💰LoanTap - ML CaseStudy💰

Logistic Regression

Analysed by : **Pavan Eleti**

💳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
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 ar
home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit repo
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, exclud
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 revolvi
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
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

```
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_scor
    ConfusionMatrixDisplay, RocCurveDisplay,f1_score,recall_score,precision_score
)

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

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

```
!gdown 1sutK5BbP4CEMbnB9hqO0K75uFW754scz
```

```
Downloading...
From: https://drive.google.com/uc?id=1sutK5BbP4CEMbnB9hqO0K75uFW754scz
To: /content/loantap.csv
100% 100M/100M [00:01<00:00, 53.6MB/s]
```

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

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_l
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4
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	6
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9

5 rows × 27 columns

```
pd.set_option('display.max_columns', None)
```

🧐Exploration of data :

```
df.shape
```

```
(396030, 27)
```

```
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
```

```
df.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       '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_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

📝 Statistical Summary

```
df.describe().T
```

	count	mean	std	min	25%	50%
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	64000.00
dti	396030.0	17.379514	18.019092	0.00	11.28	16.91
open_acc	396030.0	11.311153	5.137649	0.00	8.00	10.00
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	0.00
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	11181.00
revol_util	395754.0	53.791749	24.452193	0.00	35.80	54.80
total_acc	396030.0	25.414744	11.886991	2.00	17.00	24.00
mort_acc	358235.0	1.813991	2.147930	0.00	0.00	1.00

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

	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

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

loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_leng

Insights

- The dataset does not contain any duplicates.

❓ Null Detection

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

```
emp_title               True
emp_length              True
title                   True
revol_util              True
mort_acc                True
```

```
pub_rec_bankruptcies    True
dtype: bool
```

```
df.isna().sum().sort_values(ascending=False)
```

```
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
```

```
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_values(as
    missing_data_df = pd.concat([total_missing_df, percent_missing_df], axis=1, k
    return missing_data_df

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

	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

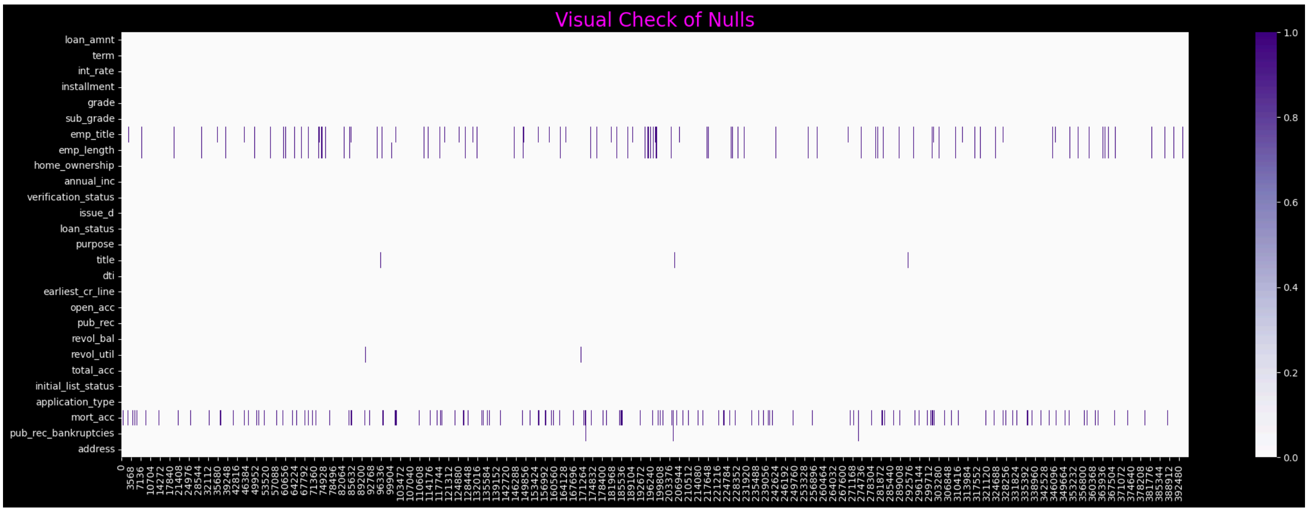
Insight

> Following columns has missing values
>
> 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

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

```
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()
```




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

```
81590
```

```
#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.
```

🔴Null Treatment:

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

```
df.isna().sum()
```

```
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
```

```
df.describe().T
```

	count	mean	std	min	25%	50%
loan_amnt	396030.0	14113.888089	8357.441341	500.00	8000.00	12000.00
int_rate	396030.0	13.639400	4.472157	5.32	10.49	13.33
installment	396030.0	431.849698	250.727790	16.08	250.33	375.43
annual_inc	396030.0	74203.175798	61637.621158	0.00	45000.00	64000.00
dti	396030.0	17.379514	18.019092	0.00	11.28	16.91
open_acc	396030.0	11.311153	5.137649	0.00	8.00	10.00
pub_rec	396030.0	0.178191	0.530671	0.00	0.00	0.00
revol_bal	396030.0	15844.539853	20591.836109	0.00	6025.00	11181.00
revol_util	396030.0	53.754260	24.484857	0.00	35.80	54.80
total_acc	396030.0	25.414744	11.886991	2.00	17.00	24.00
mort_acc	396030.0	1.640873	2.111249	0.00	0.00	1.00

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

	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

```
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_bankruptcies
```

```
df.sample()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title
60136	35000.0	36 months	12.29	1167.36	C	C1	Superintendent of Maintenance

```
#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)
```

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

```
df['address']
```

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

```
#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)
```

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

```
(54, 10)
```

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

```
(0, 0)
```

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

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

```
df.sample()
```

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

```
df.shape
```

```
(396030, 30)
```

```
# List of categorical columns
cat_cols = df.select_dtypes(include='object')

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

```
cat_cols.sample(3)
```

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

```
num_cols.sample(3)
```

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec
225076	14400.0	14.09	492.79	50000.0	34.70	13.0	0
313487	16000.0	16.55	393.79	70000.0	27.93	6.0	0

```
num_cols.skew()
```

```
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

---

```
df1 = df.copy()
```

```
df1.sample()
```

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

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

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

```
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:

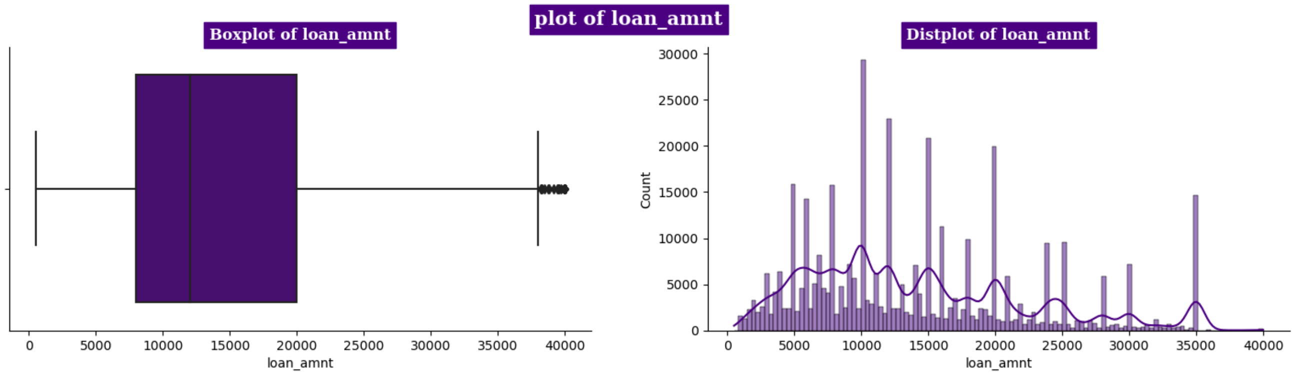
uni / bi / multi variate Analysis

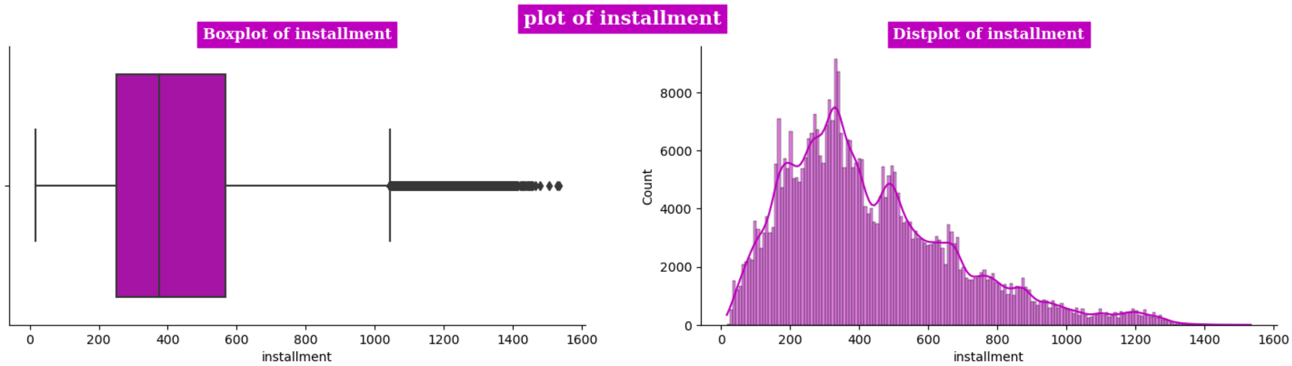
```
cp = ['indigo','m','darkviolet','magenta','mediumorchid','violet','purple','orchi
```

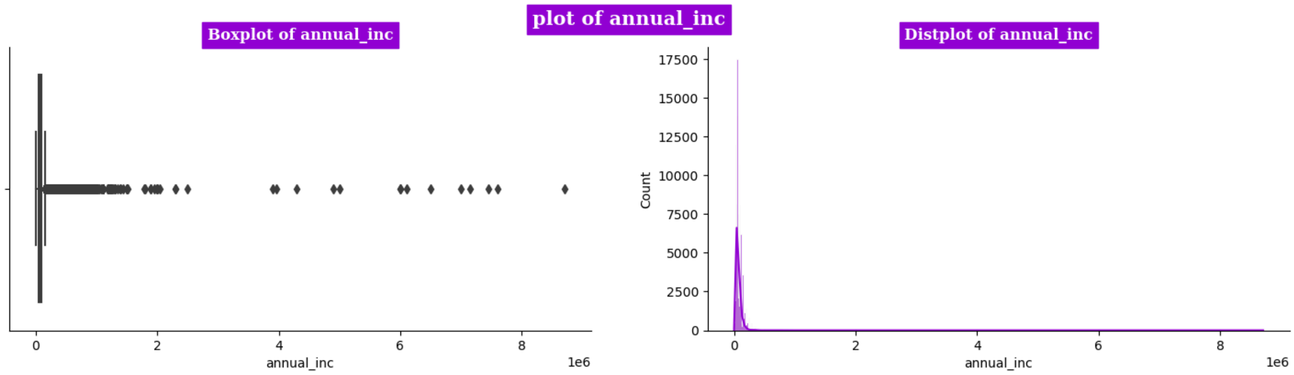
```
num_cols.iloc[:,[0,2,3,4,5,6,8,9,10]].sample()
```

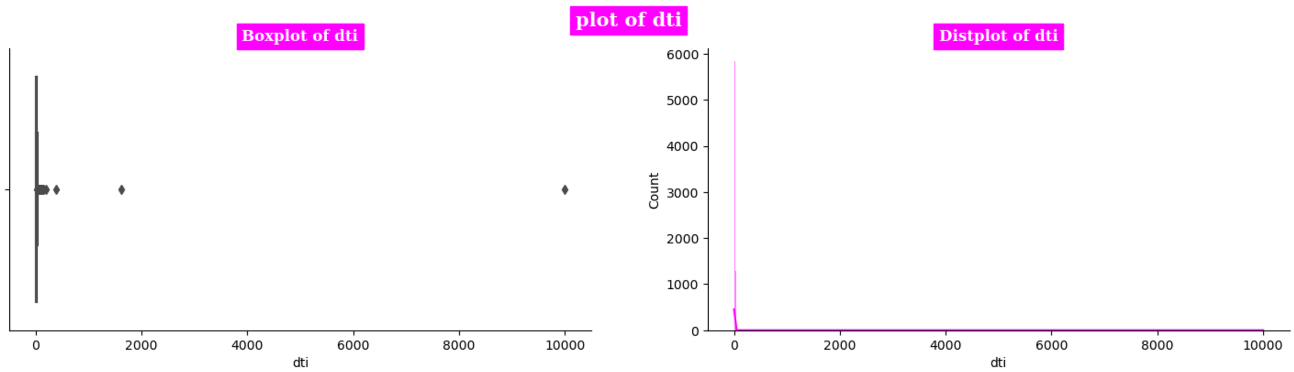
	loan_amnt	installment	annual_inc	dti	open_acc	pub_rec	revol_util
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

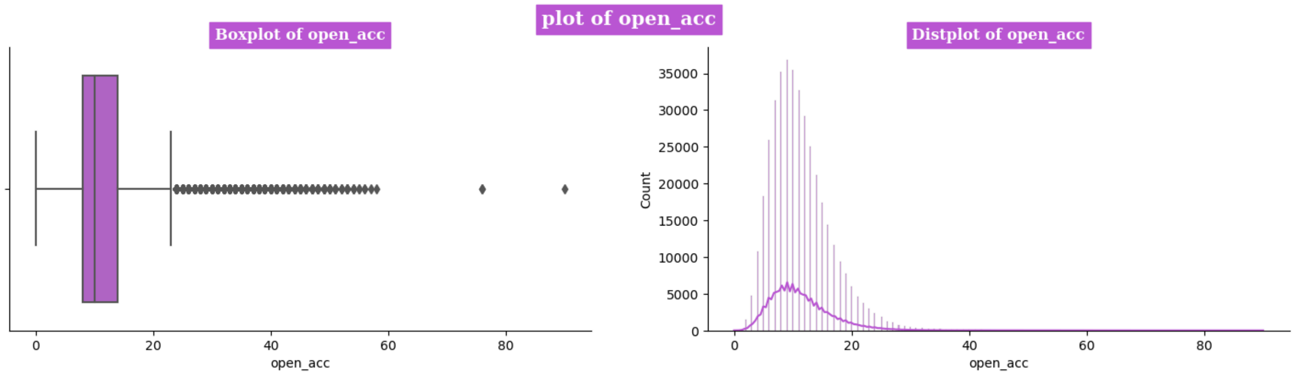
```
plt.style.use('default')
plt.style.use('seaborn-bright')
outlier_graphical_cols = num_cols.iloc[:,[0,2,3,4,5,6,8,9,10]]
for _,col in enumerate(outlier_graphical_cols.columns):
    plt.figure(figsize=(18,4))
    plt.suptitle(f'plot of {col}',fontsize=15,fontfamily='serif',fontweight='bold
    plt.subplot(121)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

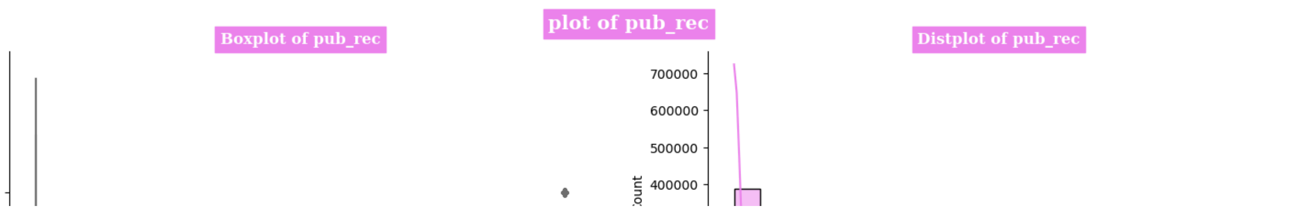

plot of loan_amnt
Boxplot of loan_amnt
loan_amnt
Distplot of loan_amnt
Count
loan_amnt

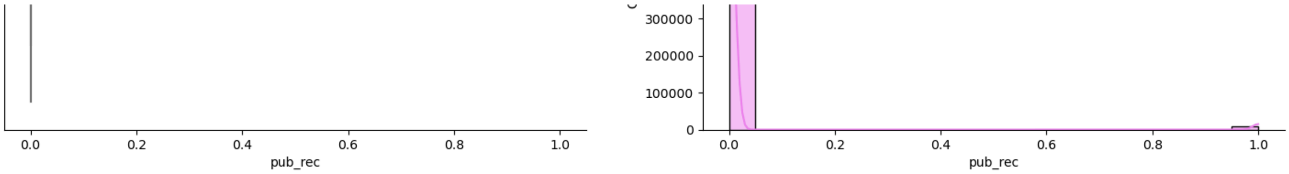

plot of installment
Boxplot of installment
installment
Distplot of installment
Count
installment

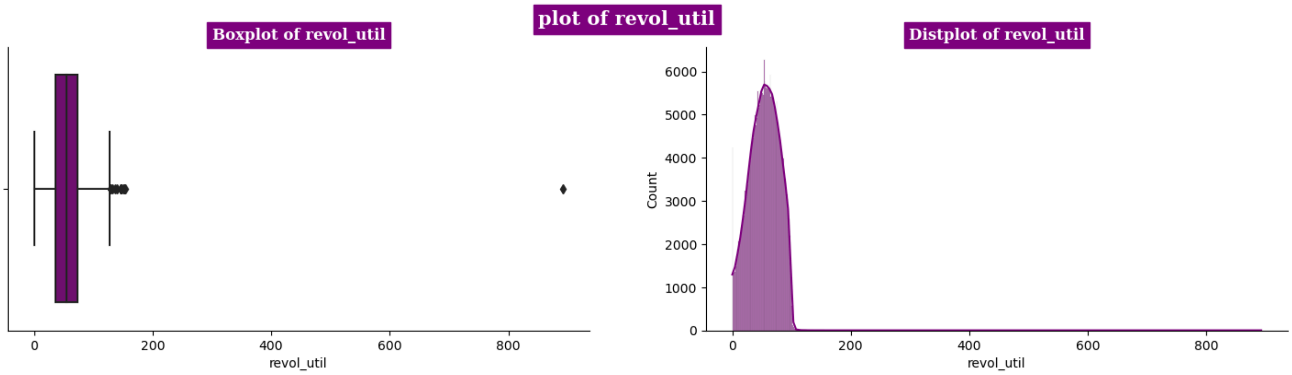

plot of annual_inc
Boxplot of annual_inc
annual_inc
1e6
Distplot of annual_inc
Count
annual_inc
1e6

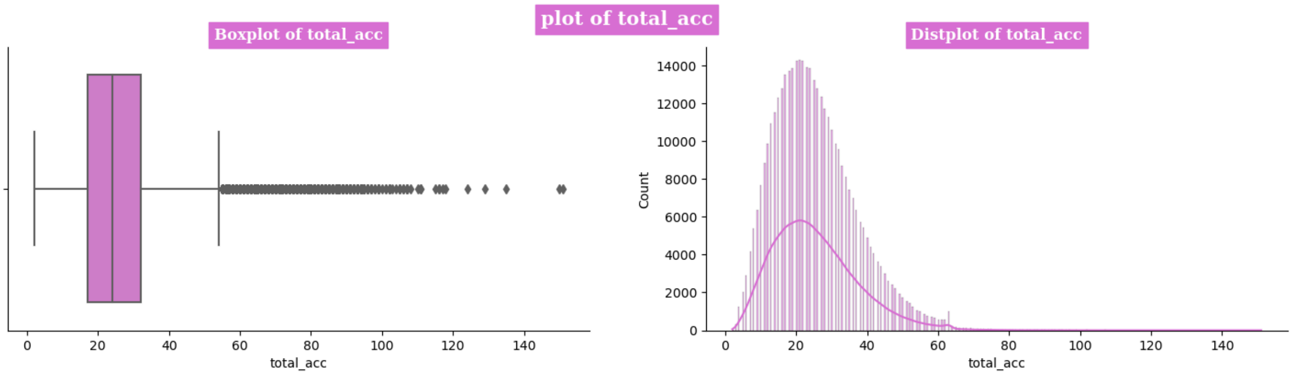

plot of dti
Boxplot of dti
dti
Distplot of dti
Count
dti

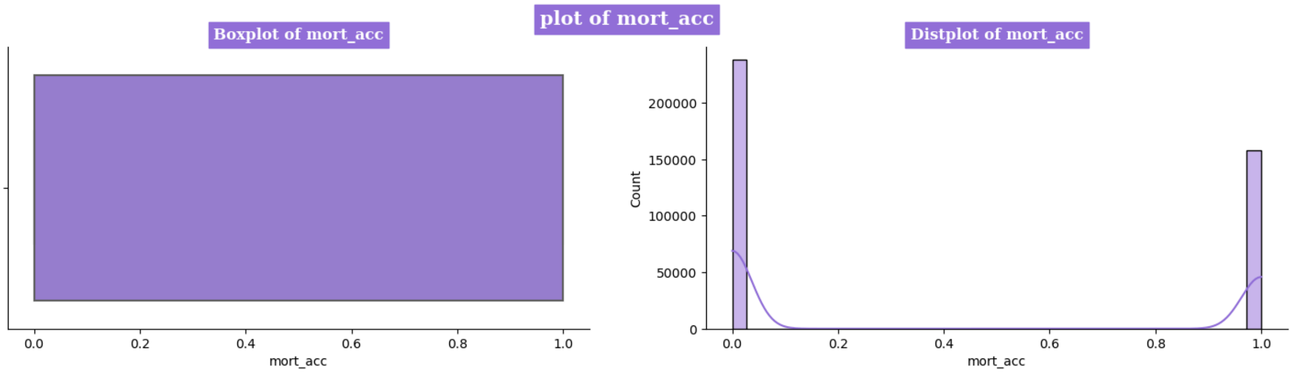

plot of open_acc
Boxplot of open_acc
open_acc
Distplot of open_acc
Count
open_acc


plot of pub_rec
Boxplot of pub_rec
Distplot of pub_rec


0.0
0.2
0.4
0.6
0.8
1.0
pub_rec
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec


plot of revol_util
Boxplot of revol_util
0
200
400
600
800
revol_util
Distplot of revol_util
Count
6000
5000
4000
3000
2000
1000
0
0
200
400
600
800
revol_util


plot of total_acc
Boxplot of total_acc
0
20
40
60
80
100
120
140
total_acc
Distplot of total_acc
Count
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140
total_acc


plot of mort_acc
Boxplot of mort_acc
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc
Distplot of mort_acc
Count
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc

💡 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.

```
#Countplots of various categorical features w.r.t. to target variable loan_status
plt.figure(figsize=(16,17))
plt.suptitle('Countplots of various categorical features w.r.t. to target variabl
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor=cp[
plt.subplot(321)
sns.countplot(data=df, x='loan_status',palette=cp)
plt.title('Loan Status Counts',fontsize=12,fontfamily='serif',fontweight='bold',b
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',fontweight
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',fontweig
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',fon
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',fontweight=
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='bol
sns.despine()
plt.show()
```


Countplots of various categorical features w.r.t. to target variable loan_status
Loan Status Counts
count
loan_status
Fully Paid
Charged Off
Term wise loan status count
term
36
60
Loan Status Vs Home Ownership
home_ownership
RENT
MORTGAGE
OWN
OTHER
NONE
ANY
Loan Status Vs Verification Status
verification_status
Not Verified
Source Verified
Verified
Loan Status Vs issue_month
issue_month
Jan
Nov
Apr
Sep
Oct
Jun
May
Dec
Jul
Feb
Mar
Aug
Loan Status Vs zipcode
zipcode
22690
05113
00813
11650
30723
70466
29597
48052
86630
93700

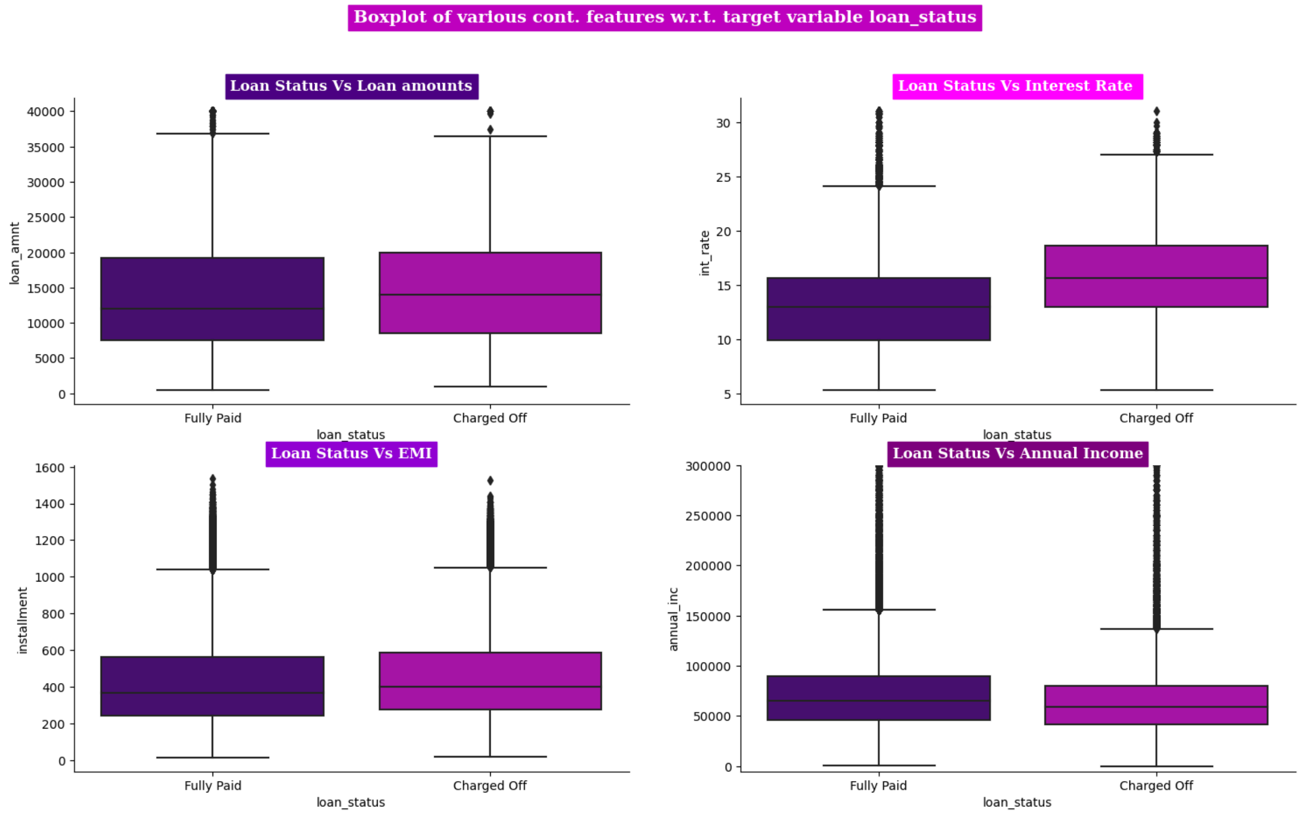
```
zip_codes = ["11650", "86630", "93700"]
states = df[df['zipcode'].isin(zip_codes)]['state']

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'.

```
#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_statu
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor=cp[
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',fontweight
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',fontweig
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='bold',b
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',fontweigh
sns.despine()
plt.show()
```



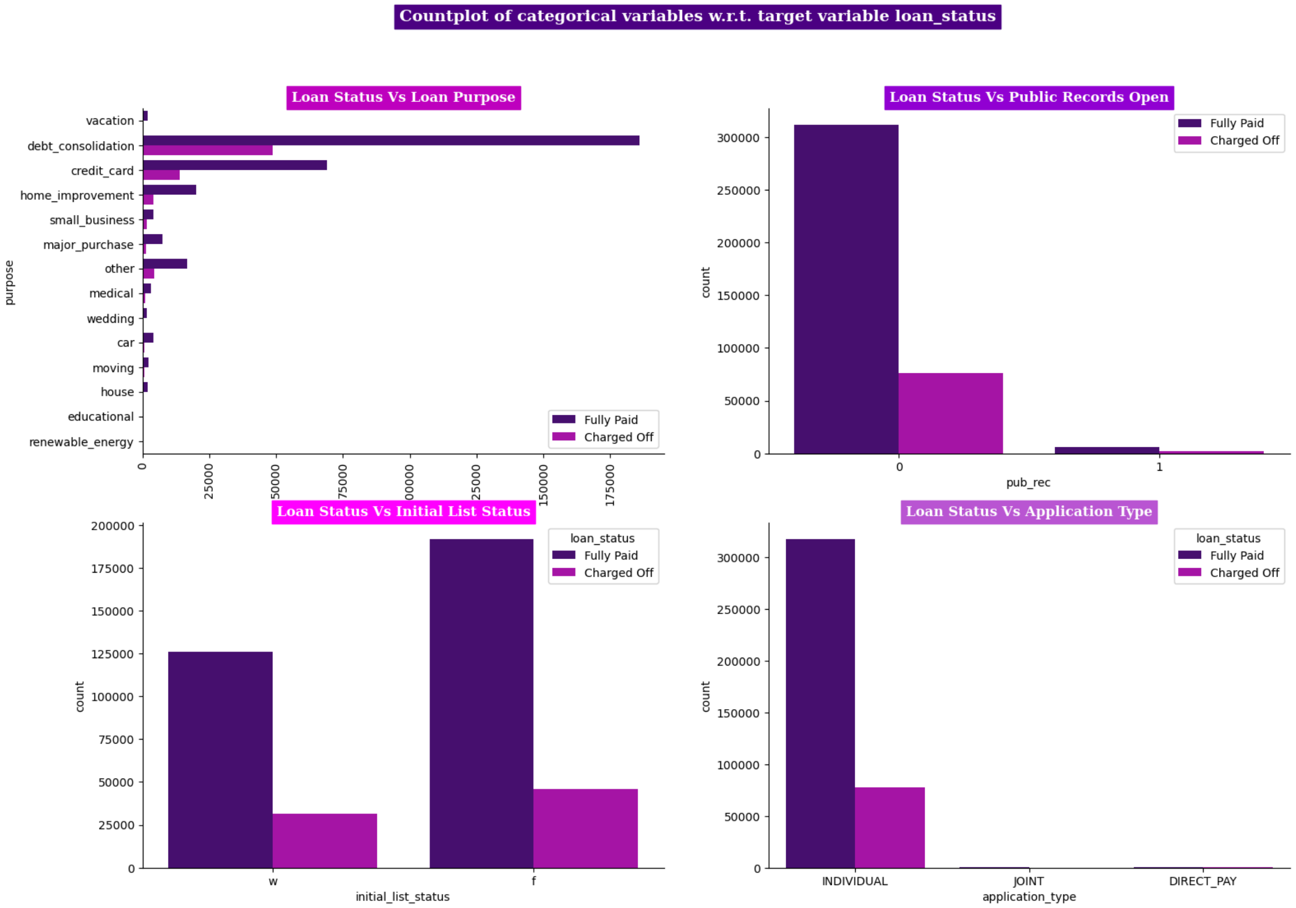
🔍Observations:

- 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.

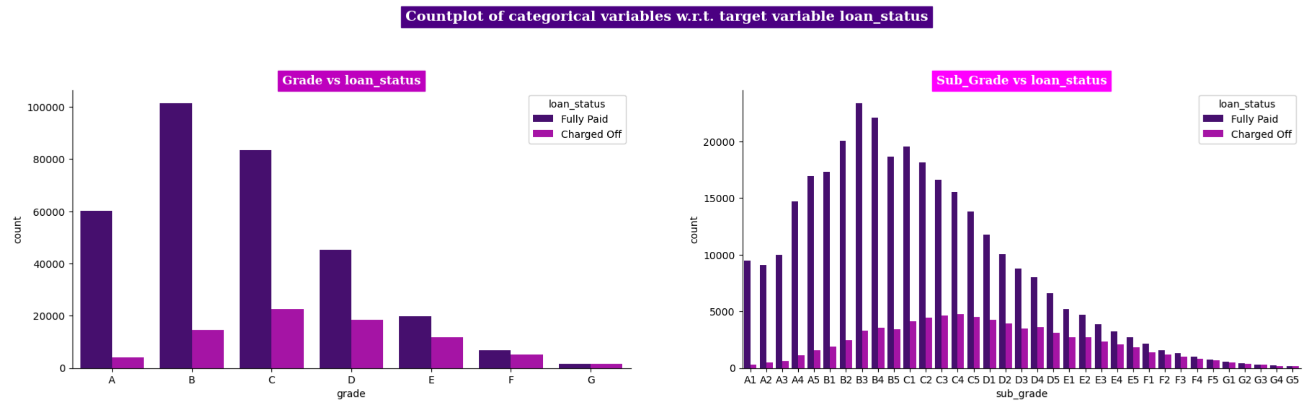
```
df.sample()
```

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

```
#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 loan_stat
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor=cp[
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',fontweight
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',fon
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',fon
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',fontwe
sns.despine()
plt.show()
```


Countplot of categorical variables w.r.t. target variable loan_status
Loan Status Vs Loan Purpose
vacation
debt_consolidation
credit_card
home_improvement
small_business
major_purchase
other
medical
wedding
car
moving
house
educational
renewable_energy
purpose
Fully Paid
Charged Off
Loan Status Vs Public Records Open
count
pub_rec
Loan Status Vs Initial List Status
loan_status
initial_list_status
w
f
Loan Status Vs Application Type
INDIVIDUAL
JOINT
DIRECT_PAY
application_type

```
plt.figure(figsize=(22,11))
plt.suptitle('Countplot of categorical variables w.r.t. target variable loan_stat
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor=cp[
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='bold'
plt.subplot(222)
sub_grade = sorted(df.sub_grade.unique().tolist())
sns.countplot(x='sub_grade', data=df, hue='loan_status', order=sub_grade,palette=
plt.title('Sub_Grade vs loan_status',fontsize=12,fontfamily='serif',fontweight='b
sns.despine()
plt.show()
```



🔍Observations:

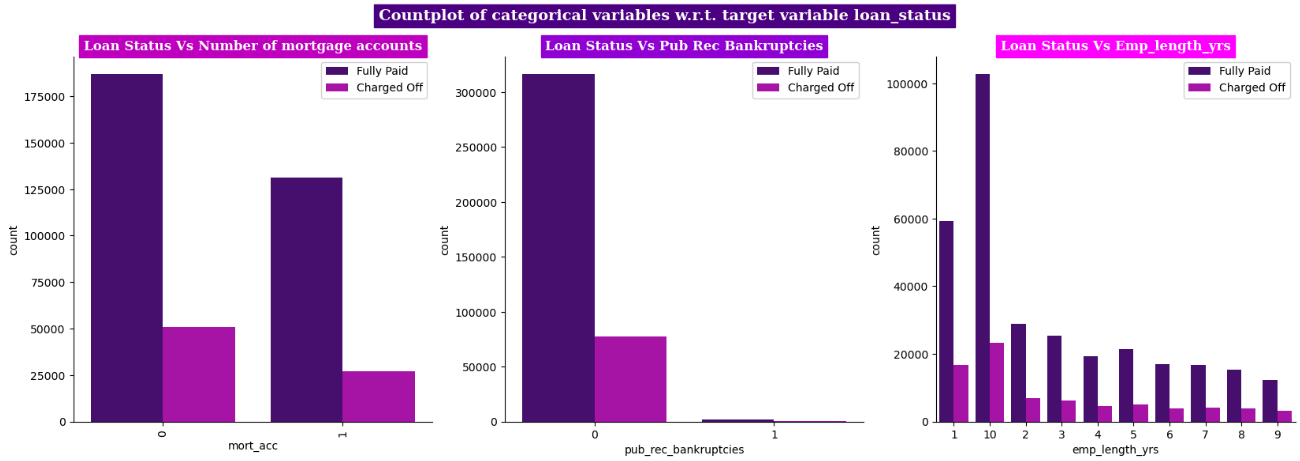
- Top 2 loan purpose categories are Debit Consolidation and Credit Card

- 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.

```
df.sample()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	ho
67006	28000.0	36	14.27	960.65	C	C2	ADP Total Souce (Specialty Manufacturing	

```
#Countplot for various categorical features w.r.t. target variable loan_status

plt.figure(figsize=(20,6))
plt.suptitle('Countplot of categorical variables w.r.t. target variable loan_stat
             fontsize=14,fontfamily='serif',fontweight='bold',backgroundcolor=cp[
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,fontfamily='se
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='serif',fo
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,palette=c
plt.title('Loan Status Vs Emp_length_yrs',fontsize=12,fontfamily='serif',fontweig
plt.legend(loc=1)
sns.despine()
plt.show()
```

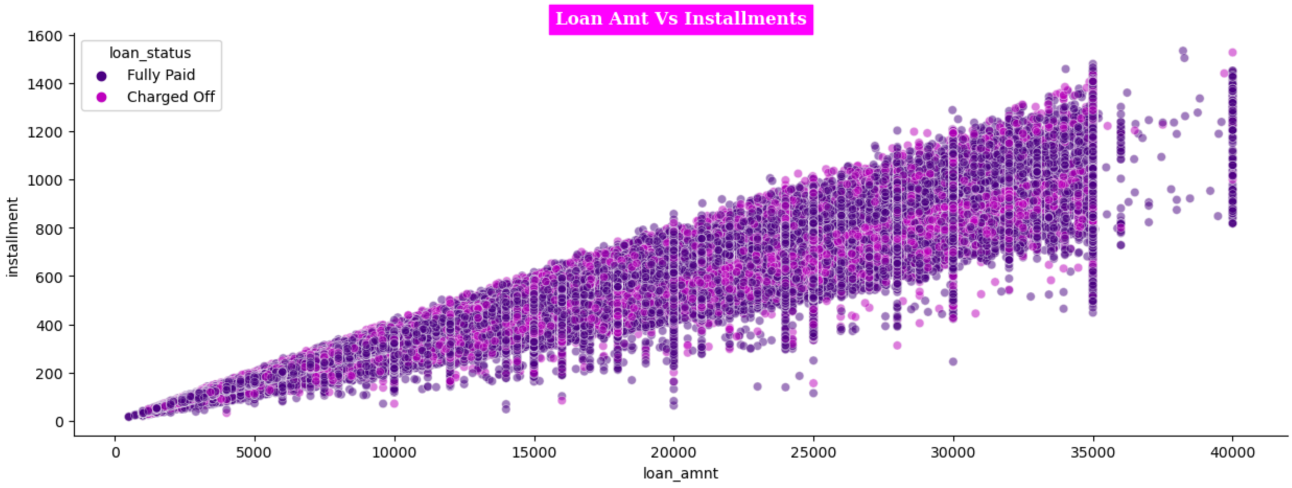


Q2. Comment about the correlation between Loan Amount and Installment features.

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

	loan_amnt	installment
loan_amnt	1.000000	0.953929
installment	0.953929	1.000000

```
plt.figure(figsize = (15,5))
sns.scatterplot(data = df, x = 'loan_amnt', y = 'installment', alpha = 0.5, hue =
plt.title('Loan Amt Vs Installments',fontsize=12,fontfamily='serif',fontweight='b
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 ___.

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

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)

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

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.

```
df[df['emp_title'] != 'No Employee Title']['emp_title'].value_counts().to_frame()
```

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

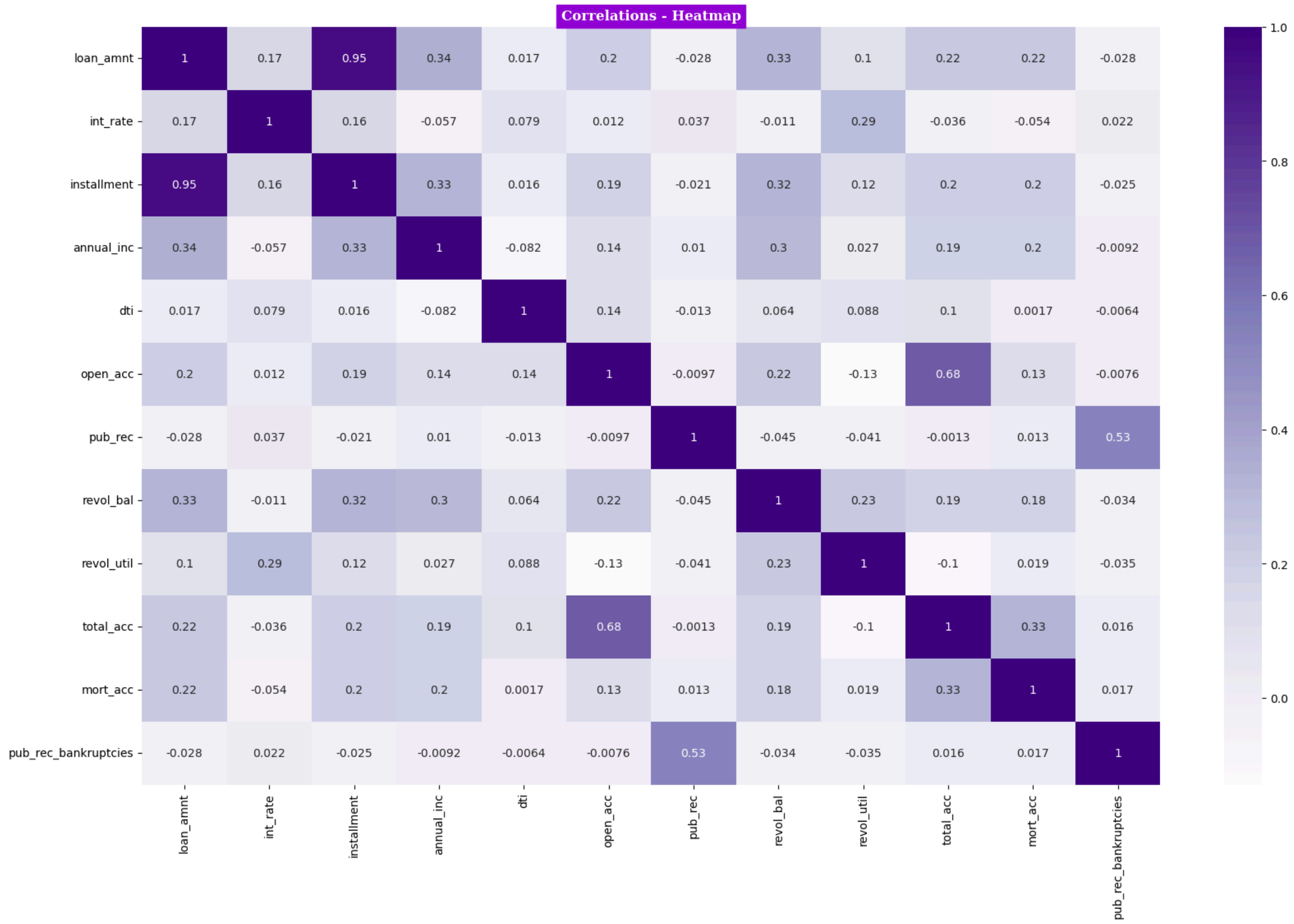
```
df.groupby('emp_title')['loan_status'].count().sort_values(ascending=False).to_fr
```

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

💡 Insights:

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

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


Correlations - Heatmap
loan_amnt
int_rate
installment
annual_inc
dti
open_acc
pub_rec
revol_bal
revol_util
total_acc
mort_acc
pub_rec_bankruptcies

💡 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:

```
numerical_cols = df.select_dtypes(include=np.number).columns
numerical_cols
```

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

```
# outlier treatment
def remove_outliers_zscore(df, threshold=2): #(considering 2 std.dev away from me
    """
    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 threshol
                           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[numerical_co

    # 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)
```

```
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 threshol
                           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[numerical_co

    # Clip outliers
    clipped_values = df[numerical_cols].clip(df[numerical_cols].mean() - threshol
                                             df[numerical_cols].mean() + threshol
                                             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)
```

```
data = cleaned_df.copy()
cp_data = clipped_df.copy()
data.sample()
```

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

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

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

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

```
(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)
```

```
data.shape
```

```
(311392, 30)
```

```
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:

```
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})
```

```
data.head()
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	home_ow
0	10000.0	36	11.44	329.48	B	B4	Marketing	
1	8000.0	36	11.99	265.68	B	B5	Credit analyst	MC
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.	MC

✅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
```

```
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['loan_s
    if p > 0.05:
        print('>>>>>>> Independent feature - Not Significant:',col,' >> p value:'
```

```
>>>>>>> Independent feature - Not Significant: emp_title  >> p value: 0.53671
>>>>>>> Independent feature - Not Significant: title  >> p value: 1.0
>>>>>>> Independent feature - Not Significant: er_cr_line_m  >> p value: 0.27
>>>>>>> Independent feature - Not Significant: state  >> p value: 0.760478089
```

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

```
(311392, 19)
```

```
lt.sample()
```

	loan_amnt	term	int_rate	installment	grade	home_ownership	annual_i

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

```
ltd.shape
```

```
(311392, 50)
```

```
ltd.dtypes
```

```
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
```

```
ltd.sample(8)
```

	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:

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

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

```
#Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,stratify=
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

```
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)
```

```
X_train.head()
```

	loan_amnt	term	int_rate	installment	annual_inc	dti	open_acc	revol_
0	0.379538	0.0	0.339161	0.411590	0.207250	0.465341	0.368421	0.17
1	0.643564	1.0	0.680070	0.524221	0.367868	0.252652	0.473684	0.22
2	0.168317	0.0	0.208625	0.176198	0.134712	0.357576	0.368421	0.05
3	0.379538	1.0	0.680070	0.307444	0.367868	0.449242	0.315789	0.25
4	0.368812	0.0	0.543706	0.421460	0.246109	0.315530	0.263158	0.09

Model-1

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

```
▾ LogisticRegression
LogisticRegression()
```

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

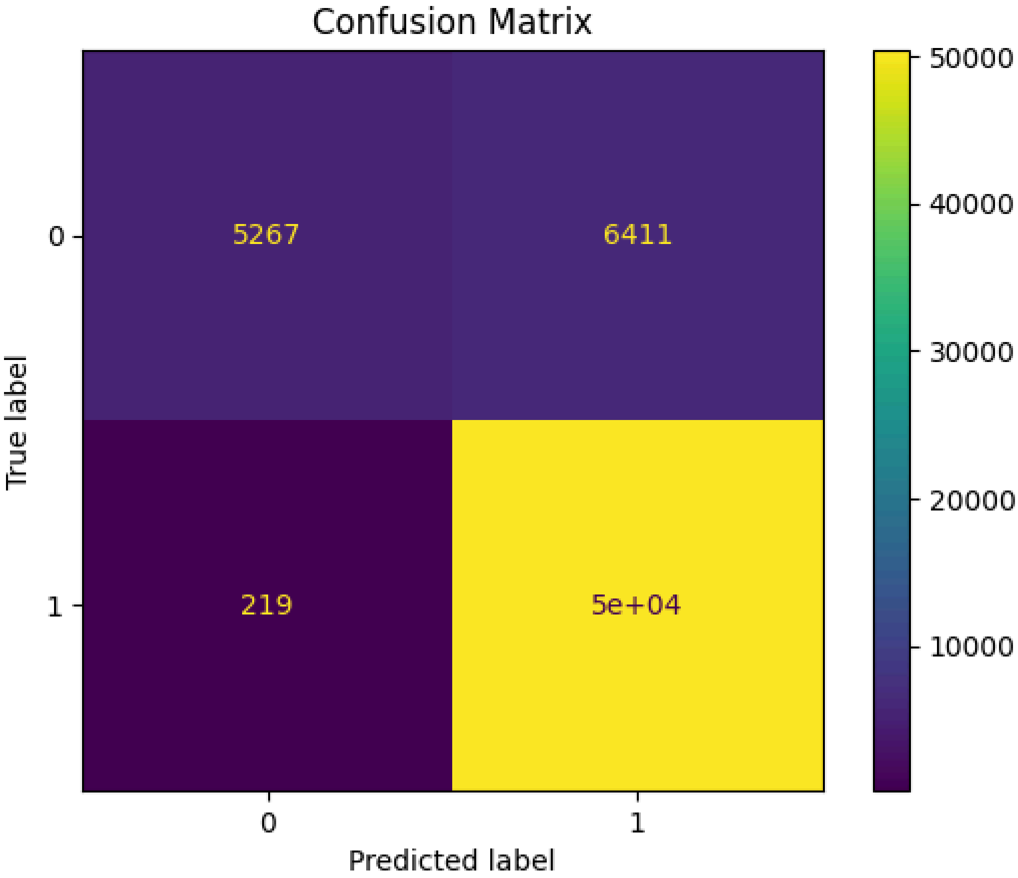
```
logreg_model.score(X_test, y_test) , logreg_model.score(X_test, y_test_pred)
```

```
(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.

```
#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
```




```
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

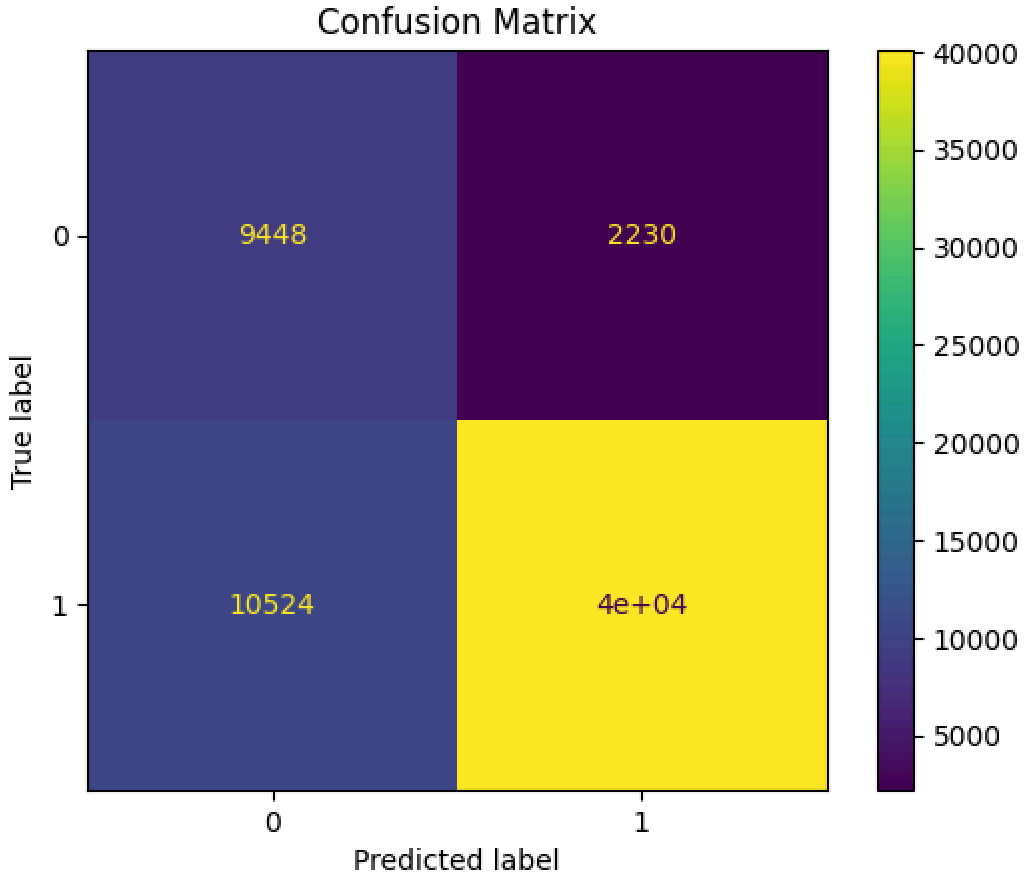
Model-2

```
# 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
```

```
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
```




```
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.

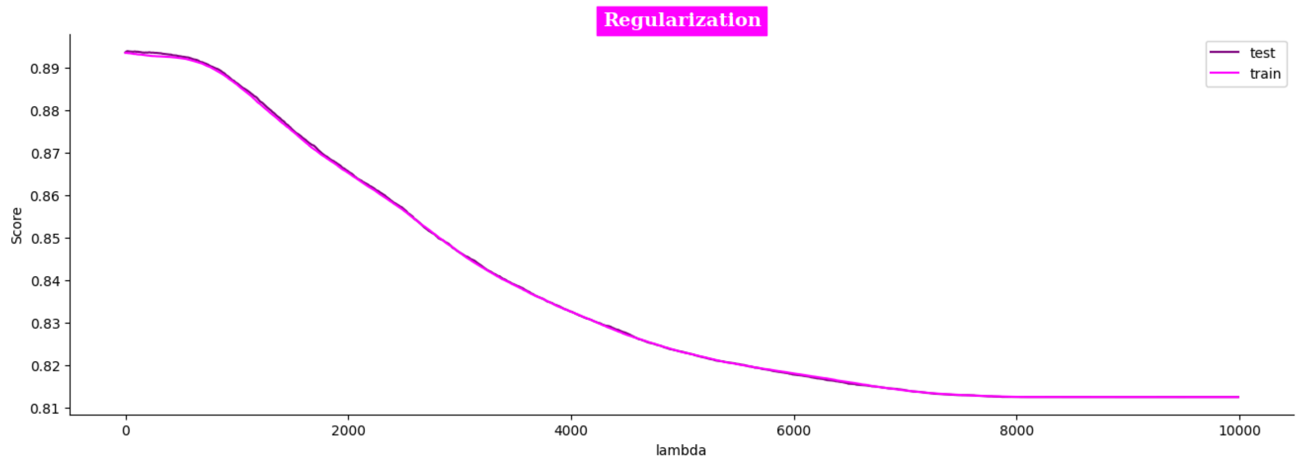
> Explanation :

- 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

```
#Try with different regularization factor lamda and choose the best to build the

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)
```

```
#Plot the train and test scores with respect lambda values i.e. regularization fa
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',backg
plt.xlabel("lambda")
plt.ylabel("Score")
sns.despine()
plt.show()
```




```
#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
```

```
#Calculate the best lambda value based on the index of best test score

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

```
20.01
```

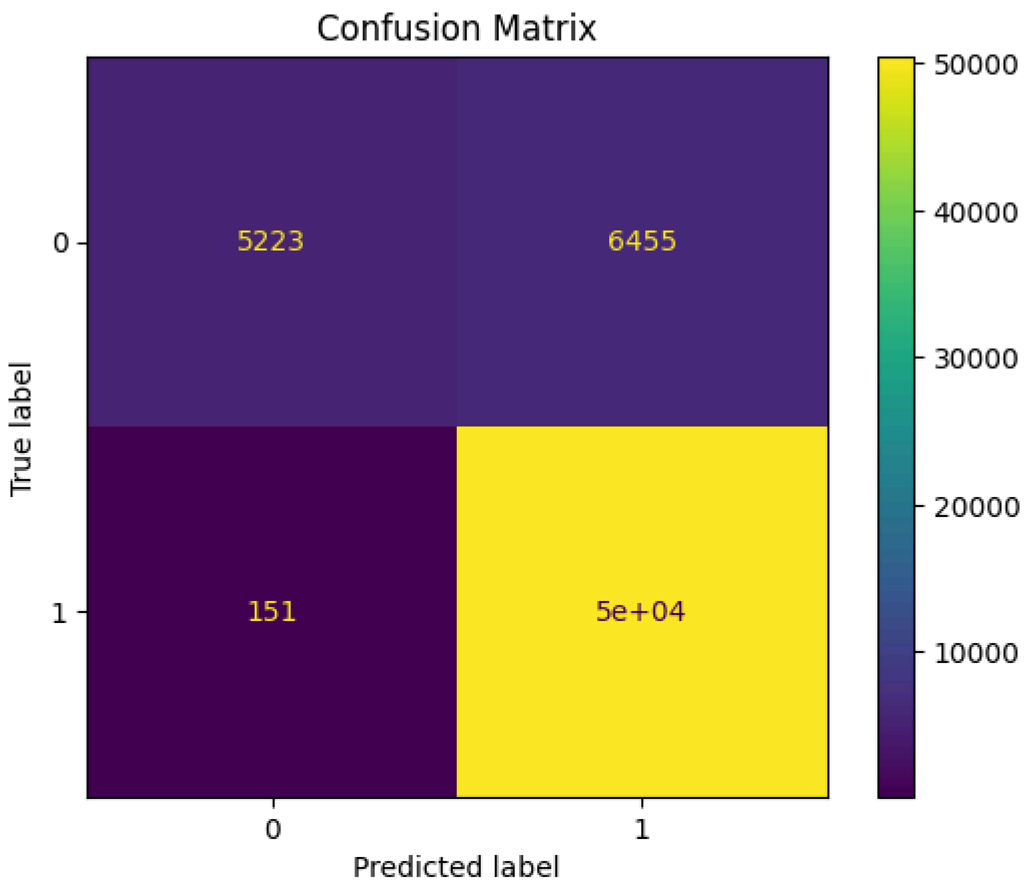
```
#Fit the model using best lambda

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

```
LogisticRegression
LogisticRegression(C=0.04997501249375312)
```

```
#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)


#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 %
```

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



```
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%

```
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
```

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

	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

```
#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
```