### Performing OneHotEncoding on feature having multiple variable
dummies=['zipcode', 'grade','purpose','home_ownership','verification_status'
ltd = pd.get_dummies(lt, columns=dummies, drop_first=True)*1
```

In [ ]:
```
ltd.shape
```

Out[ ]:
```
(311392, 50)
```

In [ ]:
```
ltd.dtypes
```

Out[ ]:
```
loan_amnt                              float64
term                                    object
int_rate                               float64
installment                            float64
annual_inc                             float64
loan_status                              int64
dti                                    float64
open_acc                               float64
revol_bal                              float64
revol_util                             float64
total_acc                              float64
mort_acc                                 int64
emp_length_yrs                          object
zipcode_05113                            int64
zipcode_11650                            int64
zipcode_22690                            int64
zipcode_29597                            int64
zipcode_30723                            int64
zipcode_48052                            int64
zipcode_70466                            int64
zipcode_86630                            int64
zipcode_93700                            int64
grade_B                                  int64
grade_C                                  int64
grade_D                                  int64
grade_E                                  int64
grade_F                                  int64
grade_G                                  int64
purpose_credit_card                      int64
purpose_debt_consolidation               int64
purpose_educational                      int64
purpose_home_improvement                 int64
purpose_house                            int64
purpose_major_purchase                   int64
purpose_medical                          int64
purpose_moving                           int64
purpose_other                            int64
purpose_renewable_energy                 int64
purpose_small_business                   int64
purpose_vacation                         int64
purpose_wedding                          int64
home_ownership_MORTGAGE                  int64
home_ownership_NONE                      int64
home_ownership_OTHER                     int64
home_ownership_OWN                       int64
home_ownership_RENT                      int64
verification_status_Source Verified      int64
verification_status_Verified             int64
application_type_INDIVIDUAL              int64
application_type_JOINT                   int64
dtype: object
```

In [ ]:
```
ltd.sample(8)
```

Out[ ]:

	loan_amnt	term	int_rate	installment	annual_inc	loan_status	dti
118504	15000.0	36	10.99	491.01	85000.0	0	17.05
20036	26000.0	36	11.99	863.45	100000.0	1	13.22
388815	9175.0	36	13.35	310.70	40000.0	1	15.12
388094	13000.0	60	18.24	331.82	82000.0	1	18.29
254903	9600.0	36	15.31	334.25	65000.0	1	11.78
264585	22500.0	36	15.61	786.71	81000.0	1	18.42
368842	15000.0	36	8.90	476.30	120000.0	1	9.56
44417	24000.0	60	13.99	558.32	75000.0	1	16.26

Model:

In [ ]:

```
#Prepare X and y dataset i.e. independent and dependent datasets

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

In [ ]:

```
#Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,stra
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(249113, 49)
(62279, 49)
(249113,)
(62279,)
```

Minmax scaling the data

In [ ]:

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pd.DataFrame(X_test, columns=X.columns)
```

In [ ]:

```
X_train.head()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	annual_inc	dti	open_acc	re
0	0.379538	0.0	0.339161	0.411590	0.207250	0.465341	0.368421	0.
1	0.643564	1.0	0.680070	0.524221	0.367868	0.252652	0.473684	0.
2	0.168317	0.0	0.208625	0.176198	0.134712	0.357576	0.368421	0.
3	0.379538	1.0	0.680070	0.307444	0.367868	0.449242	0.315789	0.
4	0.368812	0.0	0.543706	0.421460	0.246109	0.315530	0.263158	0.

🏷️Model-1

In [ ]:
```
#Fit the Model on training data
logreg_model = LogisticRegression()
logreg_model.fit(X_train, y_train)
```

Out[ ]:

```
▾ LogisticRegression
LogisticRegression()
```

In [ ]:
```
#Predit the data on test dataset
y_train_pred = logreg_model.predict(X_train)
y_test_pred = logreg_model.predict(X_test)
```

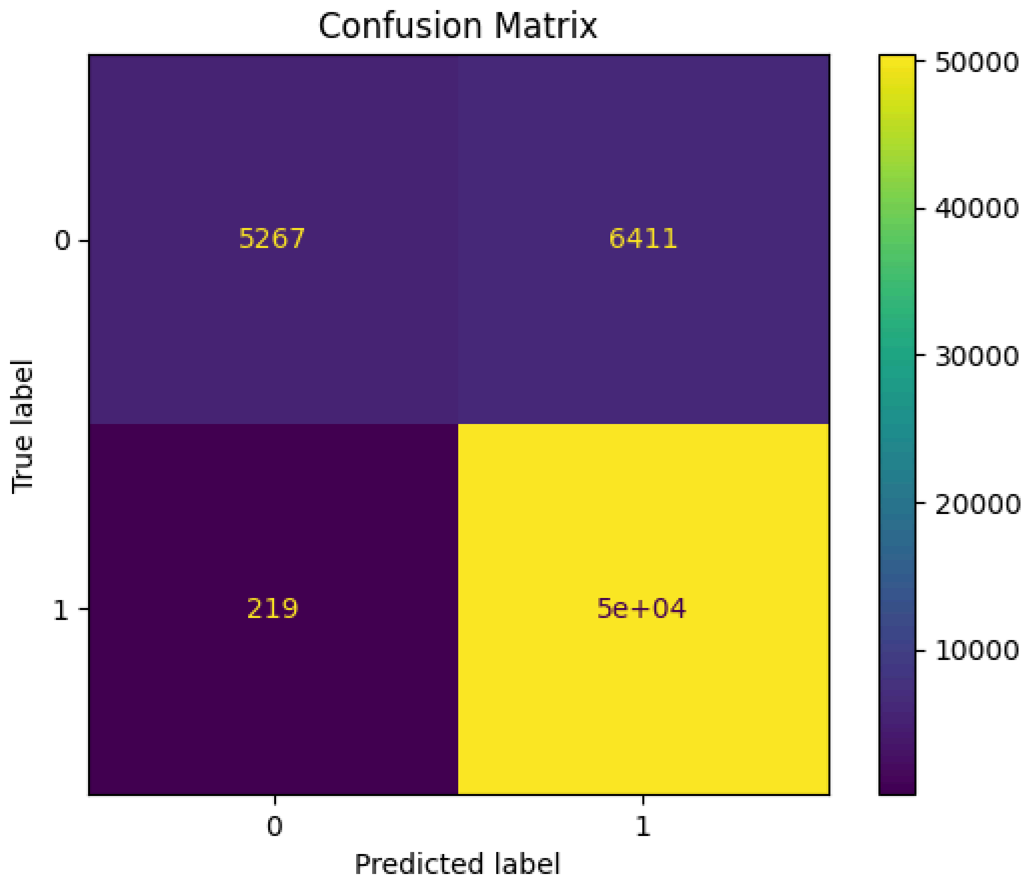
In [ ]:
```
logreg_model.score(X_test, y_test) , logreg_model.score(X_test, y_test_pred)
```

Out[ ]:
```
(0.8935435700637454, 1.0)
```

If logreg_model.score(X_test, y_test) consistently returns 1, it would imply that your model is predicting the test set perfectly, which could be a sign of overfitting, data leakage, or an issue with the evaluation process.

In [ ]:
```
#Model Evaluation
print('Train Accuracy :', logreg_model.score(X_train, y_train).round(2))
print('Train F1 Score:',f1_score(y_train,y_train_pred).round(2))
print('Train Recall Score:',recall_score(y_train,y_train_pred).round(2))
print('Train Precision Score:',precision_score(y_train,y_train_pred).round(2

print('\nTest Accuracy :',logreg_model.score(X_test,y_test).round(2))
print('Test F1 Score:',f1_score(y_test,y_test_pred).round(2))
print('Test Recall Score:',recall_score(y_test,y_test_pred).round(2))
print('Test Precision Score:',precision_score(y_test,y_test_pred).round(2))

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

```
Train Accuracy : 0.89
Train F1 Score: 0.94
Train Recall Score: 1.0
Train Precision Score: 0.89

Test Accuracy : 0.89
Test F1 Score: 0.94
Test Recall Score: 1.0
Test Precision Score: 0.89
```



In [ ]:
```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.45      0.61     11678
           1       0.89      1.00      0.94     50601

    accuracy                           0.89     62279
   macro avg       0.92      0.72      0.78     62279
weighted avg       0.90      0.89      0.88     62279
```

- Here the recall value for the 'charged off' is very low, Hence will build a better model

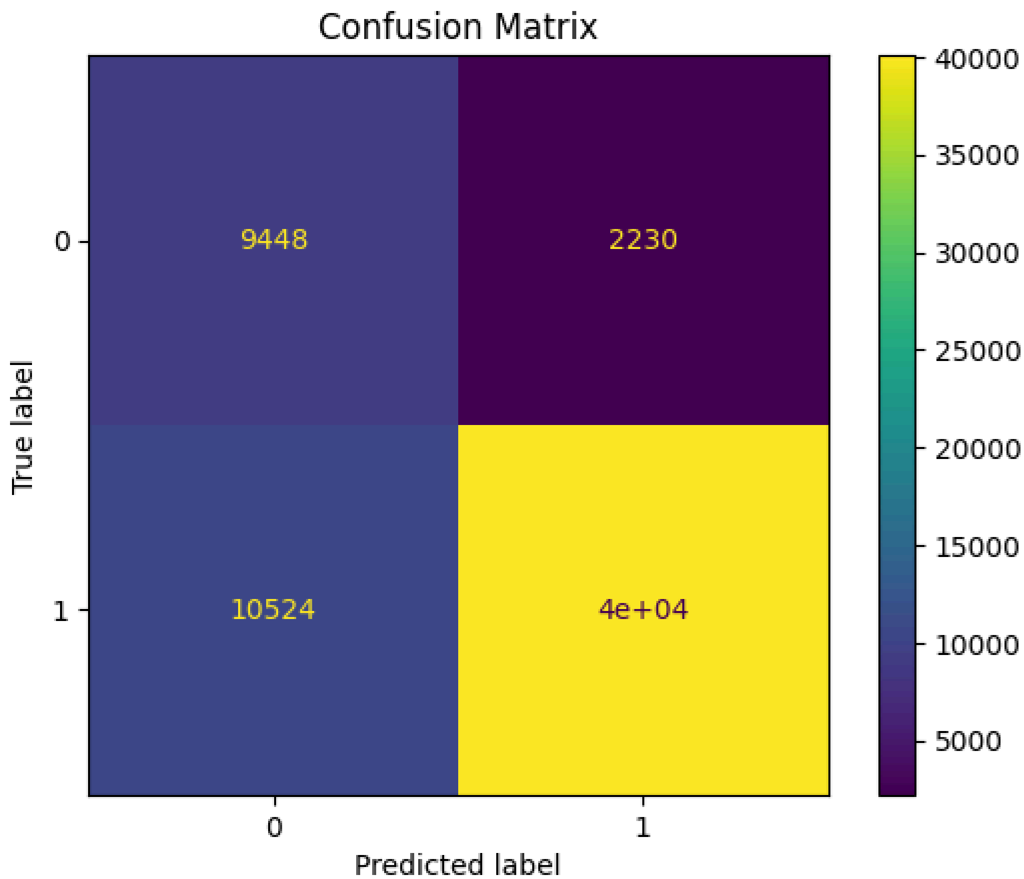
🏷️Model-2

In [ ]:
```
# 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: 202401
Before OverSampling, count of label 0: 46712
After OverSampling, count of label 1: 202401
After OverSampling, count of label 0: 202401
```

In [ ]:
```
model = LogisticRegression()
model.fit(X_train_res, y_train_res)
train_preds = model.predict(X_train)
test_preds = model.predict(X_test)

#Model Evaluation
print('Train Accuracy :', model.score(X_train, y_train).round(2))
print('Train F1 Score:',f1_score(y_train,train_preds).round(2))
print('Train Recall Score:',recall_score(y_train,train_preds).round(2))
print('Train Precision Score:',precision_score(y_train,train_preds).round(2)

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

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

```
Train Accuracy : 0.79
Train F1 Score: 0.86
Train Recall Score: 0.79
Train Precision Score: 0.95

Test Accuracy : 0.8
Test F1 Score: 0.86
Test Recall Score: 0.79
Test Precision Score: 0.95
```




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

```
              precision    recall  f1-score   support

           0       0.47      0.81      0.60     11678
           1       0.95      0.79      0.86     50601

    accuracy                           0.80     62279
   macro avg       0.71      0.80      0.73     62279
weighted avg       0.86      0.80      0.81     62279
```

🔍Observations:

- The model demonstrates a high recall score, successfully identifying 80% of actual defaulters.
- However, the precision for the positive class (defaulters) is low; only 47% of predicted defaulters are actually defaulters.
- This high recall and low precision indicate that while the model is effective at flagging most defaulters, it also results in many false positives. Consequently, many deserving customers may be denied loans.
- The low precision adversely affects the F1 score, reducing it to 60%, despite an overall accuracy of 80%. This highlights the trade-off between precision and recall in the model's performance.

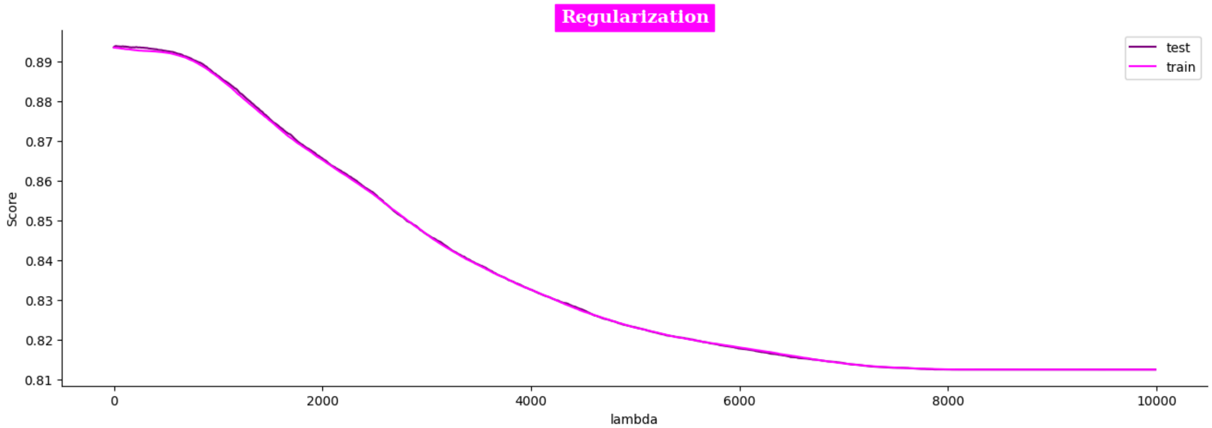
> Explanation :

- The model is good at catching most people who don't pay back their loans it catches 80% of them.
- But, when it says someone won't pay back, it's right only half of the time.47% So, there's a chance it's making mistakes and wrongly flagging people.
- Because of these mistakes, some people who deserve loans might not get them.
- Even though the model seems okay overall, its balance between being right and not making mistakes isn't great. It's like a seesaw; when one side goes up, the other goes down.

🏷️Regularization Model

In [ ]:
```
#Try with different regularization factor lamda and choose the best to buil

lamb = np.arange(0.01, 10000, 10)

train_scores = []
test_scores = []

for lam in lamb:
    model = LogisticRegression(C = 1/lam)
    model.fit(X_train, y_train)

    tr_score = model.score(X_train, y_train)
    te_score = model.score(X_test, y_test)

    train_scores.append(tr_score)
    test_scores.append(te_score)
```

In [ ]:
```
#Plot the train and test scores with respect lambda values i.e. regularizati
ran = np.arange(0.01, 10000, 10)
plt.figure(figsize=(16,5))
sns.lineplot(x=ran,y=test_scores,color='purple',label='test')
sns.lineplot(x=ran,y=train_scores,color='magenta',label='train')
plt.title('Regularization',fontsize=14,fontfamily='serif',fontweight='bold',
plt.xlabel("lambda")
plt.ylabel("Score")
sns.despine()
plt.show()
```



In [ ]:
```
#Check the index of best test score and the check the best test score

print(np.argmax(test_scores))
print(test_scores[np.argmax(test_scores)])
```

```
2
0.8939289327060486
```

In [ ]:
```
#Calculate the best lambda value based on the index of best test score

best_lamb = 0.01 + (10*2)
best_lamb
```

Out[ ]: 20.01

In [ ]:
```
#Fit the model using best lambda

reg_model = LogisticRegression(C=1/best_lamb)
reg_model.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression(C=0.04997501249375312)
```

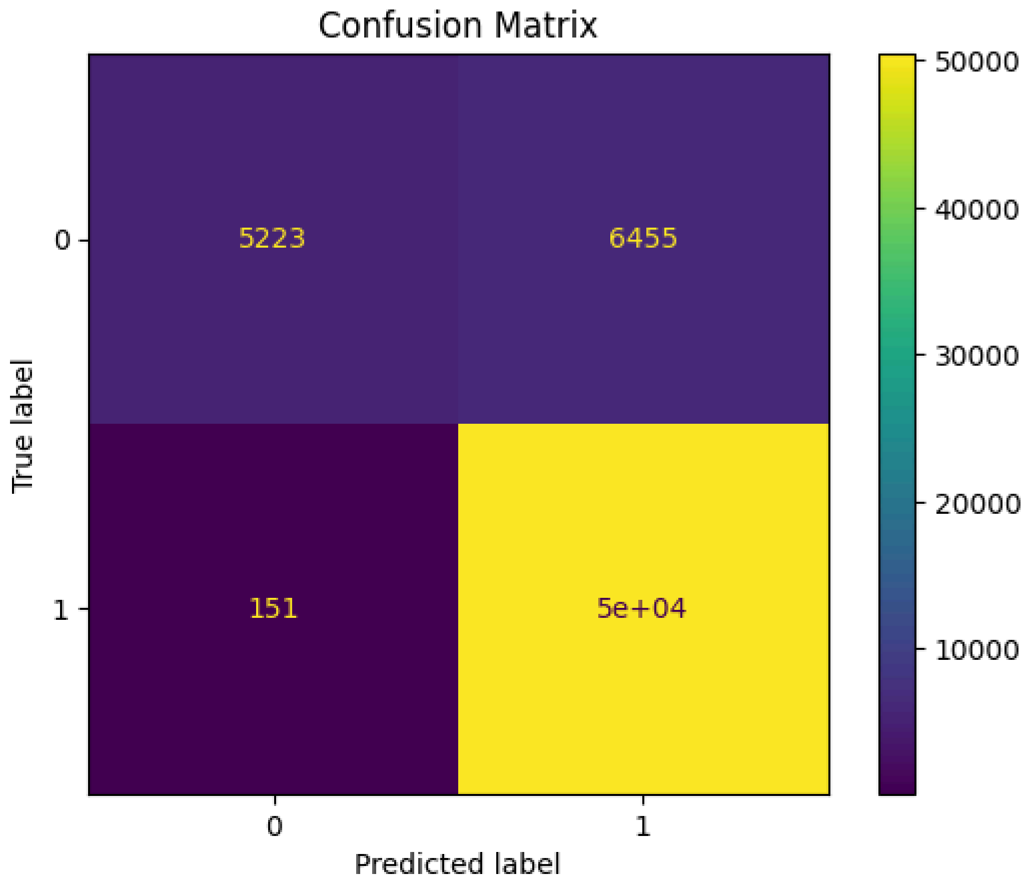
In [ ]:
```
#Predict the y_values and y_probability values

y_reg_pred = reg_model.predict(X_test)
y_reg_pred_proba = reg_model.predict_proba(X_test)
```

In [ ]:
```
#Print model score

print(f'Logistic Regression Model Score with best lambda: ',end='')
print(round(model.score(X_test, y_test)*100,2),'%')
```

```
Logistic Regression Model Score with best lambda: 89.39 %
```

In [ ]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_reg_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
```

```python
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:
```python
print(classification_report(y_test, y_reg_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.45      0.61     11678
           1       0.89      1.00      0.94     50601

    accuracy                           0.89     62279
   macro avg       0.93      0.72      0.78     62279
weighted avg       0.90      0.89      0.88     62279
```

💡Observations from classification report:

> Regularized model

- `Precision : 89%`
- `Recall : 100%`
- `F1-score : 94%`
- `Accuracy : 89%`

🔹 K-fold - Cross_validation

- cross validation accuracy has to be approx 89%

In [ ]:
```
x=scaler.fit_transform(X)

kfold = KFold(n_splits=10)
accuracy = np.mean(cross_val_score(reg_model,x,y,cv=kfold,scoring='accuracy'
print("Cross Validation accuracy : {:.3f}".format(accuracy))
```

```
Cross Validation accuracy : 0.894
```

In [ ]:
```
cm = confusion_matrix(y_test, y_reg_pred)
cm_df = pd.DataFrame(cm, index=['Defaulter','Fully paid'], columns=['Default
cm_df
```

Out[ ]:

	Defaulter	Fully paid
Defaulter	5223	6455
Fully paid	151	50450

💡Insights:

- TN = 5223 (True Negative: Correctly predicted Charged Off)
- TP = 50450 (True Positive: Correctly predicted Fully Paid)
- FP = 6455 (False Positive: Predicted Fully Paid but actually Charged Off)
- FN = 151 (False Negative: Predicted Charged Off but actually Fully Paid)
- Actual Negative (Charged Off) = 5223 + 6455 = 11678
- Actual Positive (Fully Paid) = 151 + 50450 = 50601
- Predicted Negative (Charged Off) = 5223 + 151 = 5374
- Predicted Positive (Fully Paid) = 6455 + 50450 = 56905

In [ ]:
```
#Collect the model coefficients and print those in dataframe format
coeff_df = pd.DataFrame()
coeff_df['Features'] = X_train_res.columns
coeff_df['Weights'] = model.coef_[0]
coeff_df['ABS_Weights'] = abs(coeff_df['Weights'])
coeff_df = coeff_df.sort_values(['ABS_Weights'], ascending=False)
coeff_df
```

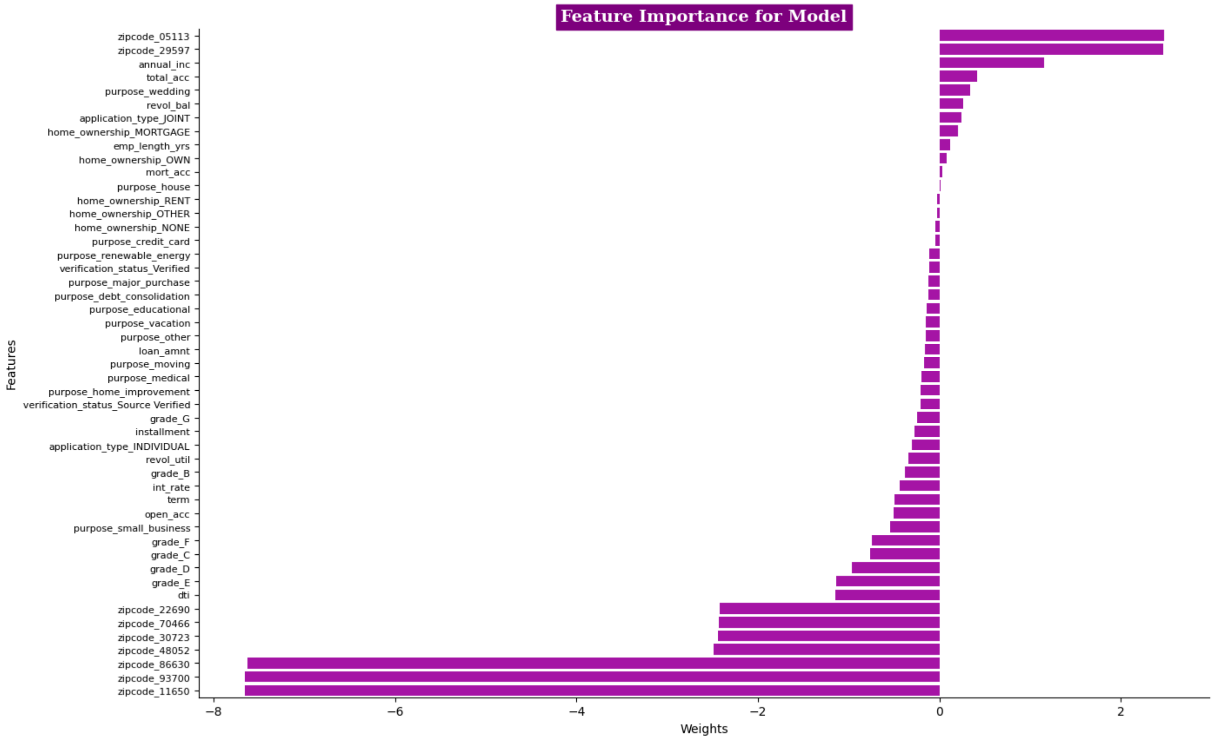
Out[ ]:

	Features	Weights	ABS_Weights
13	zipcode_11650	-7.658994	7.658994
20	zipcode_93700	-7.655336	7.655336
19	zipcode_86630	-7.631667	7.631667
17	zipcode_48052	-2.484366	2.484366
12	zipcode_05113	2.473869	2.473869
15	zipcode_29597	2.466530	2.466530
16	zipcode_30723	-2.442974	2.442974
18	zipcode_70466	-2.432947	2.432947
14	zipcode_22690	-2.425458	2.425458
4	annual_inc	1.159623	1.159623
5	dti	-1.147357	1.147357
24	grade_E	-1.134186	1.134186
23	grade_D	-0.968284	0.968284
22	grade_C	-0.764751	0.764751
25	grade_F	-0.746807	0.746807
37	purpose_small_business	-0.538707	0.538707
6	open_acc	-0.500688	0.500688
1	term	-0.492242	0.492242
2	int_rate	-0.436760	0.436760
9	total_acc	0.413223	0.413223
21	grade_B	-0.374553	0.374553
39	purpose_wedding	0.342119	0.342119
8	revol_util	-0.336643	0.336643
47	application_type_INDIVIDUAL	-0.301003	0.301003
3	installment	-0.273351	0.273351
7	revol_bal	0.260325	0.260325
26	grade_G	-0.244293	0.244293
48	application_type_JOINT	0.239988	0.239988
45	verification_status_Source Verified	-0.206324	0.206324
40	home_ownership_MORTGAGE	0.205282	0.205282
30	purpose_home_improvement	-0.202956	0.202956
33	purpose_medical	-0.193980	0.193980
34	purpose_moving	-0.166084	0.166084

	Features	Weights	ABS_Weights
0	loan_amnt	-0.158722	0.158722
35	purpose_other	-0.152204	0.152204
38	purpose_vacation	-0.149749	0.149749
29	purpose_educational	-0.143844	0.143844
11	emp_length_yrs	0.120506	0.120506
28	purpose_debt_consolidation	-0.117974	0.117974
32	purpose_major_purchase	-0.116039	0.116039
46	verification_status_Verified	-0.112949	0.112949
36	purpose_renewable_energy	-0.112775	0.112775
43	home_ownership_OWN	0.078195	0.078195
27	purpose_credit_card	-0.047232	0.047232
41	home_ownership_NONE	-0.044166	0.044166
10	mort_acc	0.032352	0.032352
42	home_ownership_OTHER	-0.028362	0.028362
44	home_ownership_RENT	-0.021251	0.021251
31	purpose_house	0.009252	0.009252

In [ ]:

```
imp_feature = coeff_df.sort_values(by='Weights',ascending=False)
plt.figure(figsize=(15,10))
sns.barplot(y = imp_feature['Features'],
            x = imp_feature['Weights'],color='m')
plt.title("Feature Importance for Model",fontsize=14,fontfamily='serif',font
plt.xlabel("Weights")
plt.yticks(fontsize=8)
plt.ylabel("Features")
sns.despine()
plt.show()
```



In [ ]:

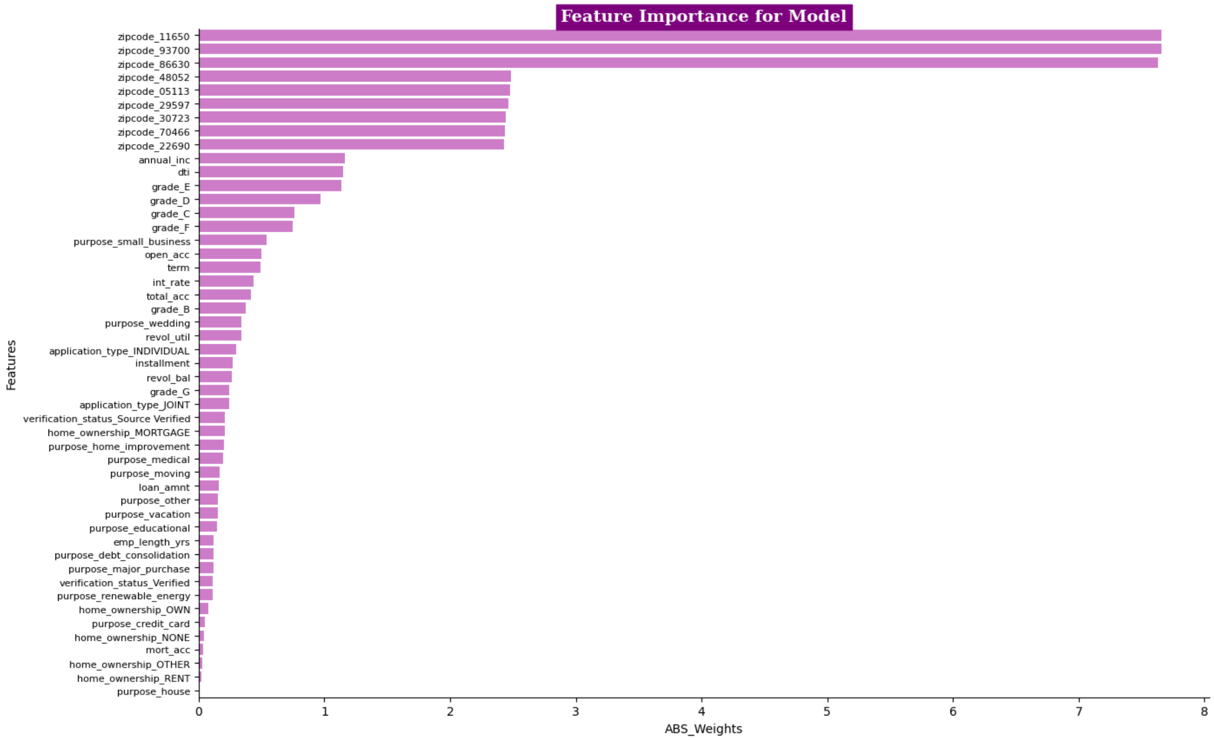
```
#Logistic Regression model intercept

model.intercept_
```

Out[ ]:

```
array([5.62531109])
```

In [ ]:

```
plt.figure(figsize=(15,10))
sns.barplot(y = coeff_df['Features'],x = coeff_df['ABS_Weights'],color='orch
plt.title("Feature Importance for Model",fontsize=14,fontfamily='serif',font
plt.xlabel("ABS_Weights")
plt.yticks(fontsize=8)
plt.ylabel("Features")
sns.despine()
plt.show()
```



🔍Observations:

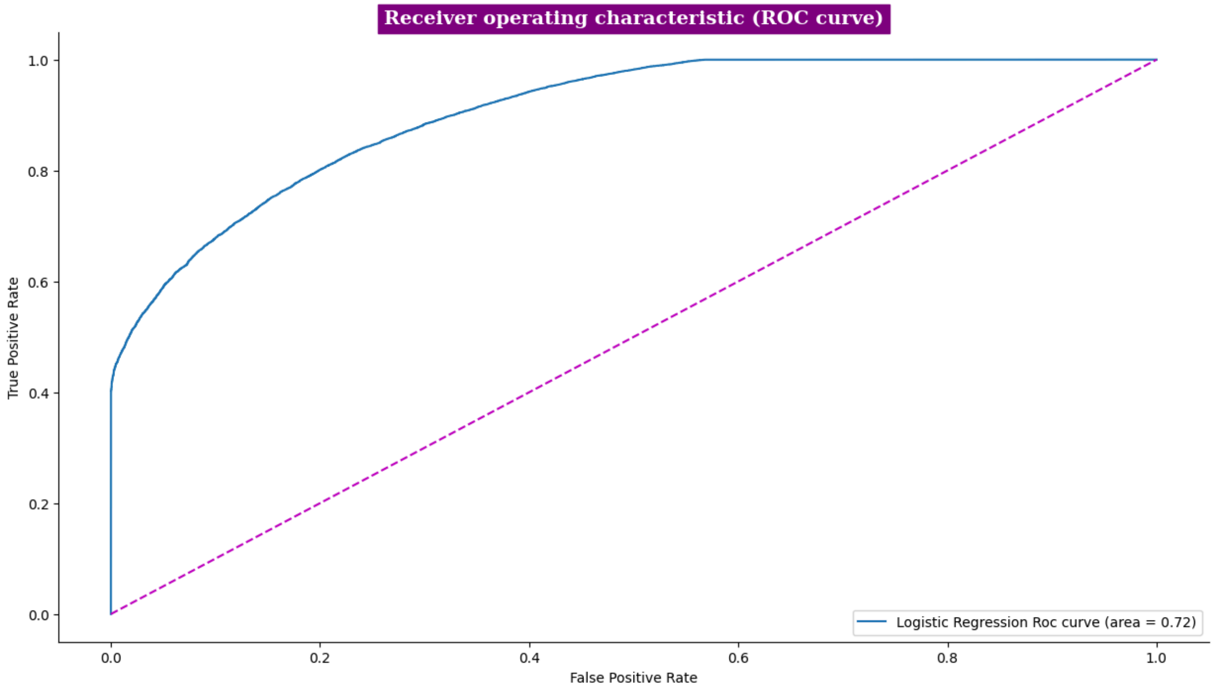
- The model has assigned significant weight to the zip_code, Annual Income, grade features, indicating that certain zip codes strongly influence the prediction of defaulters.
- Features such as dti (debt-to-income ratio), open_acc (number of open accounts), and loan_amnt (loan amount) also have high positive coefficients, highlighting their importance in predicting default risk.
- On the other hand, several zip codes have large negative coefficients, suggesting that they are associated with a lower likelihood of default.

ROC AUC curve

In [ ]:

```
# area under ROC curve
logit_roc_auc = roc_auc_score(y_test,y_reg_pred)

# Compute the false positive rate, true positive rate, and thresholds
fpr,tpr,thresholds = roc_curve(y_test,y_reg_pred_proba[:,1])

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

# plot ROC curve
plt.figure(figsize=(15,8))
plt.plot(fpr,tpr,label='Logistic Regression Roc curve (area = %0.2f)'% logit
plt.plot([0,1],[0,1],'m--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC curve)',fontsize=14,fontfa
```

```
plt.legend(loc="lower right")
sns.despine()
plt.show()
```



In [ ]:
```
logit_roc_auc
```

Out[ ]: 0.7221335554512818

In [ ]:
```
roc_auc = auc(fpr, tpr)
roc_auc
```

Out[ ]: 0.9037105453317709

💡Insights:

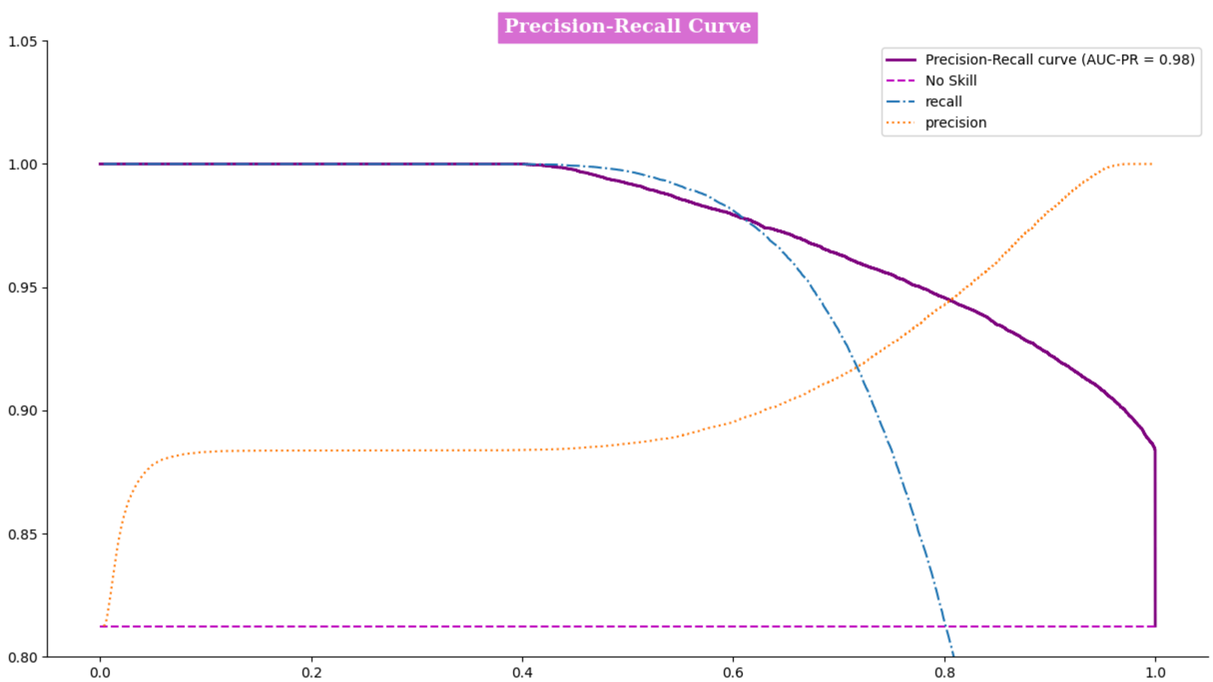
Trade-off in Performance: The ROC curve area, representing model performance, is 72%. This indicates that the model effectively distinguishes between classes 72% of the time.

- Ideally, we aim for a higher True Positive Rate (TPR) and a lower False Positive Rate (FPR) to ensure accurate predictions.
- The ROC curve illustrates that as True Positives increase, there's a simultaneous increase in False Positives.
- Misclassification: This trade-off implies that while identifying more Fully Paid customers, there's a heightened risk of misclassifying Charged Off customers as Fully Paid, potentially leading to Non-Performing Assets (NPAs).

These points emphasize the need to mitigate this risk:

- Reducing FPR while maintaining TPR is crucial to minimize misclassifications and associated risks.
- By shifting False Positives towards the left on the ROC curve, the model's overall performance, as measured by AUC, can improve.
- This improvement in AUC relies on maintaining a high True Positive Rate while reducing False Positives.

In [ ]:
```
precision, recall, thresholds = precision_recall_curve(y_test, y_reg_pred_pr

average_precision = average_precision_score(y_test, y_reg_pred_proba[:,1])

no_skill = len(y_test[y_test==1]) / len(y_test)

plt.figure(figsize=(15,8))
plt.plot(recall, precision, color='purple', lw=2, label=f'Precision-Recall c
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill', col
plt.plot(thresholds, recall[0:thresholds.shape[0]], label='recall',linestyle
plt.plot(thresholds, precision[0:thresholds.shape[0]], label='precision',lin
# plt.xlim([0.0, 1.0])
plt.ylim([0.8, 1.05])
plt.title('Precision-Recall Curve',fontsize=14,fontfamily='serif',fontweight
plt.legend(loc='upper right')
sns.despine()
plt.show()
```



In [ ]:
```
auc(recall, precision).round(3)
```

Out[ ]: 0.975

🔍Observations:

> 💡Insight:

- The Area Under the Curve (AUC) for the precision-recall curve is 0.975. This high AUC value suggests that the model achieves excellent performance in distinguishing between positive and negative classes, showcasing strong precision-recall characteristics.

- **Precision-Recall Curve Superiority**: Precision-recall curves are pivotal, especially in imbalanced datasets, focusing on accurate predictions of the relevant class (Class 1 - Fully paid in this case).

- **Irrelevance of True Negatives**: Precision and recall computations disregard true negatives, simplifying focus to the correct prediction of Fully Paid customers.

- **AUC Strengthens Model Evaluation**: A high AUC (97.5%) underscores the model's robustness in distinguishing between classes, indicating its efficacy.

- **Precision Enhancement Priority**: Optimal model refinement centers on elevating precision by minimizing False Positives, vital for improving overall performance and mitigating risks.

In [ ]:

```
# balenced Model
lr = LogisticRegression(max_iter=1000, class_weight='balanced')

lr_model = lr.fit(X_train, y_train)

print(classification_report(y_test, lr_model.predict(X_test)))

cm_bal = confusion_matrix(y_test, lr_model.predict(X_test))
cm_bal_df = pd.DataFrame(cm_bal, index=['Defaulter','Fully paid'], columns=[
cm_bal_df
```

```
              precision    recall  f1-score   support

           0       0.47      0.81      0.60     11678
           1       0.95      0.79      0.86     50601

    accuracy                           0.79     62279
   macro avg       0.71      0.80      0.73     62279
weighted avg       0.86      0.79      0.81     62279
```

Out[ ]:

	Defaulter	Fully paid
Defaulter	9466	2212
Fully paid	10573	40028

💡Observations from classification report:

> Balenced model

- `Precision : 95%`
- `Recall : 79%`
- `F1-score : 86%`
- `Accuracy : 79%`

💡Insights:

- TN = 9466 (True Negative: Correctly predicted Charged Off)
- TP = 40028 (True Positive: Correctly predicted Fully Paid)
- FP = 2212 (False Positive: Predicted Fully Paid but actually Charged Off)
- FN = 10573 (False Negative: Predicted Charged Off but actually Fully Paid)
- Actual Negative (Charged Off) = 9466 + 2212 = 11678
- Actual Positive (Fully Paid) = 10573 + 40028 = 50601
- Predicted Negative (Charged Off) = 9466 + 10573 = 20039
- Predicted Positive (Fully Paid) = 2212 + 40028 = 42240

In [ ]:
```
lr_model.intercept_
```

Out[ ]:
```
array([7.57421815])
```

🔖Q6: Thinking from a bank's perspective, which metric should our primary focus be on..

a. ROC AUC
b. Precision
c. Recall
d. F1 Score

Ans:

> From a bank's perspective, minimizing risks and maximizing profitability are paramount. `ROC AUC (Receiver Operating Characteristic Area Under Curve)` is indeed a crucial metric because it encompasses both True Positive Rate (TPR) and False Positive Rate (FPR)

- Bank's primary focus should be on ROC AUC , because bank needs to reduce FPR (False Positive Rate) and needs to increase the TPR (True Positive Rate).

- Maximizing TPR ensures that the bank correctly identifies customers who fully pay their loans (reducing False Negatives), while minimizing FPR ensures that the bank doesn't wrongly classify customers as fully paid when they're actually charged off (reducing False Positives).

- By optimizing ROC AUC, the bank can strike a balance between correctly identifying creditworthy customers and minimizing the risk of defaulters, thereby enhancing the overall performance and reliability of its credit scoring model.

Another approach:

- since I'm having `High Recall` value of 100% in Regularized model(most efficient model:

> From a bank's perspective, the primary focus should be on minimizing risks while maximizing profitability. Therefore, the most relevant metric would be **Precision**.

- Precision represents the proportion of correctly predicted positive instances (e.g., customers who fully pay their loans) out of all instances predicted as positive. In the context of a bank, precision reflects the accuracy of identifying creditworthy customers who are likely to repay their loans. Maximizing precision ensures that the bank minimizes the number of false positives, which are instances where the bank incorrectly identifies customers as creditworthy when they are not. By prioritizing precision, the bank can reduce the risk of loan defaults and associated financial losses.

- While ROC AUC, Recall, and F1 Score are also important metrics, precision aligns closely with the bank's objective of minimizing risks and ensuring the quality of its loan portfolio.

🔖Q7. How does the gap in precision and recall affect the bank?

> Ans:

- To comprehend the errors made by a model, it's crucial to evaluate both false positives and false negatives, which are gauged through metrics like recall and precision. When recall is low, it poses a significant risk for the bank.
- So, the gap between precision and recall will affect the bank. As the gap widens, there will be increase in incorrect predictions.
- Good precision means less False Positives. i.e. Less NPA loan accounts.
- Good recall means less False Negatives. i.e. not loosing on good customer.

🔖Q8. Which were the features that heavily affected the outcome?

> Ans:

- `Address(Zipcode), Annual_Income, Grade` seems to be most important feature in our case.
- Loan duration `term`, Total Credit balance `revol_bal`, : Monthly debt vs. monthly income ratio `dti`, Interest `int_rate` also has high weights(coeffients) in the model .

🔖Q9. Will the results be affected by geographical location? (Yes/No)

- Yes, we can see that zip_code (Address) is a very important feature so geographical location has impact on our result.

> Ans:

📣💰⚡Business Recommendations for LoanTap⚡💰📣

- Focus on maximizing the F1 score and area under the Precision-Recall Curve to effectively manage the precision-recall trade-off. This ensures identifying most defaulters while reducing false positives, enhancing risk management.
- Consider using more complex classifiers like Random Forests or XGBoost and perform hyperparameter tuning to enhance model performance and capture intricate relationships in the data.
- Employed stratified k-fold cross-validation to ensure representative distribution of minority class in each fold, providing reliable estimates of model performance.

> **Policy Adjustments Based on Insights**

> **Cross-Validation:**

> **Model Improvement:**

> **Optimize Loan Approval Strategy:**

- Scrutinize loans with lower grades more rigorously and consider adjusting interest rates to compensate for higher risk.
- Implement targeted strategies for high-risk zip codes, such as additional verification steps or higher interest rates.

- Evaluate small business loans with additional financial health checks and collateral requirements to mitigate default risk.

By implementing these recommendations, LoanTap can enhance their loan approval process, minimize the risk of NPAs, and ensure sustainable growth and financial stability.

In [ ]:

```

```