```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import sys
path_to_module = '/content/drive/MyDrive/DSML/Custom_Functions'
sys.path.append(path_to_module)
```

```
from Data_Analysis_Visualization import custom_get_df_summary, custom_plot_hist, custom_plot_box, custom_plot_numeric_distribution, custo
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)
import seaborn as sns
# import textwrap
import math
import re
from scipy import stats

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.regression.linear_model import RegressionModel

import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.stattools import durbin_watson
```

```
# import pandas as pd
# import numpy as np
# import seaborn as sns
# from scipy import stats
# import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
```

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.metrics import precision_recall_curve, auc
```

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.10/dist-packages (2.6.1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.22.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.10.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.13.5)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_enco
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (202
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
```

```
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encode
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encode
```

```
from category_encoders import TargetEncoder
```

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

Define Problem Statement and perform Exploratory Data Analysis

About the Business:

LoanTap is an online platform committed to delivering customized loan products to millennials. They innovate in an otherwise dull loan segment, to deliver instant, flexible loans on consumer friendly terms to salaried professionals and businessmen.

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

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

Business Problem

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

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

Concept Used:

- Exploratory Data Analysis
- Feature Engineering
- Logistic Regression
- Precision Vs Recall Tradeoff

Download the dataset

```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv')
df.head(4)
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	a
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	

Columns Profiling:

- 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

Non-Graphical Univariate Analysis Summary

- Observe shape of the data, the data types of all attributes
- Missing value detection, outlier checking, statistical summarization

```
df_summary = custom_get_df_summary(df, print_summary=False, properties_as_columns=False)
```

```
RangeIndex: 396030 entries; Data columns (total 27 columns)
memory usage: 81.6+ MB
```

```
df_summary
```

	purpose	verification_status	application_type	initial_list_status	earliest_cr_l
dtype	object	object	object	object	ob
Missing Counts	0	0	0	0	
nUniques	14	3	3	2	
Top 10 Unique Values	debt_consolidation (59%), credit_card (20%), h...	Verified (35%), Source Verified (33%), Not Ver...	INDIVIDUAL (99%), JOINT (0%), DIRECT_PAY (0%)	f (60%), w (39%)	Oct-2000 (C Aug-2000 (C Oct-2001 (0%),

Check for duplicte records

```
df.duplicated().sum()
```

```
0
```

Unique renewable_energy Source Verified (33%), JOINT (0%), w (39%), f (60%) Nov-1957 (C

Map Loan Status: 1: Fully Paid, 0: Charged Off

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

Univarite and Bivariate Analysis of Categorical Variables

Custom Function to get details analysis of a Categorical Variable

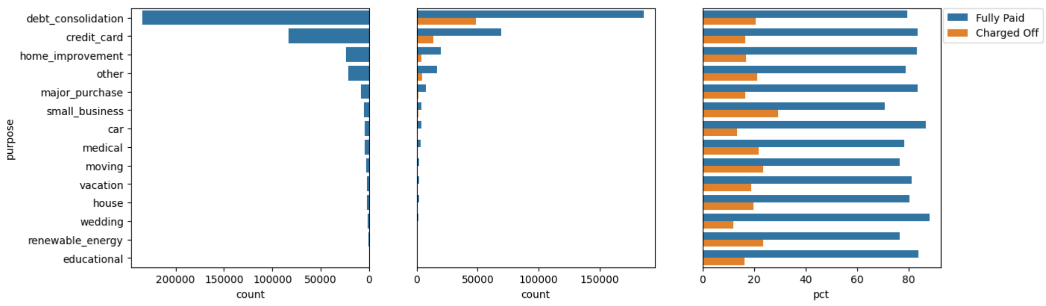
Median nan nan nan nan

```
def analyse_categorical_variable(data=df, var='purpose', target_var='loan_status'):
  if data[var].nunique() <=15:
    fig = plt.figure()
    ax1 = plt.subplot(1, 3, 1)
    ax2 = plt.subplot(1, 3, 2)
    ax3 = plt.subplot(1, 3, 3)

    sns.countplot(ax=ax1, data=data, y=var, order=data[var].value_counts().index, color=sns.color_palette()[0])
    ax1.invert_xaxis()
    # ax1.set_yticks([])
    # ax1.set_yticklabels('')
    ax1.legend('', frameon=False)

    # sns.countplot(ax=ax3, data=data, y=var, order=data[var].value_counts().index, hue=target_var)
    sns.countplot(ax=ax2, data=data, y=var, order=data[var].value_counts().index, hue=target_var)
    ax2.set_ylabel('')
    df2 = df.groupby(var)[target_var].value_counts(normalize=True).mul(100).rename('pct').reset_index()
    sns.barplot(ax=ax3, data=df2, hue=target_var, x='pct', y=var,order=data[var].value_counts().index)
    ax2.set_yticks([])
    ax3.set_yticklabels('')
    ax2.legend('', frameon=False)

    # sns.countplot(ax=ax3, data=data, y=var, order=data[var].value_counts().index, palette = ['darkturquoise']*4 + ['gray']*20)
    ax3.set_ylabel('')
    ax3.set_yticks([])
    ax3.set_yticklabels('')
    ax3.legend(loc='upper left', borderaxespad=0, bbox_to_anchor=(1.01, 1))

    fig.set_size_inches((14, 1+data[var].nunique()/4))
    # fig.subplots_adjust(hspace=0.6, wspace=0.6)
    # fig.tight_layout()

  # Test for Dependance
  pval = stats.chi2_contingency(pd.crosstab(index=df[target_var], columns=df[var]))[1]

  print('Testing for Dependance (Chi2 Test)')
  if pval <= 0.05:
    print(f"Since pval ({pval:.03f}) <= significance level (0.05), Dependant variable ({target_var}) and Predictor variable ({var}): Sigr
  else:
    print(f"Since pval ({pval:.03f}) > significance level (0.05), Dependant variable ({target_var}) and Predictor variable ({var}): Not [
```

purpose

```
analyse_categorical_variable(df, 'purpose')
```

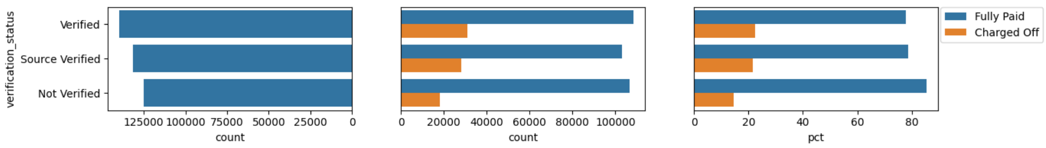
```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



verification_status

```
analyse_categorical_variable(df, 'verification_status')
```

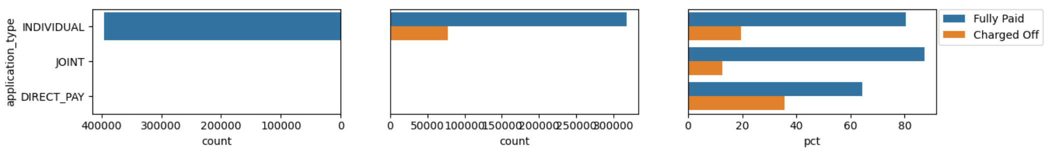
```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



application_type

```
analyse_categorical_variable(df, 'application_type')
```

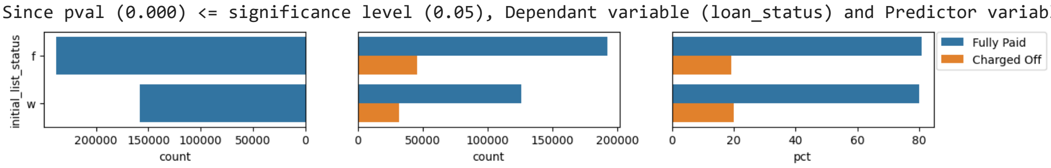
```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



initial_list_status

```
analyse_categorical_variable(df, 'initial_list_status')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



title

```
df.loc[df['title'].isna(), 'title'] = 'unknown'
```

```
df['title'] = df['title'].str.lower().str.replace(' ', '')
```

```
analyse_categorical_variable(df, 'title')
```

```
Testing for Dependance (Chi2 Test)
Since pval (1.000) > significance level (0.05), Dependant variable (loan_status) and Predictor variable (title): Not Dependant
```

```
df = df.drop('title', axis=1)
```

emp_title

```
df.loc[df['emp_title'].isna(), 'emp_title'] = 'unknown'
```

```
df['emp_title'] = df['emp_title'].str.lower().str.replace(' ', '')
```

```
analyse_categorical_variable(df, 'emp_title')
```

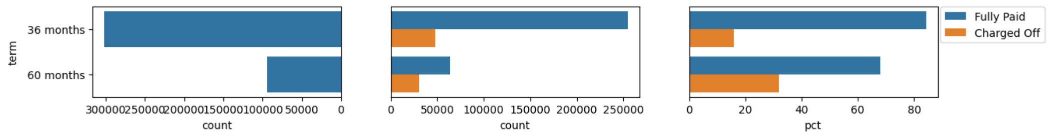
```
Testing for Dependance (Chi2 Test)
Since pval (0.840) > significance level (0.05), Dependant variable (loan_status) and Predictor variable (emp_title): Not Dependant
```

```
df = df.drop('emp_title', axis=1)
```

term

```
analyse_categorical_variable(df, 'term')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```

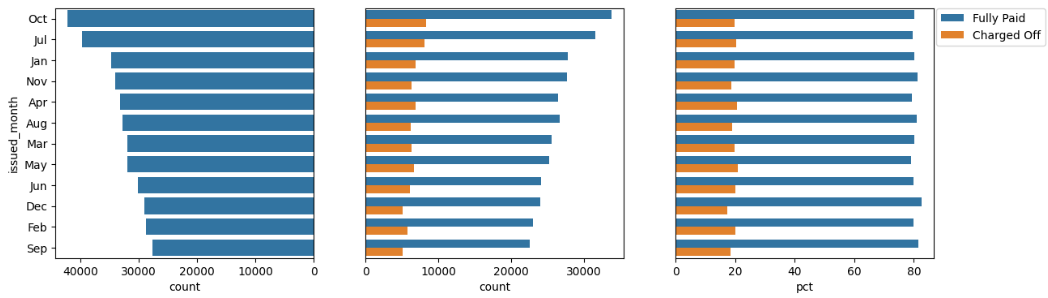


issue_d

```
df['issued_month'] = pd.to_datetime(df['issue_d']).dt.month_name().str[:3]
df = df.drop('issue_d', axis=1)
```

```
analyse_categorical_variable(df, 'issued_month')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```

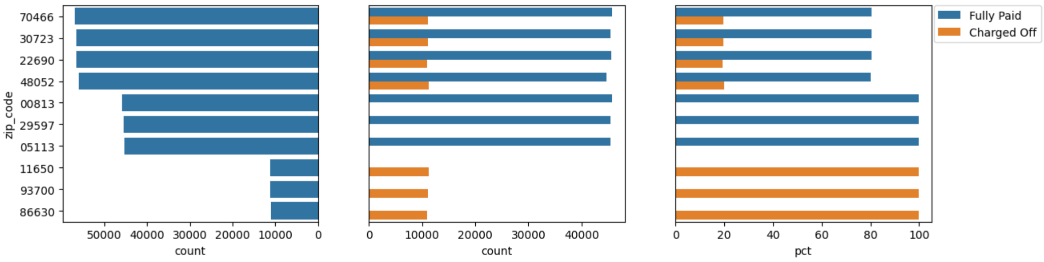


address

```
df['zip_code'] = df['address'].str.split().apply(lambda x: x[-1])
df = df.drop('address', axis=1)
```

```
analyse_categorical_variable(df, 'zip_code')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```

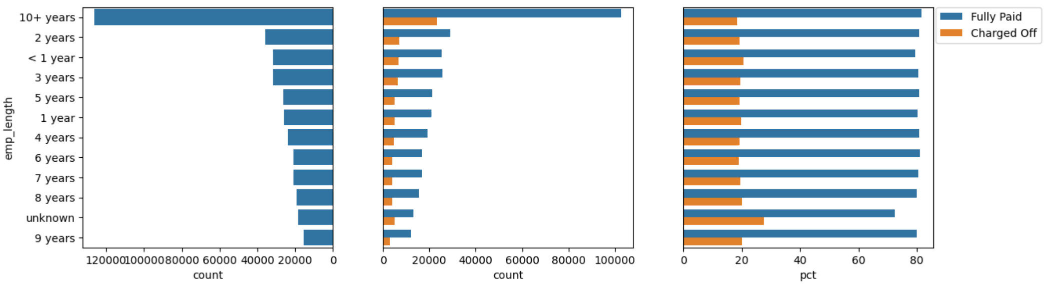


emp_length

```
df.loc[df['emp_length'].isna(), 'emp_length'] = 'unknown'
```

```
analyse_categorical_variable(df, 'emp_length')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



grade

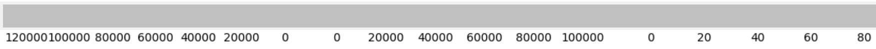
```
analyse_categorical_variable(df, 'grade')
```

```
Testing for Dependance (Chi2 Test)
```

sub_grade


```
analyse_categorical_variable(df, 'sub_grade')
```

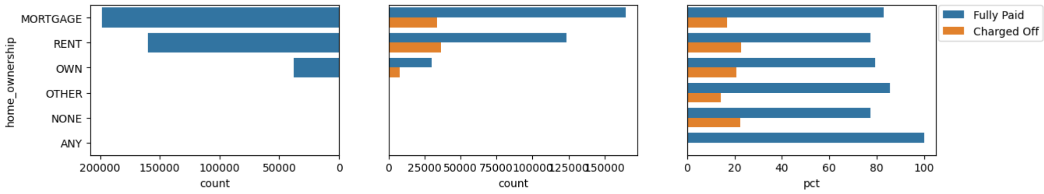
```
    Testing for Dependance (Chi2 Test)
    Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variable (sub_grade): Significnatly
```



home_ownership

```
analyse_categorical_variable(df, 'home_ownership')
```

```
    Testing for Dependance (Chi2 Test)
    Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



Custom Function to get details analysis of a Categorical Variable

```
def analyse_numeric_variable(data=df, var='loan_amnt', target_var='loan_status'):
  fig = plt.figure()
  ax1 = plt.subplot(2, 1, 1)
  ax2 = plt.subplot(2, 1, 2, sharex=ax1)

  sns.histplot(ax=ax1, data=data, x=var, kde=True, bins=30)
  low = max(df['borrower_since_yrs'].mean() - 3*df['borrower_since_yrs'].std(), df['borrower_since_yrs'].min())
  high = min(df['borrower_since_yrs'].mean() + 3*df['borrower_since_yrs'].std(), df['borrower_since_yrs'].max())

  df_low = pd.DataFrame({'x': [low, low], 'y': ax1.get_ybound()})
  df_high = pd.DataFrame({'x': [high, high], 'y': ax1.get_ybound()})
  sns.lineplot(ax=ax1, data=df_low, x='x', y='y', color='red', linestyle='--', estimator=None, linewidth = 1)
  text = 'mean-3*std'
  ax1.annotate(text, xy=(low, 0.7), xycoords=('data', 'figure fraction'), rotation=90)
  sns.lineplot(ax=ax1, data=df_high, x='x', y='y', color='red', linestyle='--', estimator=None, linewidth = 1)
  text = 'mean+3*std'
  ax1.annotate(text, xy=(high, 0.7), xycoords=('data', 'figure fraction'), rotation=90)
  ax1.set_xlabel('')

  sns.histplot(ax=ax2, data=data, x=var, hue=target_var, kde=True, bins=30)

  sns.lineplot(ax=ax2, data=df_low, x='x', y='y', color='red', linestyle='--', estimator=None, linewidth = 1)
  sns.lineplot(ax=ax2, data=df_high, x='x', y='y', color='red', linestyle='--', estimator=None, linewidth = 1)

  if stats.kstest((df[var]), cdf='norm')[1] > 0.05 and stats.levene(df.loc[df['loan_status']=='Fully Paid', 'loan_amnt'], df.loc[df['loar
    print('Testing for Correlation (T Test)')
    pval = stats.ttest_ind(df.loc[df[target_var]=='Fully Paid', var], df.loc[df[target_var]=='Charged Off', var])[1]
  else:
    print('Testing for Correlation (KS Test)')
    pval = stats.kstest(df.loc[df[target_var]=='Fully Paid', var], df.loc[df[target_var]=='Charged Off', var])[1]

  if stats.ttest_ind(df.loc[df[target_var]=='Fully Paid', var], df.loc[df[target_var]=='Charged Off', var])[1] <= 0.05:
    print(f"Since pval ({pval:.03f}) <= significance level (0.05), Dependant variable ({target_var}) and Predictor variable ({var}): Sigr
  else:
    print(f"Since pval ({pval:.03f}) > significance level (0.05), Dependant variable ({target_var}) and Predictor variable ({var}): Not I
```

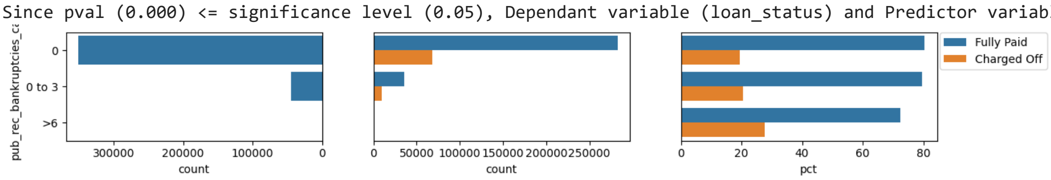
Numberic variable transformation to categorical variable

pub_rec_bankruptcies

```
df['pub_rec_bankruptcies_cat'] = pd.DataFrame(SimpleImputer(strategy='median').fit_transform(df[['pub_rec_bankruptcies']]))
df['pub_rec_bankruptcies_cat'] = pd.cut(df['pub_rec_bankruptcies_cat'], bins=[-1, 0.1, 3, 10], labels=['0', '0 to 3', '>6'])
```

```
analyse_categorical_variable(df, 'pub_rec_bankruptcies_cat')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



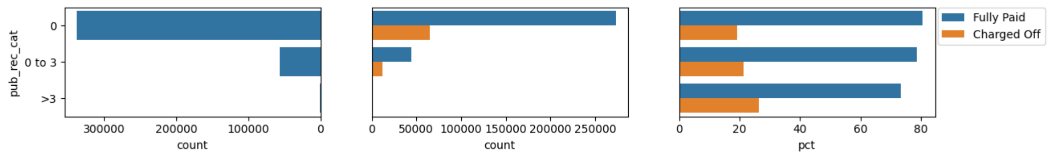

```
df = df.drop('pub_rec_bankruptcies', axis=1)
```

pub_rec

```
df['pub_rec_cat'] = pd.DataFrame(SimpleImputer(strategy='median').fit_transform(df[['pub_rec']]))
df['pub_rec_cat'] = pd.cut(df['pub_rec_cat'], bins=[-1, 0.1, 3, 100], labels=['0', '0 to 3', '>3'])
```

```
analyse_categorical_variable(df, 'pub_rec_cat')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



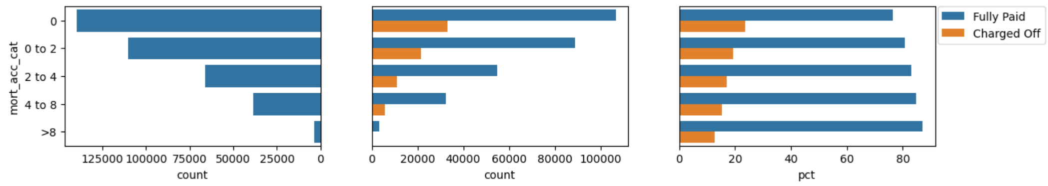

```
df = df.drop('pub_rec', axis=1)
```

mort_acc

```
df['mort_acc'] = pd.DataFrame(SimpleImputer(strategy='median').fit_transform(df[['mort_acc']]))
df['mort_acc_cat'] = pd.cut(df['mort_acc'], bins=[-1, 0.1, 2, 4, 8, 100], labels=['0', '0 to 2', '2 to 4', '4 to 8', '>8'])
```

```
analyse_categorical_variable(df, 'mort_acc_cat')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



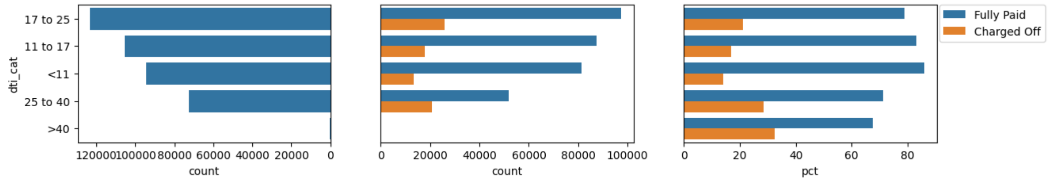

```
df = df.drop('mort_acc', axis=1)
```

dti

```
df['dti_cat'] = pd.cut(df['dti'], bins=[-1, 11, 17, 25, 40, 10000000000], labels=['<11', '11 to 17', '17 to 25', '25 to 40', '>40'])
```

```
analyse_categorical_variable(df, 'dti_cat')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



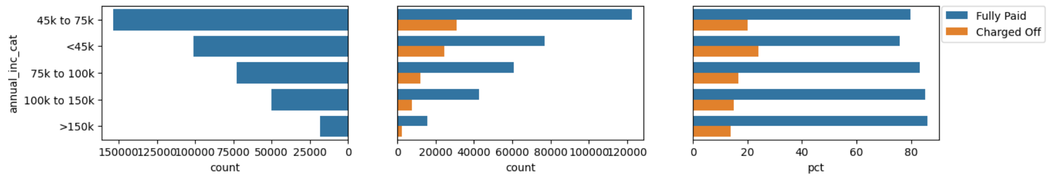

```
df = df.drop('dti', axis=1)
```

annual_inc

```
df['annual_inc_cat'] = pd.cut(df['annual_inc'], bins=[-1, 45000, 75000, 100000, 150000, 10000000000], labels=['<45k', '45k to 75k', '75k
```

```
analyse_categorical_variable(df, 'annual_inc_cat')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



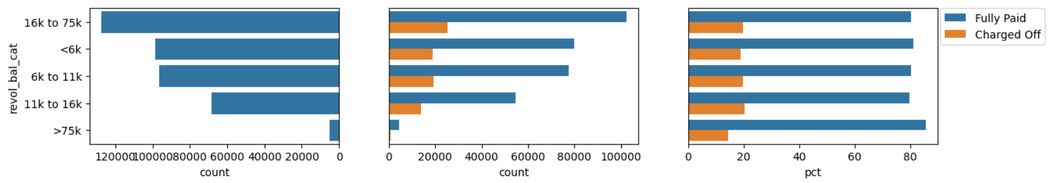

```
df = df.drop('annual_inc', axis=1)
```

revol_bal

```
df['revol_bal_cat'] = pd.cut(df['revol_bal'], bins=[-1, 6000, 11000, 16000, 75000, 10000000000], labels=['<6k', '6k to 11k', '11k to 16k'
```

```
analyse_categorical_variable(df, 'revol_bal_cat')
```

```
Testing for Dependance (Chi2 Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```




```
df = df.drop('revol_bal', axis=1)
```

Numeric Variable Analysis

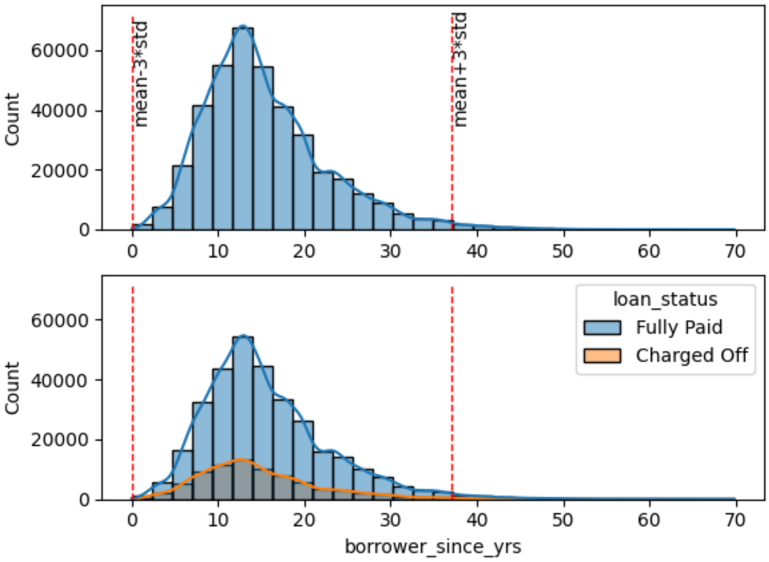
earliest_cr_line

SInce 'earliest_cr_line' represent the month the borrower's earliest reported credit line was opened. We will replace it with new feature 'borrower_since_yrs'

```
df['borrower_since_yrs'] = round((pd.to_datetime(df['earliest_cr_line']).max() - pd.to_datetime(df['earliest_cr_line'])).dt.days/365,2)
df = df.drop('earliest_cr_line', axis=1)
```

```
analyse_numeric_variable(data=df, var='borrower_since_yrs', target_var='loan_status')
```

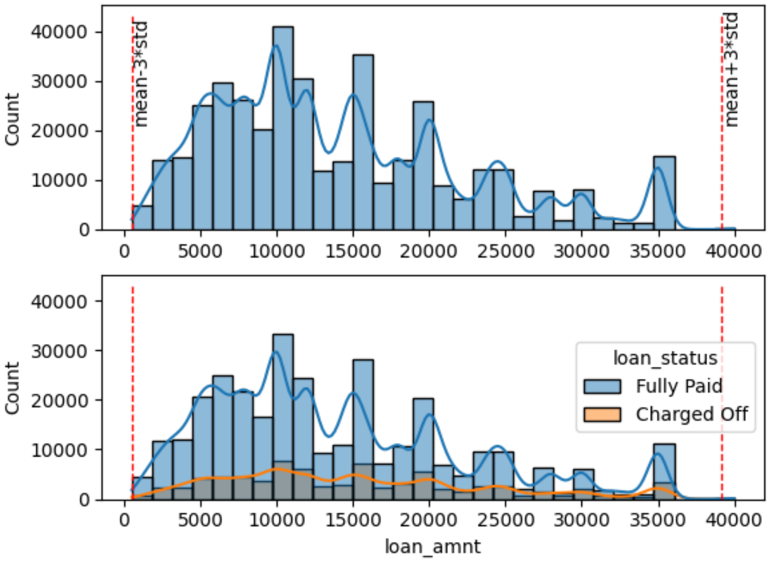
```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



loan_amnt

```
analyse_numeric_variable(data=df, var='loan_amnt', target_var='loan_status')
```

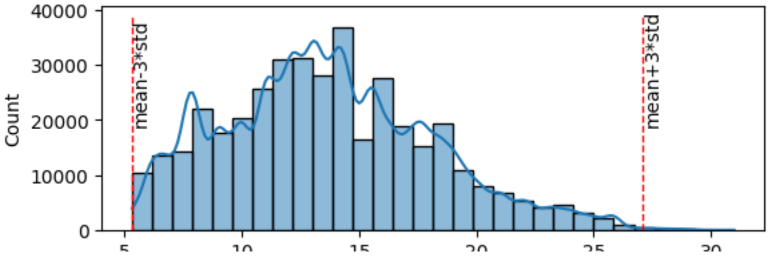
```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



int_rate

```
analyse_numeric_variable(data=df, var='int_rate', target_var='loan_status')
```

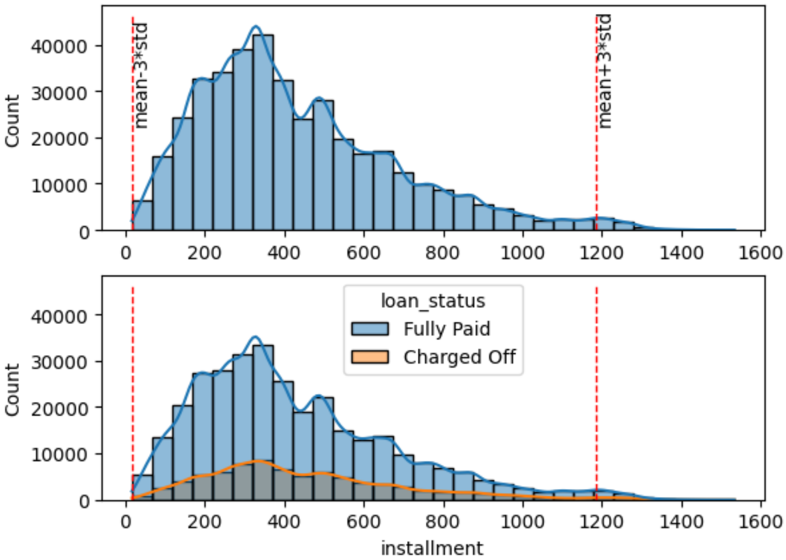
```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



installment

```
analyse_numeric_variable(data=df, var='installment', target_var='loan_status')
```

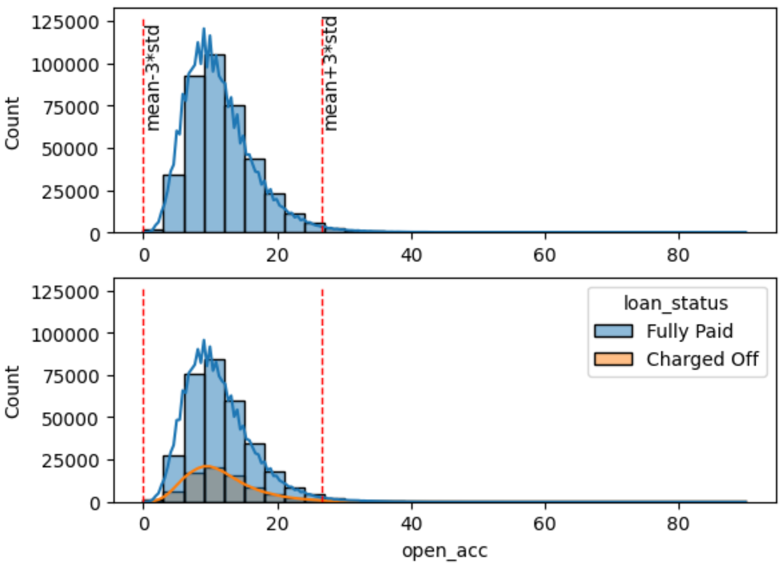
```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



open_acc

```
analyse_numeric_variable(data=df, var='open_acc', target_var='loan_status')
```

```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```

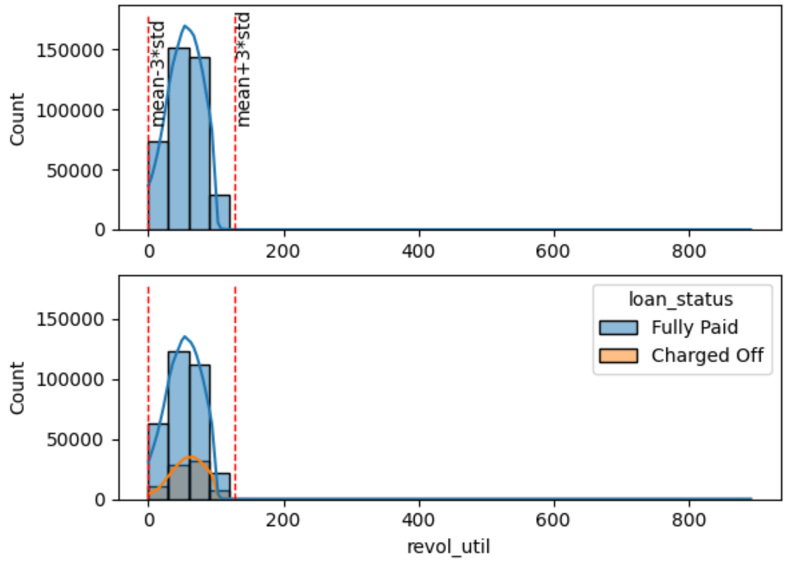


revol_util

```
df['revol_util'] = pd.DataFrame(SimpleImputer(strategy='median').fit_transform(df[['revol_util']]))
```

```
analyse_numeric_variable(data=df, var='revol_util', target_var='loan_status')
```

```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



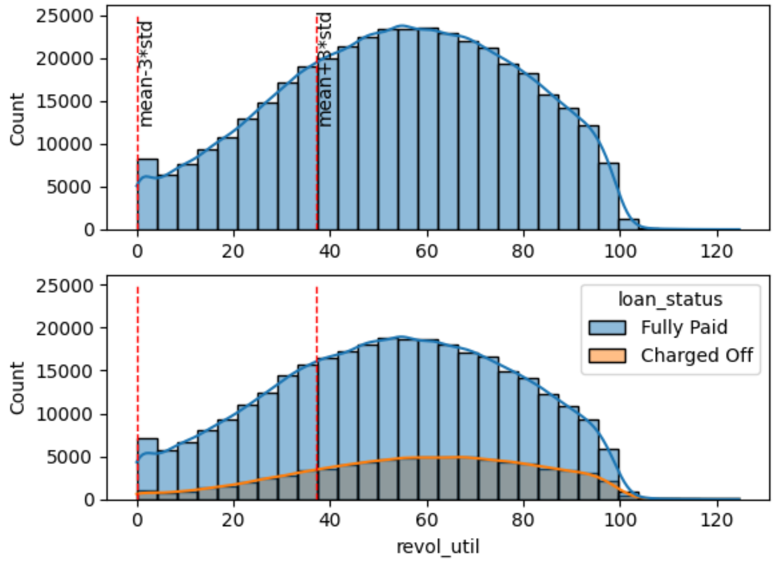

```
(df['revol_util'] > df['revol_util'].mean()+3*df['revol_util'].std()).value_counts()
```

```
False    396014
True         16
Name: revol_util, dtype: int64
```

```
df = df.loc[~(df['revol_util'] > df['revol_util'].mean()+3*df['revol_util'].std())]
```

```
analyse_numeric_variable(data=df, var='revol_util', target_var='loan_status')
```

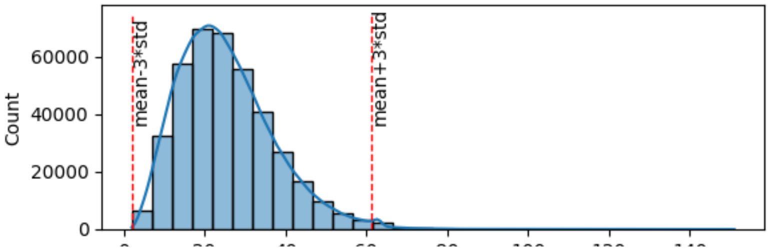
```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



total_acc

```
analyse_numeric_variable(data=df, var='total_acc', target_var='loan_status')
```

```
Testing for Correlation (KS Test)
Since pval (0.000) <= significance level (0.05), Dependant variable (loan_status) and Predictor variabl
```



Get the final Summary of processed Dataset

```
df_summary_1 = custom_get_df_summary(df, False, False)
df_summary_1
```

```
RangeIndex: 396014 entries; Data columns (total 26 columns)
memory usage: 65.7+ MB
```

	initial_list_status	zip_code	issued_month	grade	sub_grade	emp_length	home_ownership
dtype	object	object	object	object	object	object	object
Missing Counts	0	0	0	0	0	0	0
nUniques	2	10	12	7	35	12	6
Top 10 Unique Values	f (60%), w (39%)	70466 (14%), 30723 (14%), 22690 (14%), 48052 (...	Oct (10%), Jul (10%), Jan (8%), Nov (8%), Apr ...	B (29%), C (26%), A (16%), D (16%), E (7%), F ...	B3 (6%), B4 (6%), C1 (5%), C2 (5%), B2 (5%), B...	10+ years (31%), 2 years (9%), < 1 year (8%), ...	MORTGAGE (50%), RENT (40%), OWN (9%), OTHER (0...
Bottom 10 Unique Values	w (39%), f (60%)	86630 (2%), 93700 (2%), 11650 (2%), 05113 (11%...	Sep (6%), Feb (7%), Dec (7%), Jun (7%), May (8...	G (0%), F (2%), E (7%), D (16%), A (16%), C (2...	G5 (0%), G4 (0%), G3 (0%), G2 (0%), G1 (0%), F...	9 years (3%), unknown (4%), 8 years (4%), 7 ye...	ANY (0%), NONE (0%), OTHER (0%), OWN (9%), REN...
min	nan	nan	nan	nan	nan	nan	nan
max	nan	nan	nan	nan	nan	nan	nan
LW (1.5)	nan	nan	nan	nan	nan	nan	nan
Q1	nan	nan	nan	nan	nan	nan	nan
Median	nan	nan	nan	nan	nan	nan	nan
Q3	nan	nan	nan	nan	nan	nan	nan
UW (1.5)	nan	nan	nan	nan	nan	nan	nan
Outlier Count (1.5*IQR)	nan	nan	nan	nan	nan	nan	nan
mean-3*std	nan	nan	nan	nan	nan	nan	nan
mean	nan	nan	nan	nan	nan	nan	nan
std	nan	nan	nan	nan	nan	nan	nan
mean+3*std	nan	nan	nan	nan	nan	nan	nan
Count	nan	nan	nan	nan	nan	nan	nan

Build Logistic Regression Model for Binary Classification

Use OneHotEncoder for categorical variables having categories less than 15

```
encoder = OneHotEncoder(drop='first')
```

```
X = encoder.fit_transform(df[['initial_list_status', 'zip_code', 'issued_month', 'grade', 'emp_length', 'home_ownership', 'verification_s
```

```
# Get the column names for the encoded variables
col_names = encoder.get_feature_names_out(['initial_list_status', 'zip_code', 'issued_month', 'grade', 'emp_length', 'home_ownership', 'v
```

```
X.shape
```

```
(396014, 81)
```

```
X = np.concatenate((X, df[['loan_amnt', 'total_acc', 'revol_util', 'open_acc', 'installment', 'int_rate']].to_numpy(), df[['sub_grade']].
```

```
col_names =  np.concatenate((col_names, ['loan_amnt', 'total_acc', 'revol_util', 'open_acc', 'installment', 'int_rate', 'sub_grade']))
```

```
X.shape
```

```
(396014, 88)
```

```
y = df['loan_status_code']
```

Split the Dataset for Training the Model and Testing its performance on unseen data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)
```

```
X_train[:, :-1].shape
```

```
(316811, 87)
```

```
X_test.shape
```

```
(79203, 88)
```

Use TargetEncoding for the variables having categories more than 15

```
target_encoder = TargetEncoder()
target_encoder.fit(X_train[:, -1], y_train)
```

```
▾    TargetEncoder
TargetEncoder(cols=[0])
```

```
X_train_encoded = np.concatenate(( X_train[:, :-1], target_encoder.transform(X_train[:, -1]) ), axis=1)
```

```
X_test_encoded = np.concatenate(( X_test[:, :-1], target_encoder.transform(X_test[:, -1]) ), axis=1)
```

```
# If any of the categories in the test set are not seen in the training set, replace them with a default value
X_test_encoded[X_test_encoded[:, -1] == np.nan, -1] = X_train_encoded[:, -1].mean()
```

```
(X_test_encoded[:, -1] == np.nan).sum()
```

```
0
```

Use Standard Scaler to scale all the predictors

```
scaler = StandardScaler()
```

```
scaler.fit(X_train_encoded)
```

```
▾ StandardScaler
StandardScaler()
```

```
X_train_encoded_scaled = scaler.transform(X_train_encoded)
```

```
X_train_encoded_scaled.shape
```

```
(316811, 88)
```

```
X_test_encoded_scaled = scaler.transform(X_test_encoded)
```

```
X_test_encoded_scaled.shape
```

```
(79203, 88)
```

Check for Multi-collinearity

```
# var_vif = pd.DataFrame(list(X_train_encoded_scaled.columns)[:-1], columns=['Predictor'])
var_vif = pd.DataFrame()
var_vif['VIF'] = [variance_inflation_factor(X_train_encoded_scaled, i) for i in range(X_train_encoded_scaled.shape[1])]
var_vif
```

	VIF
0	1.098897
1	1.760653
2	1.224428
3	1.919655
4	1.764441
...	...
83	1.715370
84	2.291812
85	52.602566
86	22.598004
87	43.499347

88 rows × 1 columns

```
col_names[var_vif.loc[var_vif['VIF'] > 10].sort_index(ascending=False).index]
```

```
array(['sub_grade', 'int_rate', 'installment', 'loan_amnt',
       'purpose_debt_consolidation', 'purpose_credit_card',
       'home_ownership_RENT', 'home_ownership_OWN',
       'home_ownership_OTHER', 'home_ownership_MORTGAGE', 'grade_E',
       'grade_D', 'grade_C'], dtype=object)
```

```
for i in var_vif.loc[var_vif['VIF'] > 10].sort_index(ascending=False).index:
  X_train_encoded_scaled = np.delete(X_train_encoded_scaled, i, 1)
  X_test_encoded_scaled = np.delete(X_test_encoded_scaled, i, 1)
  col_names = np.delete(col_names, i)
```

```
X_train_encoded_scaled.shape
```

```
(316811, 75)
```

```
var_vif_1 = pd.DataFrame()
var_vif_1['VIF'] = [variance_inflation_factor(X_train_encoded_scaled, i) for i in range(X_train_encoded_scaled.shape[1])]
var_vif_1
```

	VIF
0	1.061724
1	1.760607
...	...

```
col_names[var_vif_1.loc[var_vif_1['VIF'] > 5].sort_index(ascending=False).index]
```

```
array(['application_type_JOINT', 'application_type_INDIVIDUAL'],
      dtype=object)
```

```
for i in var_vif_1.loc[var_vif_1['VIF'] > 5].sort_index(ascending=False).index:
  X_train_encoded_scaled = np.delete(X_train_encoded_scaled, i, 1)
  X_test_encoded_scaled = np.delete(X_test_encoded_scaled, i, 1)
  col_names = np.delete(col_names, i)
```

```
X_train_encoded_scaled.shape
```

```
(316811, 73)
```

Build the Model and access its Performance

```
# Train a logistic regression model
model = LogisticRegression(class_weight='balanced', max_iter = 1000)
model.fit(X_train_encoded_scaled, y_train)

# Predict the test set and evaluate the model's performance
y_pred = model.predict(X_test_encoded_scaled)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[12232  3328]
 [12412 51231]]
              precision    recall  f1-score   support

           0       0.50      0.79      0.61     15560
           1       0.94      0.80      0.87     63643

    accuracy                           0.80     79203
   macro avg       0.72      0.80      0.74     79203
weighted avg       0.85      0.80      0.82     79203
```

Precision and Recall for "Fully Paid" class is decent (94% and 80% repectively).

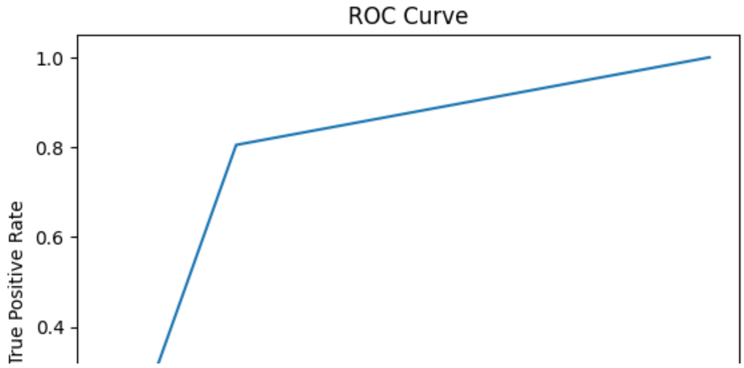
- If Precision value is low (i.e. FP are high), it means Bank's NPA (defaulters) may increase.
- If Recall value is low (i.e. FN are high), it means Bank is loosing in opportunity cost.

Overall Accuracy is Decent Too (80%)

Get AUC-ROC

```
# Predict the test set and evaluate the model's performance
y_pred_proba = model.predict(X_test_encoded_scaled)
auc = roc_auc_score(y_test, y_pred_proba)
print('AUC:', auc)

# Plot the ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```

```
AUC: 0.7955464380014693
```

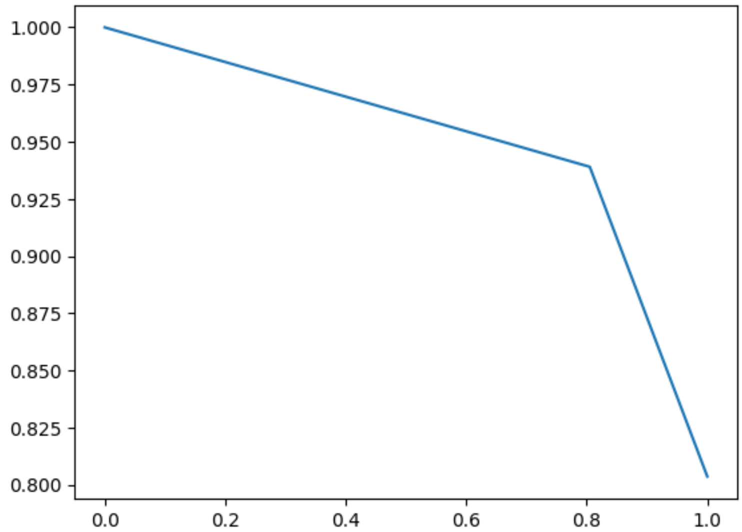


Get AUC-PRC

```
precision, recall, threshold = precision_recall_curve(y_test, y_pred_proba)
```

```
sns.lineplot(x=recall, y=precision)
```

```
<Axes: >
```



Get model intercept and coefficients

```
model.intercept_
```

```
array([2.78155112])
```

```
model_coef = pd.DataFrame(model.coef_, columns=col_names)
```

```
model_coef.T
```

	0
initial_list_status_w	0.015418
zip_code_05113	0.981635
zip_code_11650	-3.185593
zip_code_22690	-2.894815
zip_code_29597	0.983295
...	...
revol_bal_cat_<6k	-0.093033
revol_bal_cat_>75k	0.031040
total_acc	0.100805
revol_util	-0.245869
open_acc	-0.175852

73 rows × 1 columns

```
model_coef.T.sort_values(by=model_coef.T.columns[0], ascending=False)
```

	0
zip_code_29597	0.983295
zip_code_05113	0.981635
grade_B	0.175263
mort_acc_cat_0 to 2	0.135317
mort_acc_cat_2 to 4	0.124727
...	...
zip_code_70466	-2.896657
zip_code_30723	-2.898438
zip_code_86630	-3.175102
zip_code_11650	-3.185593
zip_code_93700	-3.186422

73 rows × 1 columns

Business Insights and Recommendations:

- LoanTap faces a high risk due to approximately 20% of customers defaulting on their loan payments.
- To mitigate this risk, LoanTap can implement more stringent rules to reduce the default rate to 5-6% and offer loans at a slightly higher interest rate than other banks to maintain profitability.
- The model used for prediction has high accuracy, precision, recall, and F1-score, but has a relatively low capability in identifying defaulters. The significant features impacting the outcome include interest rate, loan subgrade, number of payments, home ownership, purpose, application type, pincode, and job title.
- Pincode-based market segmentation can be included at the strategic level to increase presence in pincodes with positive coefficients and minimize marketing expenditure in pincodes with negative coefficients.
- Job titles can be used for social media-based marketing.
- Promoting joint loan applications can help reduce the chances of default.
- LoanTap should stick to giving loans for conventional purposes like marriage, cars, and avoid renewable energy.
- The company should focus more on loans with shorter durations and adapt its marketing strategy accordingly.