Problem Statement

The objective of this case study is to develop a predictive model for LoanTap, an online platform specializing in customized loan products for millennials. The focus is on the underwriting process for Personal Loans. Given a set of attributes for an individual, the goal is to determine whether a credit line should be extended and, if so, to provide recommendations on the repayment terms. The model should accurately predict the likelihood of loan repayment versus default, helping LoanTap make informed lending decisions and minimize the risk of non-performing assets (NPAs).

Imports and Dataset

Imports

Imports for EDA

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Encoding Imports

In [ ]:
```
from sklearn.impute import SimpleImputer
```

Imports for Logistic Regression

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

VIF import

In [ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Dataset

In [ ]:
```
df = pd.read_csv("loan_tap.csv")
df.head()
```

Out[ ]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_own
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_own
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MOR
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MOR

5 rows × 27 columns

Data Preprocessing and EDA

Basic Checks

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

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

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

Out[ ]:
```
0
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 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
```

- Total Number of rows: 396030
- Total Number of Columns: 27
- No duplicates.
- There are some missing values.

Missing values Treatment

In [ ]:

```
missing = df.isna().sum() * 100 / len(df)
missing = missing[(missing > 0)]

print('Columns with missing values')
missing
```

```
Columns with missing values
```

Out[ ]:

```
emp_title               5.789208
emp_length              4.621115
title                   0.443148
revol_util              0.069692
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
dtype: float64
```

Removing rows which contains less than 3% missing values.

In [ ]:

```
df.dropna(subset=(missing[missing <= 3]).index, inplace=True)
```

In [ ]:

```
missing = df.isna().sum() * 100 / len(df)
missing = missing[(missing > 0)]

print('Columns with % missing values')
missing
```

```
Columns with % missing values
```

Out[ ]:

```
emp_title     5.761122
emp_length    4.594055
mort_acc      9.453192
dtype: float64
```

Filling emp_title missing values with 'Unknown'

In [ ]:

```
df['emp_title'].fillna(value= 'Unknown', inplace=True)
```

Filling emp_length missing values with most_frequent value

In [ ]:

```
emp_length_imputer = SimpleImputer(strategy="most_frequent")
df['emp_length'] = emp_length_imputer.fit_transform(df[['emp_length']]).flatten()
```

Filling mort_acc missing values with

In [ ]:
```
mort_acc_imputer = SimpleImputer(strategy="median")
df['mort_acc'] = mort_acc_imputer.fit_transform(df[['mort_acc']]).flatten()
```

In [ ]:
```
missing = df.isna().sum() * 100 / len(df)
missing = missing[(missing > 0)]

print('Columns with % missing values')
missing
```

```
Columns with % missing values
```

Out[ ]:
```
Series([], dtype: float64)
```

Missing values has been treated.

Other Columns Treatment

Change dtype of date/time columns

In [ ]:
```
df['issue_d'][:5]
```

Out[ ]:
```
0    Jan-2015
1    Jan-2015
2    Jan-2015
3    Nov-2014
4    Apr-2013
Name: issue_d, dtype: object
```

In [ ]:
```
df['earliest_cr_line'][:5]
```

Out[ ]:
```
0    Jun-1990
1    Jul-2004
2    Aug-2007
3    Sep-2006
4    Mar-1999
Name: earliest_cr_line, dtype: object
```

In [ ]:
```
# Note below steps will add day in date value, so later on if we used these column,

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

Feature Engineering

Flagging of pub_rec, mort_acc and pub_rec_bankruptcies

In [ ]:
```
def pub_rec(n: float) -> int:
    return int(n != 0)

def mort_acc(n: float) -> int:
    return int(n != 0)


def pub_rec_bankruptcies(n: float) -> int:
    return int(n != 0)
```

In [ ]:
```
df['pub_rec'] = df['pub_rec'].apply(pub_rec)
df['mort_acc'] = df['mort_acc'].apply(mort_acc)
df['pub_rec_bankruptcies'] = df['pub_rec_bankruptcies'].apply(pub_rec_bankruptcies)
```

Extracting zip_codes and states from states

In [ ]:
```
# utility function to extract states
def find_state(s: str) -> str:
    ind = s.find(',')
    output = ""

    # finding abbrevation of state
    if ind != -1:
        output = s[ind+2 : ind+4]

    if output == "Bo" or output == "":
        output = "OTHER"

    return output

def find_zip_code(s: str) -> str:
    return s[-5:]
```

In [ ]:
```
df['state'] = df['address'].apply(find_state)
df['zip_code'] = df['address'].apply(find_zip_code)

df.drop('address', axis=1, inplace=True) # dropping address column
```

Date-time components

In [ ]:
```
df['issue_d_month'] = df['issue_d'].dt.month
df['issue_d_year'] = df['issue_d'].dt.year

df['earliest_cr_line_month'] = df['earliest_cr_line'].dt.month
df['earliest_cr_line_year'] = df['earliest_cr_line'].dt.year
```

In [ ]:
```
df.drop(['issue_d', 'earliest_cr_line'], axis=1, inplace=True) # dropping issue_d an
```

Analysis and Visualisations

value_counts()

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

Out[ ]:
```
Fully Paid     0.80381
Charged Off    0.19619
Name: loan_status, dtype: float64
```

In [ ]:
```
df['home_ownership'].value_counts()
```

Out[ ]:
```
MORTGAGE    197110
RENT        158770
OWN          37443
```

```
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

combining none and any to other

In [ ]:
```
df.loc[df['home_ownership'].isin(['NONE', 'ANY']), 'home_ownership'] = "OTHER"
df['home_ownership'].value_counts()
```

Out[ ]:
```
MORTGAGE    197110
RENT        158770
OWN          37443
OTHER          142
Name: home_ownership, dtype: int64
```

In [ ]:
```
df['title'].value_counts()[:20]
```

Out[ ]:
```
Debt consolidation            152392
Credit card refinancing        51476
Home improvement               15245
Other                          12910
Debt Consolidation             11584
Major purchase                  4759
Consolidation                   3840
debt consolidation              3543
Business                        2947
Debt Consolidation Loan         2859
Medical expenses                2733
Car financing                   2135
Credit Card Consolidation       1768
Vacation                        1715
Moving and relocation           1688
consolidation                   1594
Personal Loan                   1568
Consolidation Loan              1295
Home Improvement                1265
Home buying                     1183
Name: title, dtype: int64
```

- (Consolidation, consolidation), (Debt consolidation, debt consolidation) etc are repeating.
- using str.lower to remove such discrepancies

In [ ]:
```
df['title'] = df['title'].str.lower()
df['title'].value_counts()[:20]
```

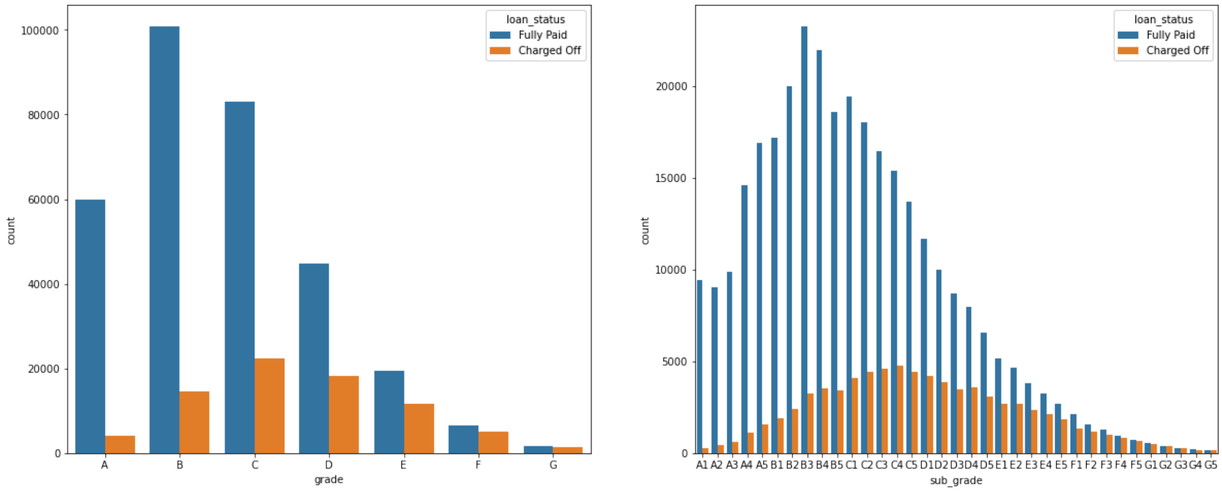
Out[ ]:
```
debt consolidation            168000
credit card refinancing        51770
home improvement               17093
other                          12973
consolidation                   5570
major purchase                  4988
debt consolidation loan         3508
business                        3015
medical expenses                2811
credit card consolidation       2629
personal loan                   2424
car financing                   2156
credit card payoff              1904
consolidation loan              1881
vacation                        1863
credit card refinance           1832
moving and relocation           1692
consolidate                     1523
```

```
personal                1457
home buying             1196
Name: title, dtype: int64
```

bar plots

In [ ]:

```
plt.figure(figsize=(20, 8))

plt.subplot(1, 2, 1)
sns.countplot(x = df['grade'], hue=df['loan_status'], order = sorted(df['grade'].uni

plt.subplot(1, 2, 2)
sns.countplot(x = df['sub_grade'], hue=df['loan_status'], order = sorted(df['sub_gra
plt.show()
```

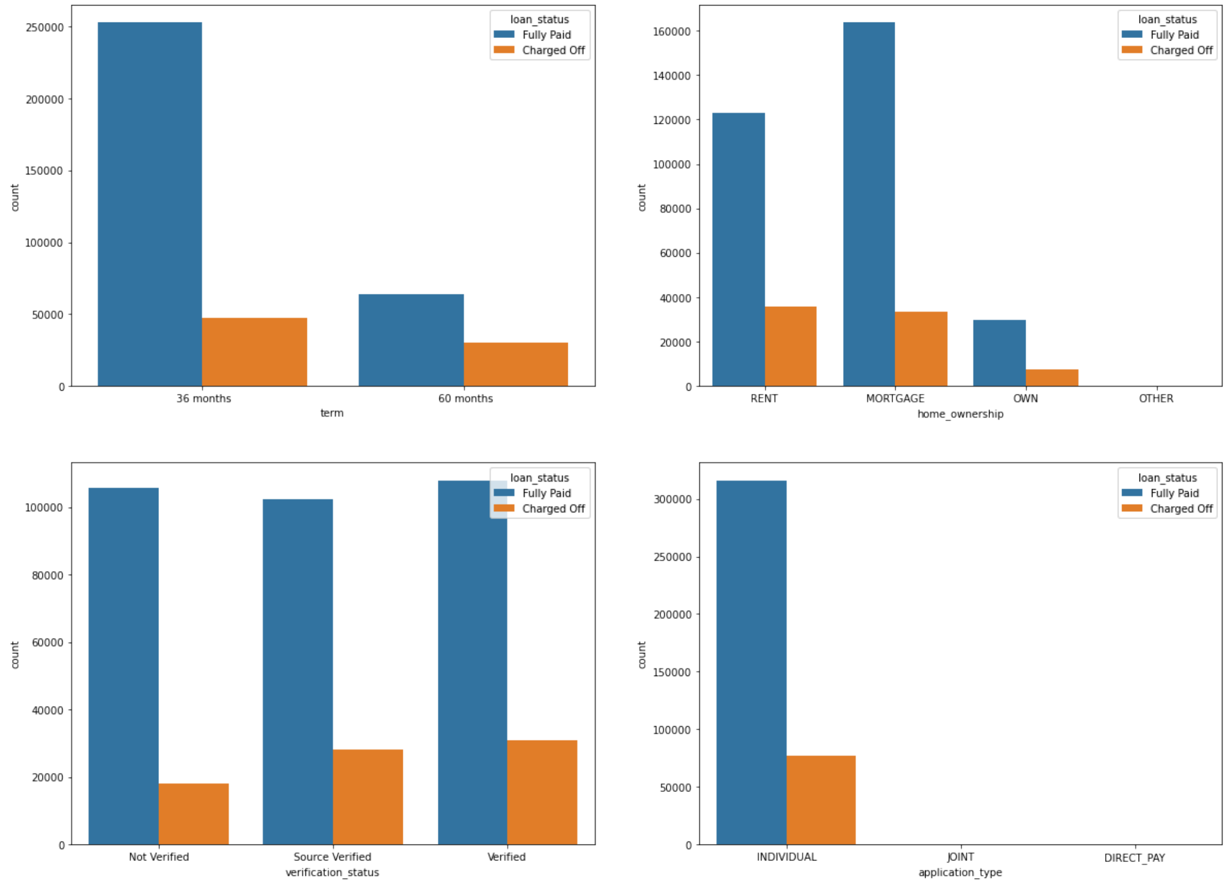


Insights:-

1. Grade A, B, C and D are grades where fully-paid applicants are twice of defaulters.
2. Out of above grades, grade B applicants are maximum that paid the loan fully.

In [ ]:

```
plt.figure(figsize=(20, 15))

plt.subplot(2, 2, 1)
sns.countplot(x=df['term'], hue=df['loan_status'])

plt.subplot(2, 2, 2)
sns.countplot(x=df['home_ownership'], hue=df['loan_status'])

plt.subplot(2, 2, 3)
sns.countplot(x=df['verification_status'], hue=df['loan_status'])

plt.subplot(2, 2, 4)
sns.countplot(x=df['application_type'], hue=df['loan_status'])

plt.show()
```

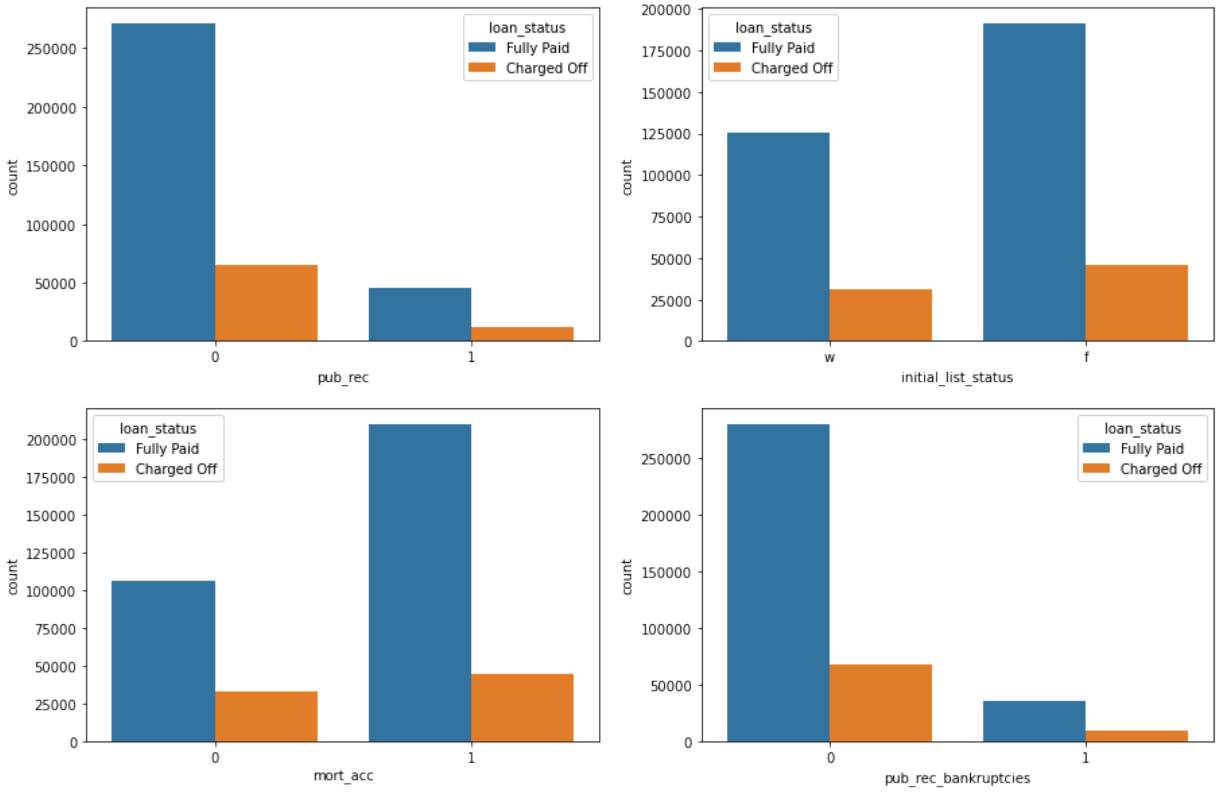


Insights:-

1. 36 months period is opted by most of the applicants.
2. 36 months tenure loan has high success rate (fully paid rate) than 60 months rate.
3. Most of the applicants report their home-ownership as 'Mortgage' or 'Rent'.
4. Individual Applicants generally applied for the loan.

In [ ]:

```
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
sns.countplot(x=df['pub_rec'], hue=df['loan_status'])

plt.subplot(2, 2, 2)
sns.countplot(x=df['initial_list_status'], hue=df['loan_status'])

plt.subplot(2, 2, 3)
sns.countplot(x=df['mort_acc'], hue=df['loan_status'])

plt.subplot(2, 2, 4)
sns.countplot(x=df['pub_rec_bankruptcies'], hue=df['loan_status'])

plt.show()
```

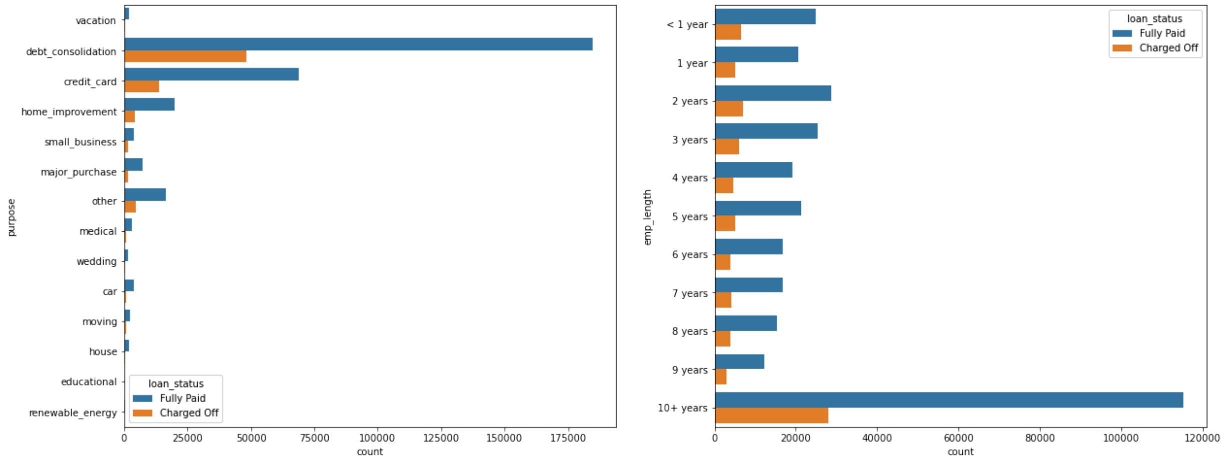


Insights:-

1. Applicants with no derogatory remarks have high chance of getting loan and higher success of paying the loan back.
2. fractional loan status count is slightly high compared to whole loan status.
3. Applicants with mortgage accounts have high chance of getting loan and higher success of paying the loan back.
4. Applicants with no bankruptcies have high chance of getting loan and higher success of paying the loan back.

In [ ]:

```
plt.figure(figsize = (20, 8))

plt.subplot(1, 2, 1)
sns.countplot(y = df['purpose'], hue = df['loan_status'])


plt.subplot(1, 2, 2)
order = ['< 1 year', '1 year', '2 years', '3 years', '4 years', '5 years', '6 years'
sns.countplot(y = df['emp_length'], hue = df['loan_status'], order = order)

plt.show()
```

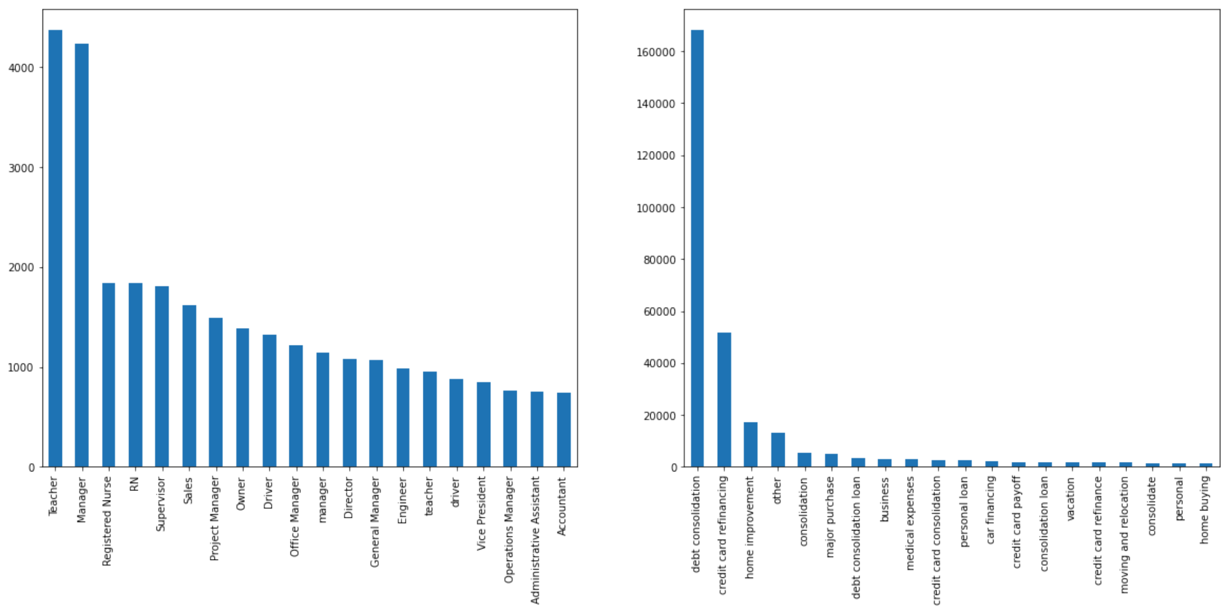


Insights:-

1. Debt consolidation and credit card are the major purpose to be reported in loan application.
2. Applicants with larger length of employment has high chance of getting a loan.

In [ ]:
```
plt.figure(figsize = (20, 8))

plt.subplot(1, 2, 1)
df['emp_title'].value_counts()[1:21].plot(kind='bar')

plt.subplot(1, 2, 2)
df['title'].value_counts()[:20].plot(kind='bar')
```

Out[ ]: <AxesSubplot:>

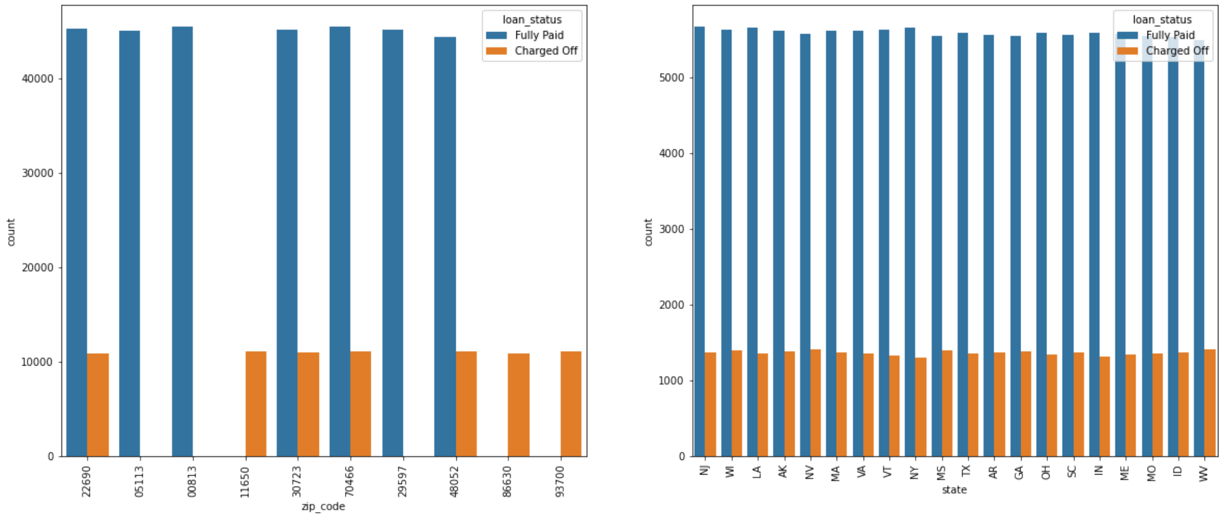


Insights:-

1. Teacher and Manager are occupation with highest chance of getting the loan.

In [ ]:
```
plt.figure(figsize = (20, 8))

plt.subplot(1, 2, 1)
sns.countplot(x = df['zip_code'], hue = df['loan_status'])
plt.xticks(rotation = 90)

plt.subplot(1, 2, 2)
top_20_states = df['state'].value_counts()[1:21].index
```

```
df_top_20 = df.loc[df['state'].isin(top_20_states)]
sns.countplot(x = df_top_20['state'], hue=df_top_20['loan_status'], order=top_20_sta
plt.xticks(rotation = 90)

plt.show()
```



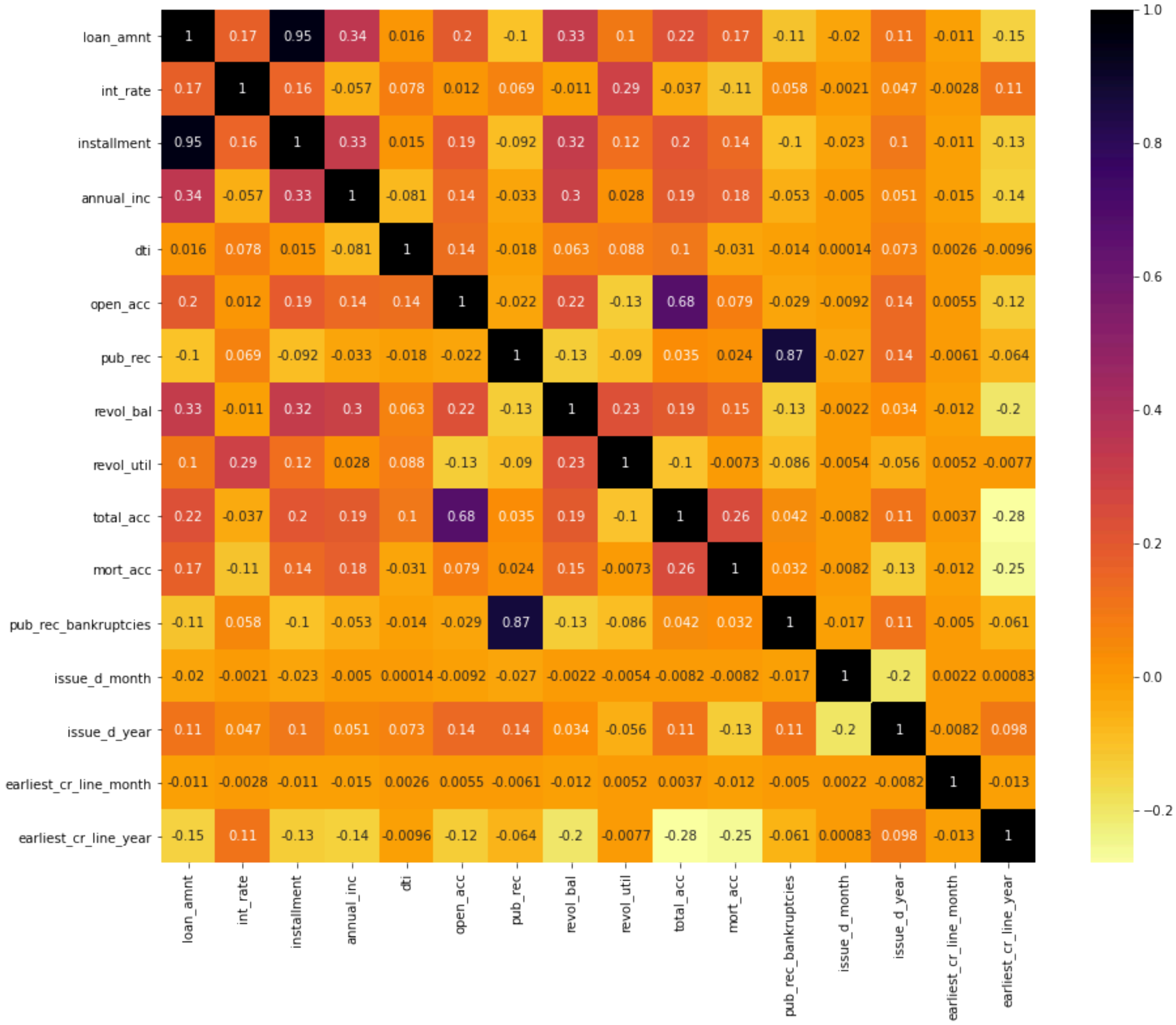
Insights:-

1. Zip-code: 05113, 00813, 29597 are the areas where almost every applicant has paid their loan successfully.
2. Zip-code 11650, 86630, 93700 are the areas where almost every applicant has defaulted.

Co-relation Matrix and Heatmaps

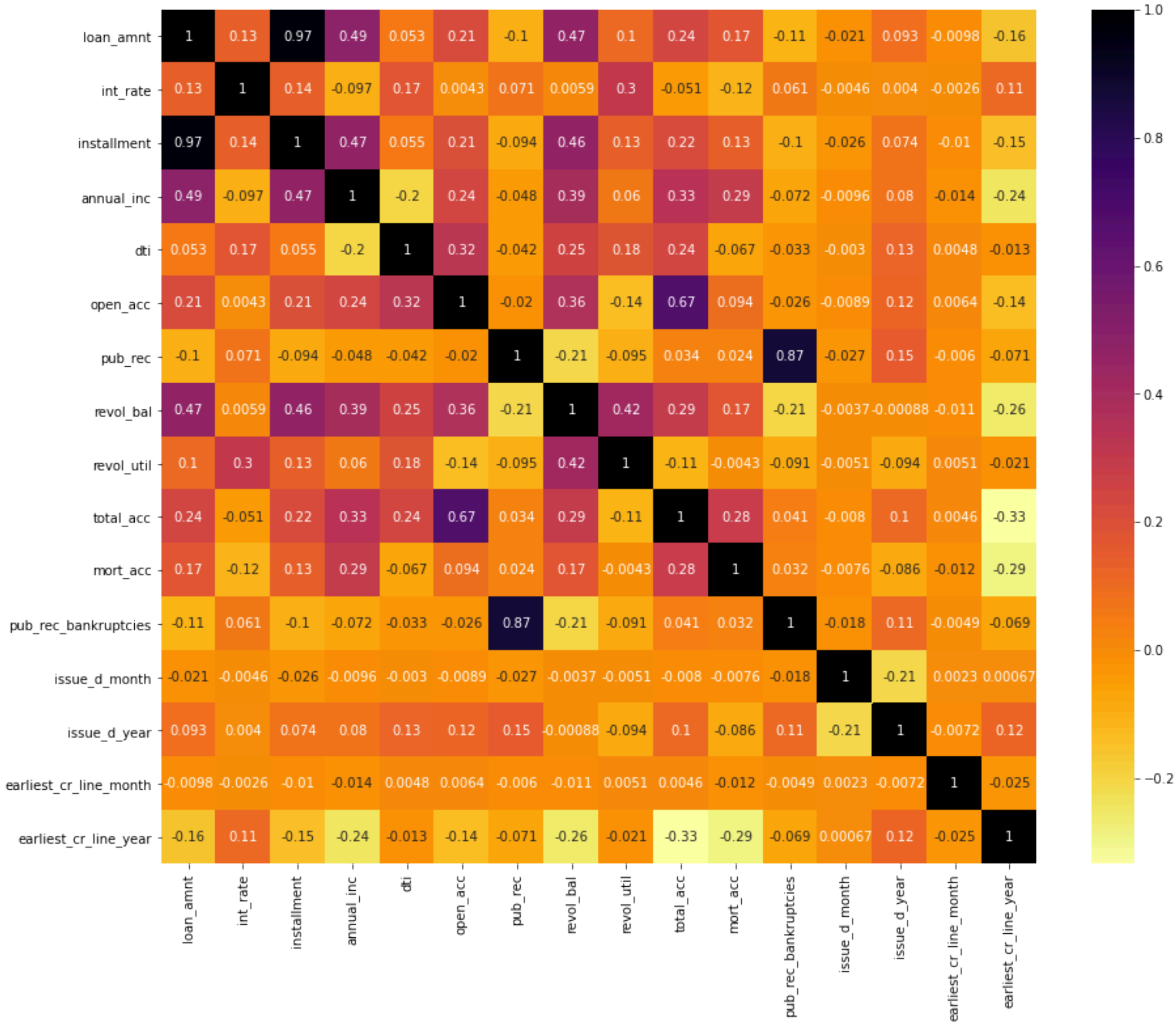
In [ ]:

```
plt.figure(figsize=(15, 12))
sns.heatmap(df.corr(method="pearson"), annot = True, cmap = "inferno_r")
plt.show()
```



In [ ]:

```
plt.figure(figsize=(15, 12))
sns.heatmap(df.corr(method="spearman"), annot = True, cmap = "inferno_r")
plt.show()
```



- (loan_amnt, installment), (pub_rec_bankruptcies, pub_rec) pairs are highly co-related.
- Hence dropping installment and pub_rec_bankruptcies.

In [ ]:
```
df.drop(columns=['installment', 'pub_rec_bankruptcies'], inplace=True)
```

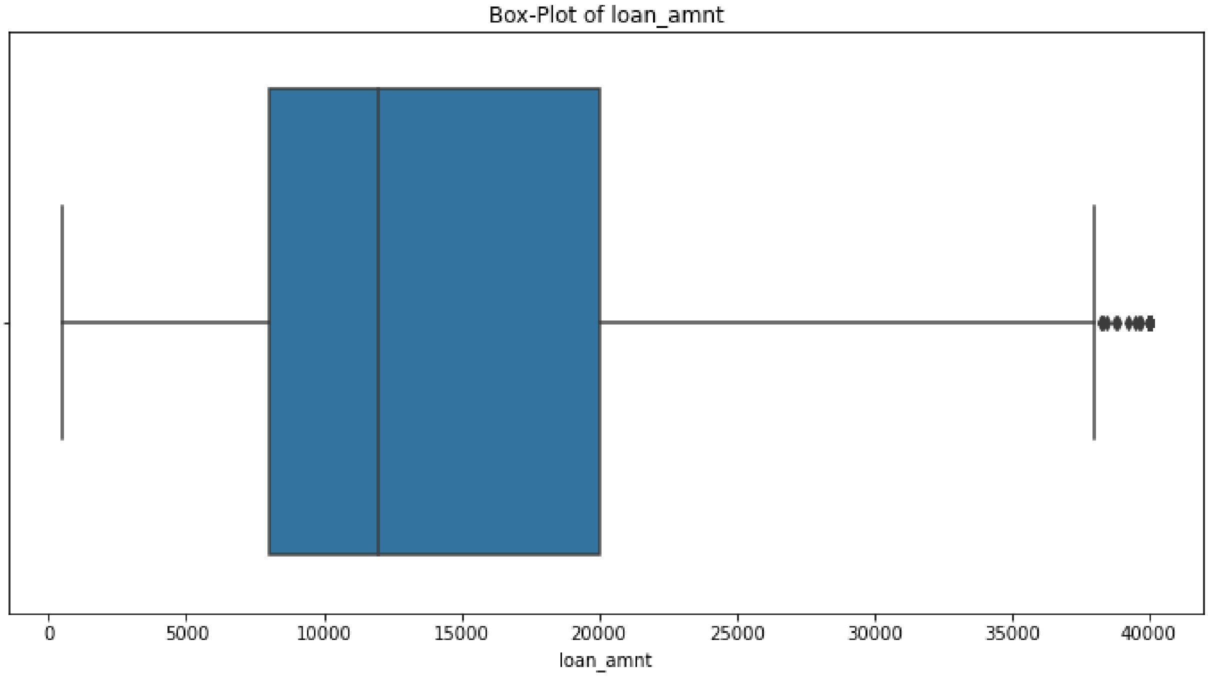
Outliers Detection using histogram and their treatment

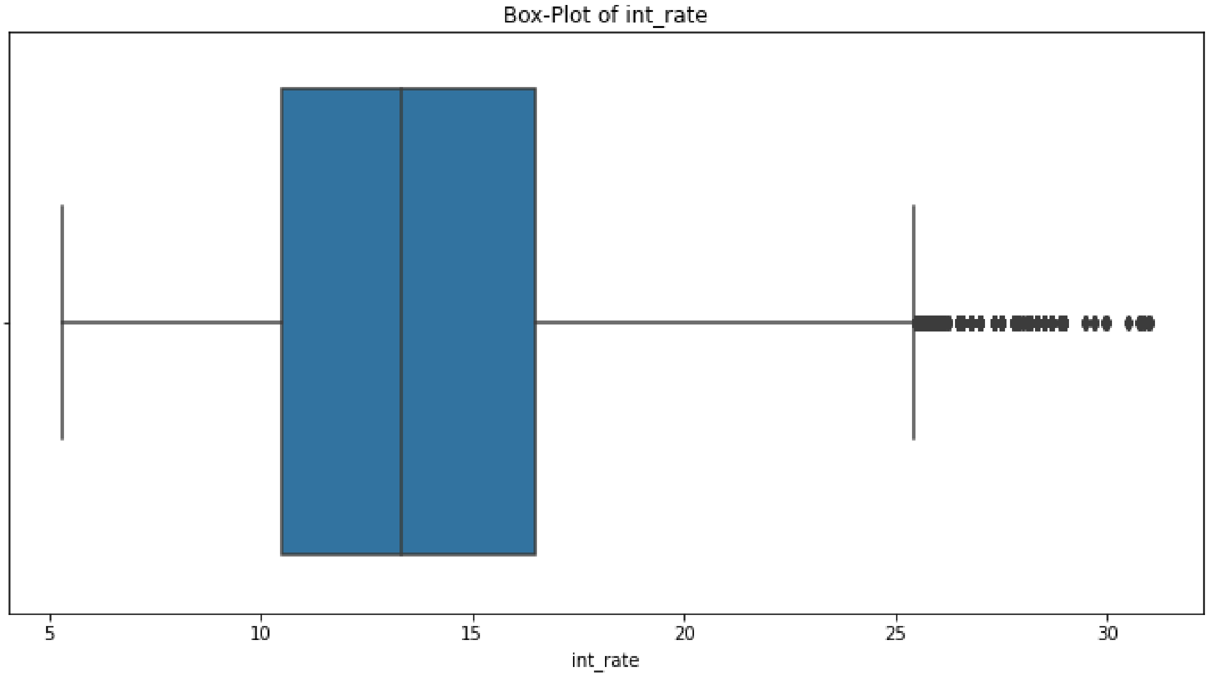
In [ ]:
```
num_cols = df.select_dtypes(include='number').columns
num_cols
```

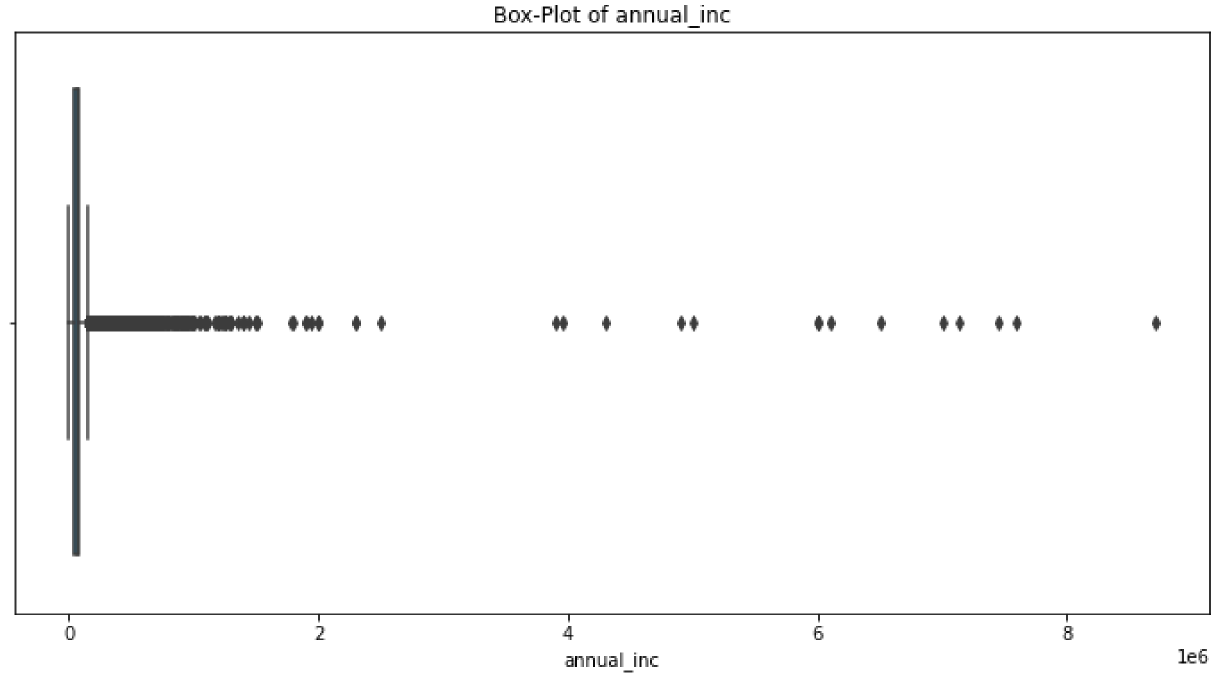
Out[ ]:
```
Index(['loan_amnt', 'int_rate', 'annual_inc', 'dti', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'issue_d_month',
       'issue_d_year', 'earliest_cr_line_month', 'earliest_cr_line_year'],
      dtype='object')
```

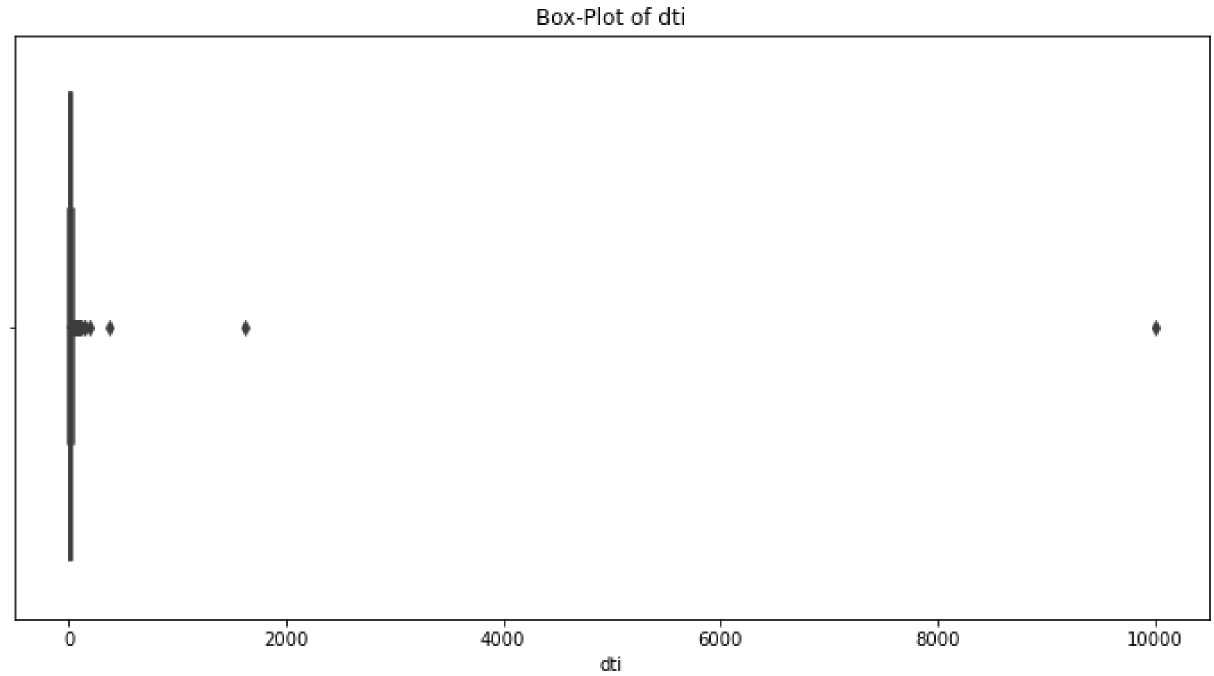
In [ ]:
```
def show_box_plot(feature: str) -> None:
    plt.figure(figsize=(12, 6))
    sns.boxplot(x = df[feature])
    plt.title(f'Box-Plot of {feature}')
    plt.show()
```

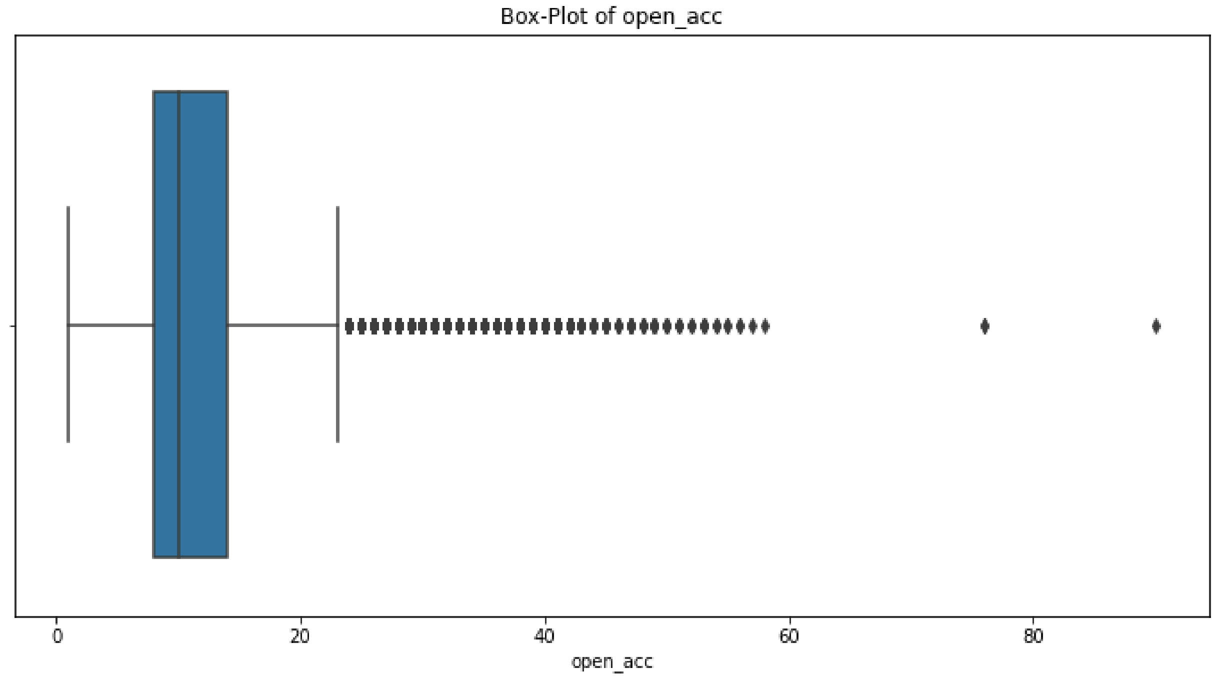
In [ ]:
```
for feature in num_cols:
    show_box_plot(feature)
```

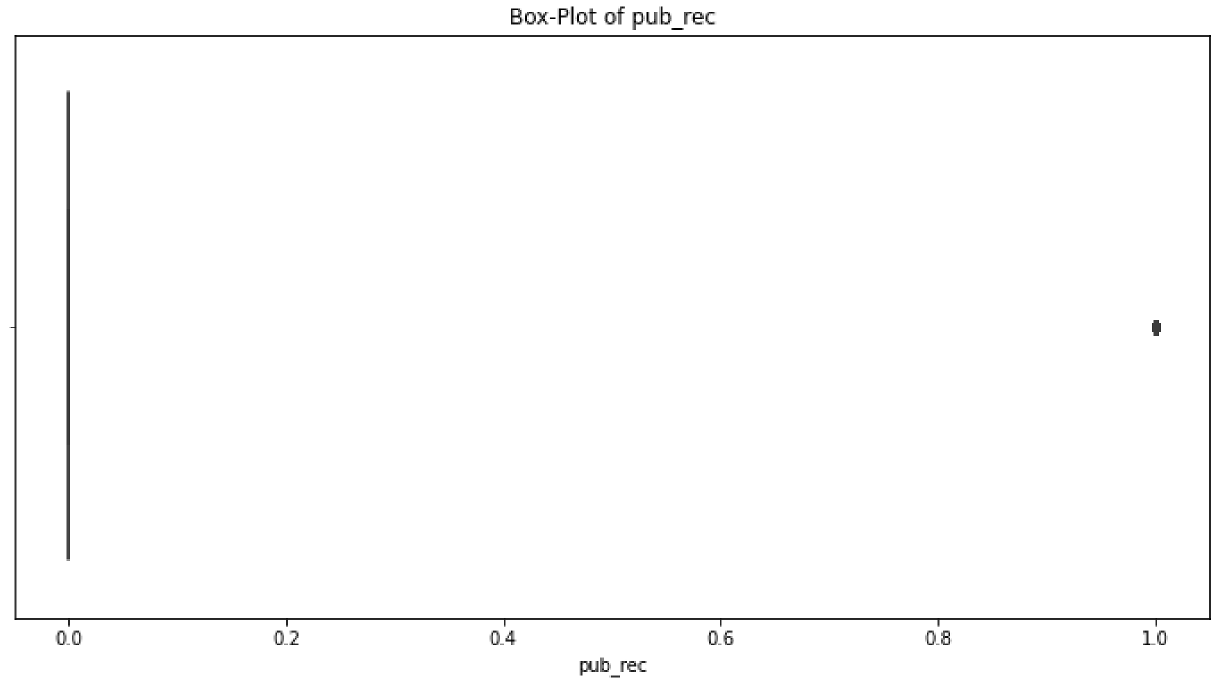

Box-Plot of loan_amnt
0
5000
10000
15000
20000
25000
30000
35000
40000
loan_amnt

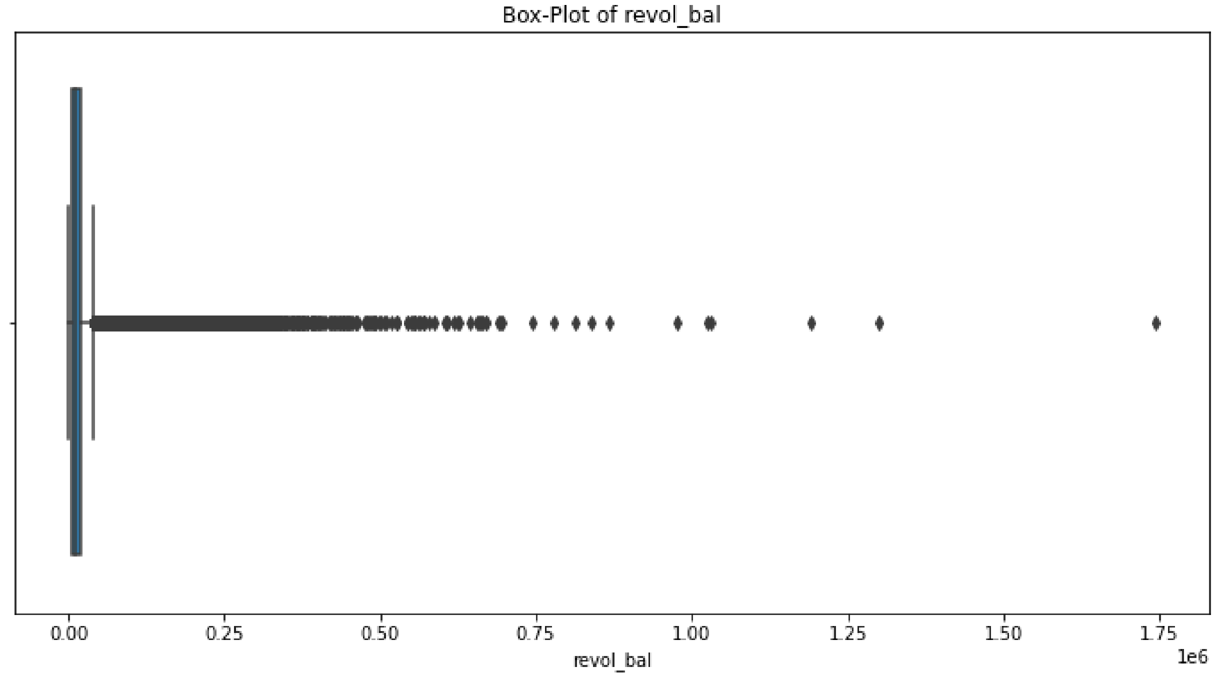

Box-Plot of int_rate
5
10
15
20
25
30
int_rate

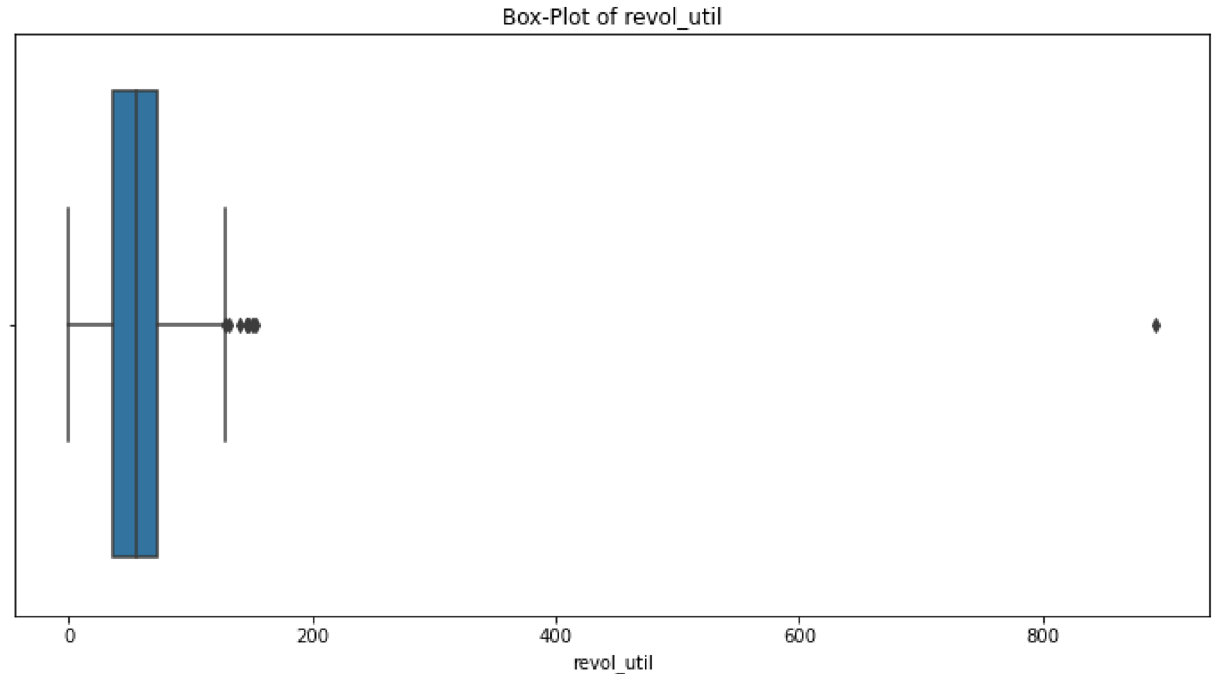

Box-Plot of annual_inc
0
2
4
6
8
1e6
annual_inc

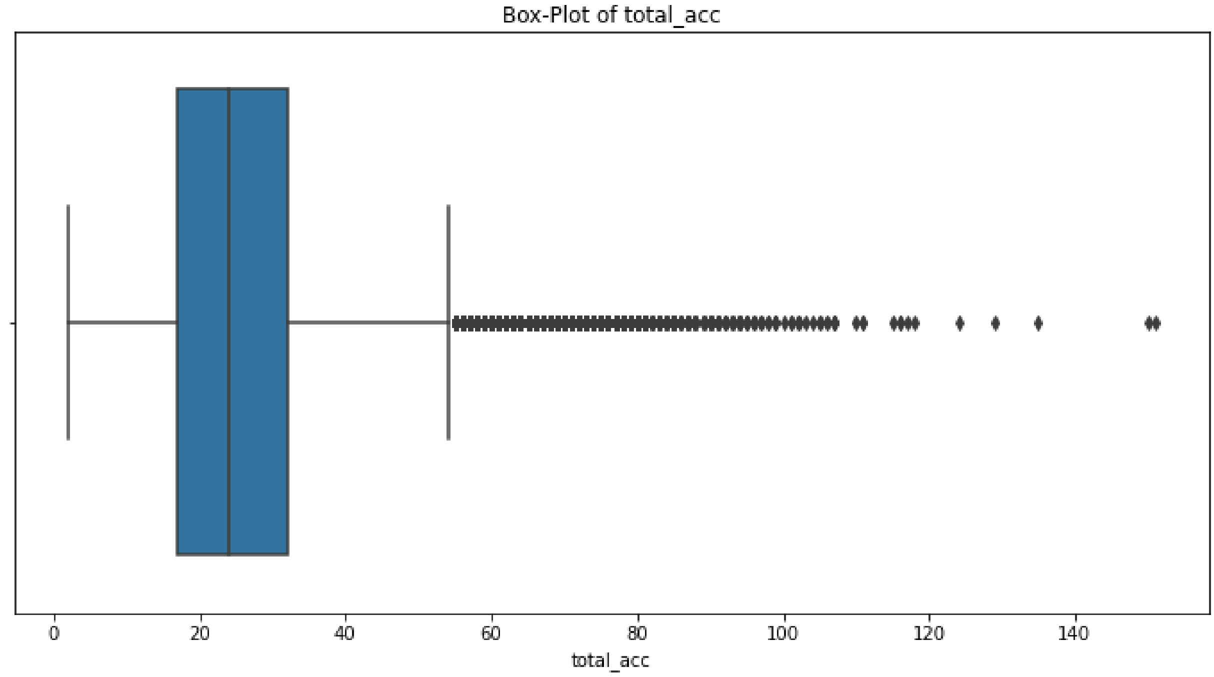

Box-Plot of dti
0
2000
4000
6000
8000
10000
dti

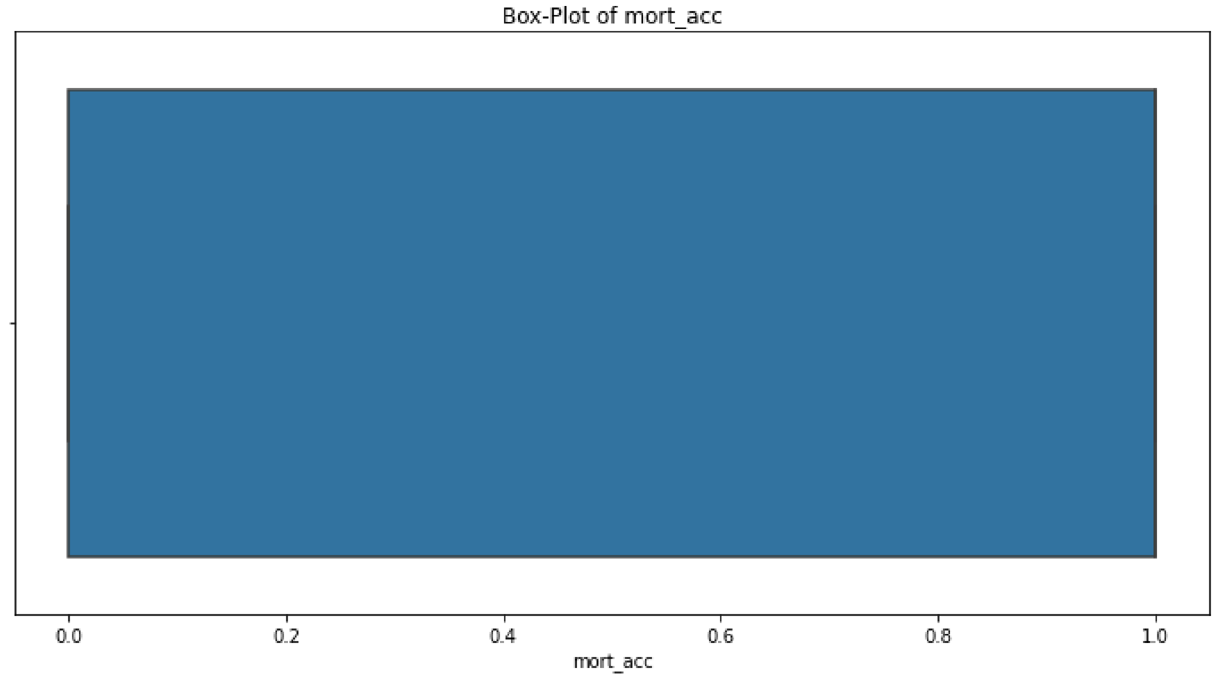

Box-Plot of open_acc
0
20
40
60
80
open_acc

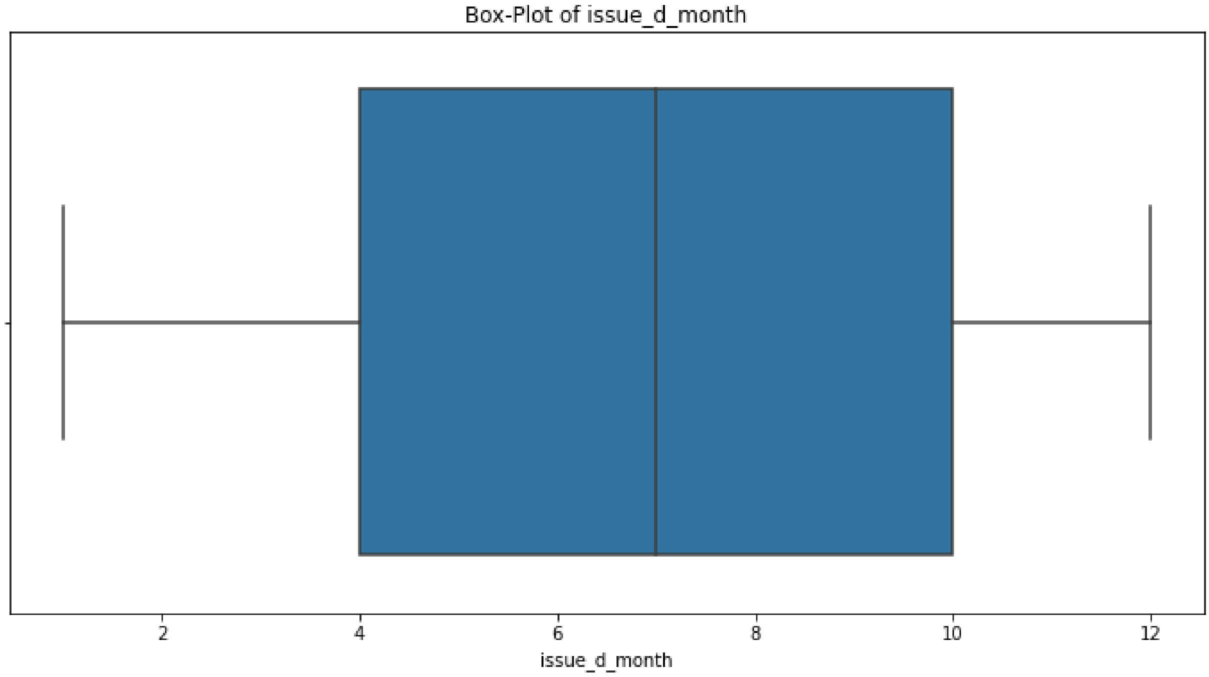

Box-Plot of pub_rec
0.0
0.2
0.4
0.6
0.8
1.0
pub_rec

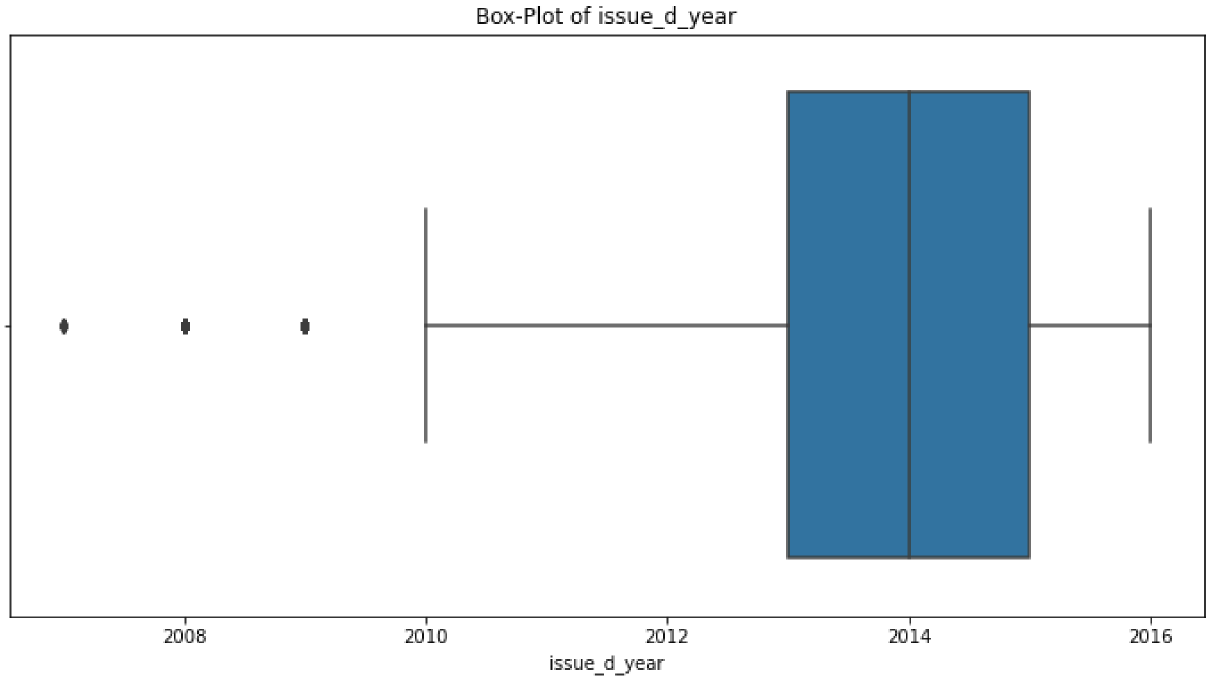

Box-Plot of revol_bal
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal

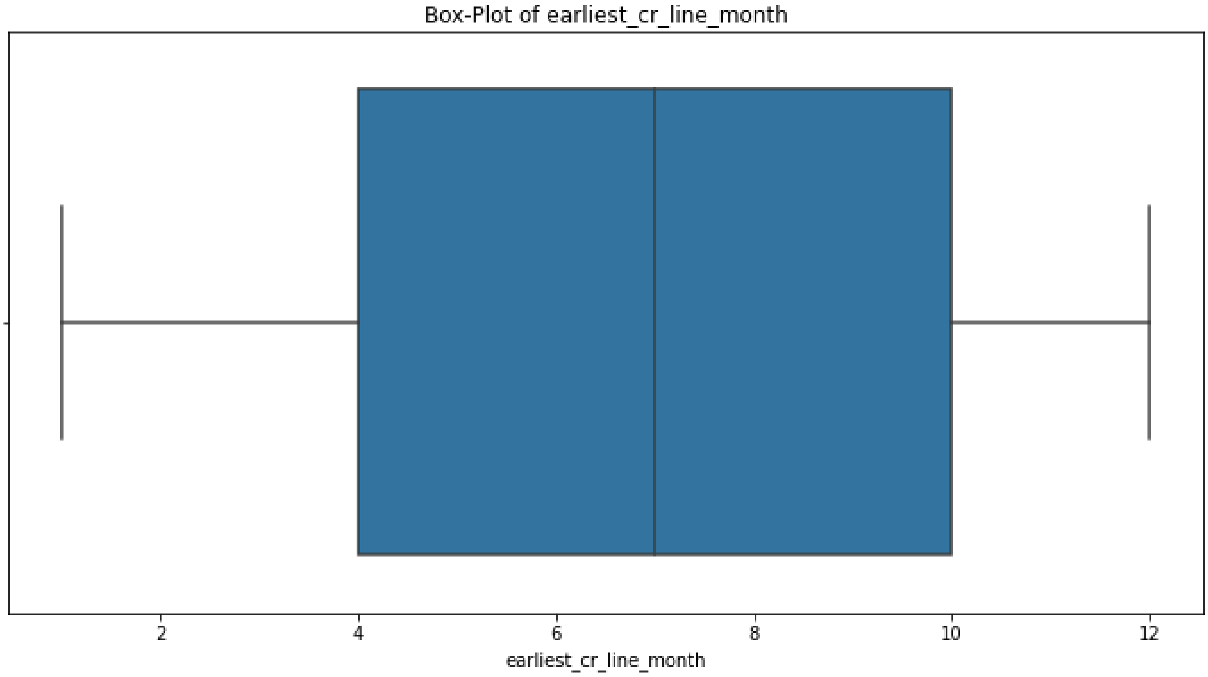

Box-Plot of revol_util
0
200
400
600
800
revol_util

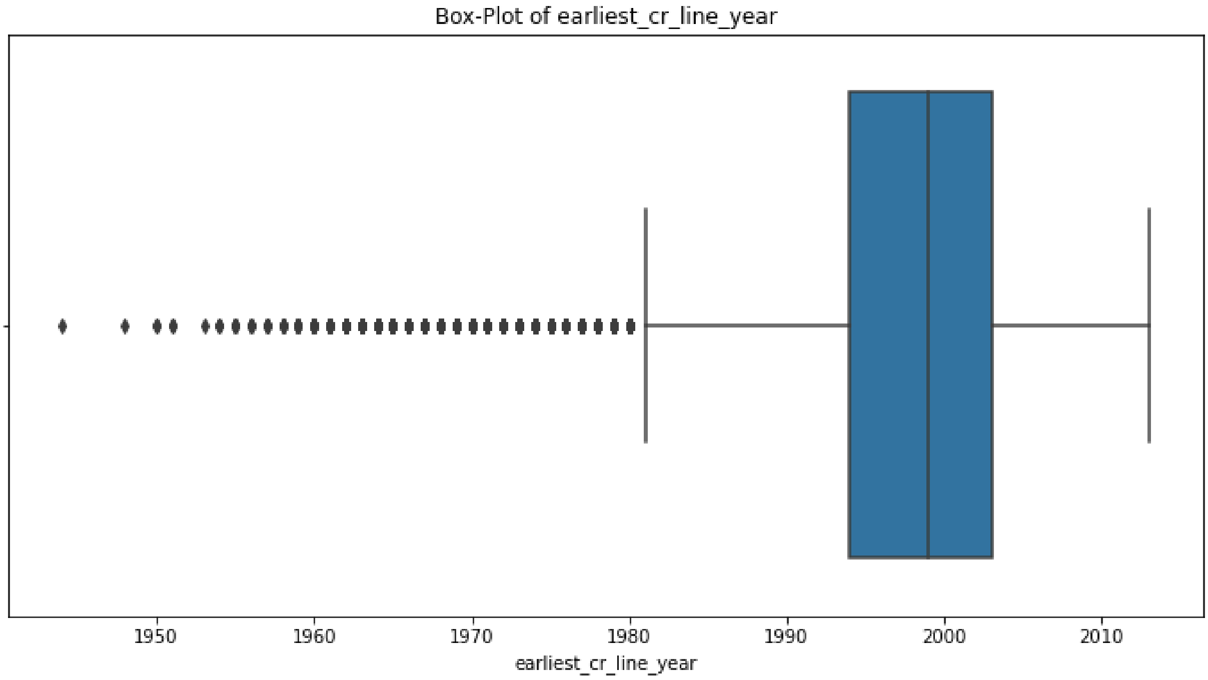

Box-Plot of total_acc
0
20
40
60
80
100
120
140
total_acc


Box-Plot of mort_acc
0.0
0.2
0.4
0.6
0.8
1.0
mort_acc


Box-Plot of issue_d_month
2
4
6
8
10
12
issue_d_month


Box-Plot of issue_d_year
2008
2010
2012
2014
2016
issue_d_year





- Boxplots have shown lot of outliers,
- let's treat them using z-score trimming method.

In [ ]:
```
for feature in num_cols:
    mean = df[feature].mean()
    std_dev = df[feature].std()

    lower_limit = mean - 3*std_dev
    upper_limit = mean + 3*std_dev

    df = df[ (df[feature] > lower_limit) & (df[feature] < upper_limit) ]
```

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

Out[ ]: (367495, 28)

Encoding

Label Encoding

In [ ]:
```
# 'term' column cleaning
term_mapping ={
    ' 36 months': 36,
    ' 60 months': 60
}

df['term'] = df['term'].map(term_mapping)
df['term'].unique()
```

Out[ ]: array([36, 60], dtype=int64)

In [ ]:
```
# 'grade' column encoding
grade_mapping = {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7}

df['grade'] = df['grade'].map(grade_mapping)
df['grade'].unique()
```

Out[ ]: array([2, 1, 3, 5, 4, 6, 7], dtype=int64)

In [ ]:
```
# 'sub_grade' column encoding
# encoding A1, B1, C1, ... -> 1 (because grade feature will take care of grades)
sub_grade_mapping = {ele: int(ele[1]) for ele in df['sub_grade'].values}

df['sub_grade'] = df['sub_grade'].map(sub_grade_mapping)
df['sub_grade'].unique()
```

Out[ ]: array([4, 5, 3, 2, 1], dtype=int64)

In [ ]:
```
# 'emp_length' column cleaning
emp_length_mapping = {
    '10+ years': 10, '4 years': 4, '< 1 year': 0, '6 years': 6, '9 years': 9,
    '2 years': 2, '3 years': 3, '8 years': 8, '7 years': 7, '5 years': 5, '1 year':
}

df['emp_length'] = df['emp_length'].map(emp_length_mapping)
df['emp_length'].unique()
```

Out[ ]: array([10, 4, 0, 6, 9, 2, 3, 7, 8, 5, 1], dtype=int64)

Mapping of Target Feature

In [ ]:
```
# loan_status column encoding
# loan_status is the target variable
loan_status_mapping = {'Fully Paid': 0, 'Charged Off': 1}

df['loan_status'] = df['loan_status'].map(loan_status_mapping)
df['loan_status'].unique()
```

Out[ ]: array([0, 1], dtype=int64)

Mapping of initial_list_status feature

- It contains 2 unique values so Label Encoding is a good choice.

In [ ]:
```
df['initial_list_status'].unique()
```

Out[ ]:
```
array(['w', 'f'], dtype=object)
```

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

Out[ ]:
```
array([0, 1], dtype=int64)
```

One-hot Encoding

In [ ]:
```
df.select_dtypes(include="object").nunique()
```

Out[ ]:
```
emp_title              162992
home_ownership              4
verification_status         3
purpose                    14
title                   37131
application_type            3
state                      52
zip_code                   10
dtype: int64
```

- emp_title, title, state contains lots of unique values, so I am dropping them to avoid curse of dimensionality.

In [ ]:
```
df.drop(['emp_title', 'title', 'state'], axis=1, inplace=True)
```

Performing one-hot encoding now for other features

In [ ]:
```
dummies = ['purpose', 'zip_code', 'verification_status', 'application_type', 'home_o
df = pd.get_dummies(df, columns=dummies, drop_first=True)
```

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

Out[ ]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_length	annual_inc	loan_status	dti	open_ac
0	10000.0	36	11.44	2	4	10	117000.0	0	26.24	16.
1	8000.0	36	11.99	2	5	4	65000.0	0	22.05	17.
2	15600.0	36	10.49	2	3	0	43057.0	0	12.79	13.
3	7200.0	36	6.49	1	2	6	54000.0	0	2.60	6.
4	24375.0	60	17.27	3	5	9	55000.0	1	33.95	13.

5 rows × 49 columns

Saving the data to a file

In [ ]:

```
df.to_csv("loan_tap_clean.csv", index=False)
```

Building Classification Model

In [ ]:

```
df = pd.read_csv('loan_tap_clean.csv')
df.head()
```

Out[ ]:

	loan_amnt	term	int_rate	grade	sub_grade	emp_length	annual_inc	loan_status	dti	open_ac
0	10000.0	36	11.44	2	4	10	117000.0	0	26.24	16.
1	8000.0	36	11.99	2	5	4	65000.0	0	22.05	17.
2	15600.0	36	10.49	2	3	0	43057.0	0	12.79	13.
3	7200.0	36	6.49	1	2	6	54000.0	0	2.60	6.
4	24375.0	60	17.27	3	5	9	55000.0	1	33.95	13.

5 rows × 49 columns

Utility functions

In [ ]:

```
def show_roc_curve(y_true, y_pred_proba):
    fpr, tpr, thresholds = metrics.roc_curve(y_true, y_pred_proba)
    roc_area_score = metrics.auc(fpr, tpr)

    plt.figure(figsize=(8, 6))

    plt.plot(fpr, tpr, label = f'Logistic Regression (ROC Area Score: {round(roc_are
    plt.plot(fpr, fpr, 'r--', label = "Random Model (ROC Area Score: 0.5)")

    # Axis labels
    plt.xlabel("False Positive rate")
    plt.ylabel("True Positive Rate")

    # Other properties
    plt.grid()
    plt.title("ROC curve")
    plt.legend(loc = "lower right")

    plt.show()
```

In [ ]:

```
def show_pr_curve(y_true, y_pred_proba):
    precisions, recalls, thresholds = metrics.precision_recall_curve(y_true, y_pred_
    pr_area_score = metrics.auc(recalls, precisions)

    plt.figure(figsize=(8, 6))

    plt.plot(recalls, precisions, label = f'Logistic Regression (PR score: {round(pr
    plt.plot(recalls, 1 - recalls, 'r--', label = "Random Model (ROC score: 0.5)")

    # Axis labels
    plt.xlabel("Recalls")
```

```
    plt.ylabel("Precisions")

    # Other properties
    plt.title("PR curve")
    plt.legend(loc = "lower left")
    plt.grid()

    plt.show()
```

In [ ]:

```
def show_pr_trade_off_curve(y_true, y_pred_proba):
    precisions, recalls, thresholds = metrics.precision_recall_curve(y_true, y_pred_
    boundary = thresholds.shape[0]

    plt.figure(figsize=(8, 6))

    # precision plot
    plt.plot(thresholds, precisions[:boundary], 'b--', label = 'precision')

    # recall plot
    plt.plot(thresholds, recalls[:boundary], 'r--', label = 'recall')

    # Axis labels
    plt.xlabel("Threshold")
    plt.xlabel("Precision/Recall")

    # Other properties
    plt.legend()
    plt.grid()
    plt.title('Precision-Recall Trade off')

    plt.show()
```

In [ ]:

```
def show_metrics_summary(y_true, y_pred, y_pred_proba):
    print(metrics.classification_report(y_true, y_pred))
    print('-------------------------------------------------------------------------

    cm = metrics.confusion_matrix(y_true, y_pred)
    print(cm)
    print('-------------------------------------------------------------------------
    print(cm*100/cm.sum())
    print('-------------------------------------------------------------------------

    metrics.ConfusionMatrixDisplay(cm).plot()
    plt.show()

    show_roc_curve(y_true, y_pred_proba)
    show_pr_curve(y_true, y_pred_proba)
    show_pr_trade_off_curve(y_true, y_pred_proba)
```

Logistic Regression Model -1

- without class balancing, regularizer and multi-collinearity check

X and y

In [ ]:

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

train-test split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10, test_size

print(X_train.shape)
print(X_test.shape)
```

```
(220497, 48)
(146998, 48)
```

Scaling

In [ ]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Classification Model

In [ ]:
```
model = LogisticRegression(max_iter=1000)  # Adjust iterations based on need

model.fit(X_train, y_train)
```

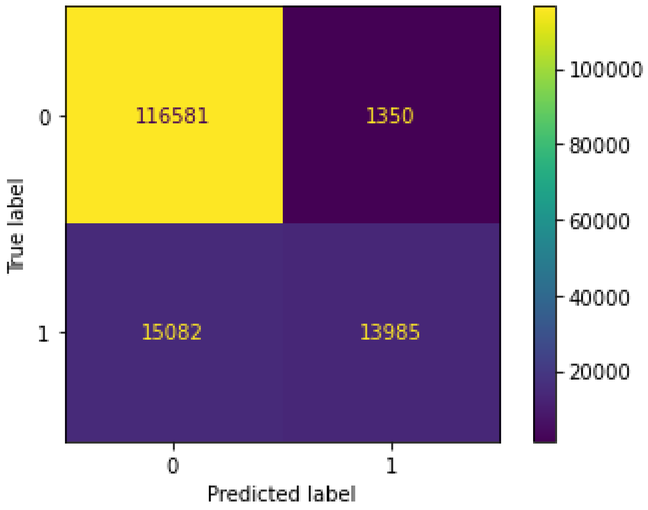
Out[ ]:
```
▾       LogisticRegression
LogisticRegression(max_iter=1000)
```

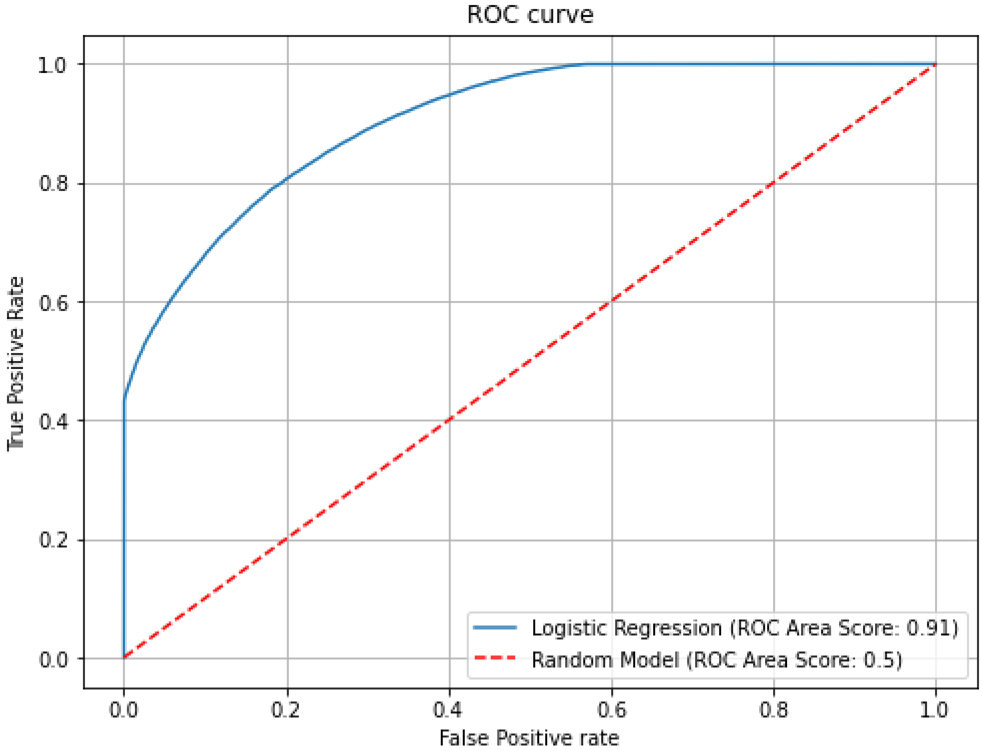
Model Performace

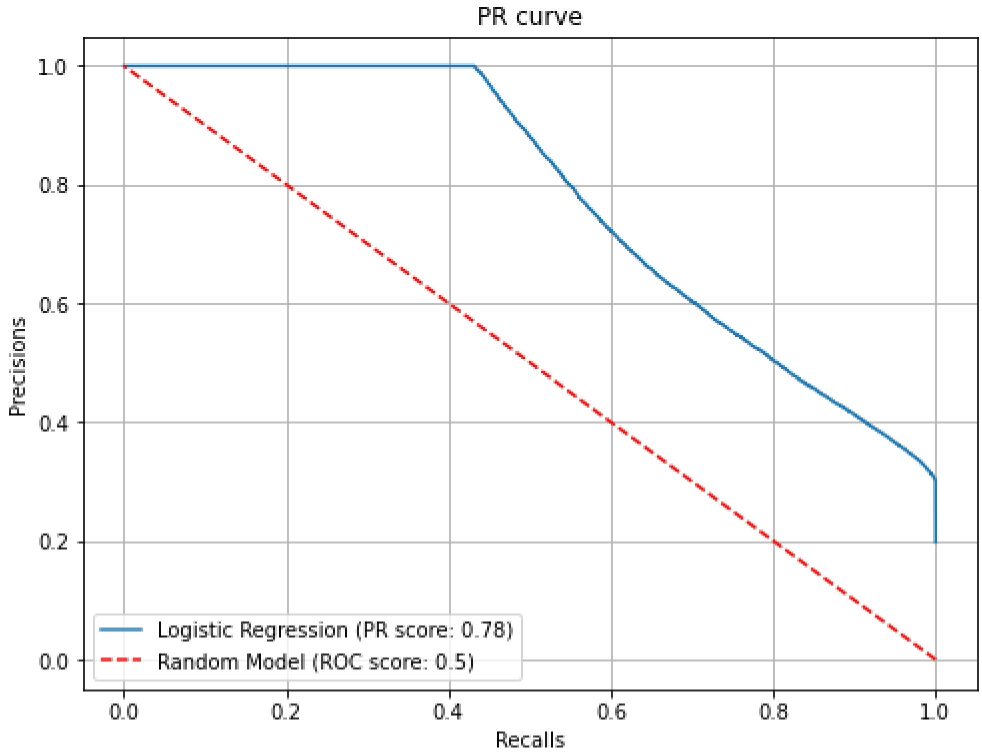
In [ ]:
```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

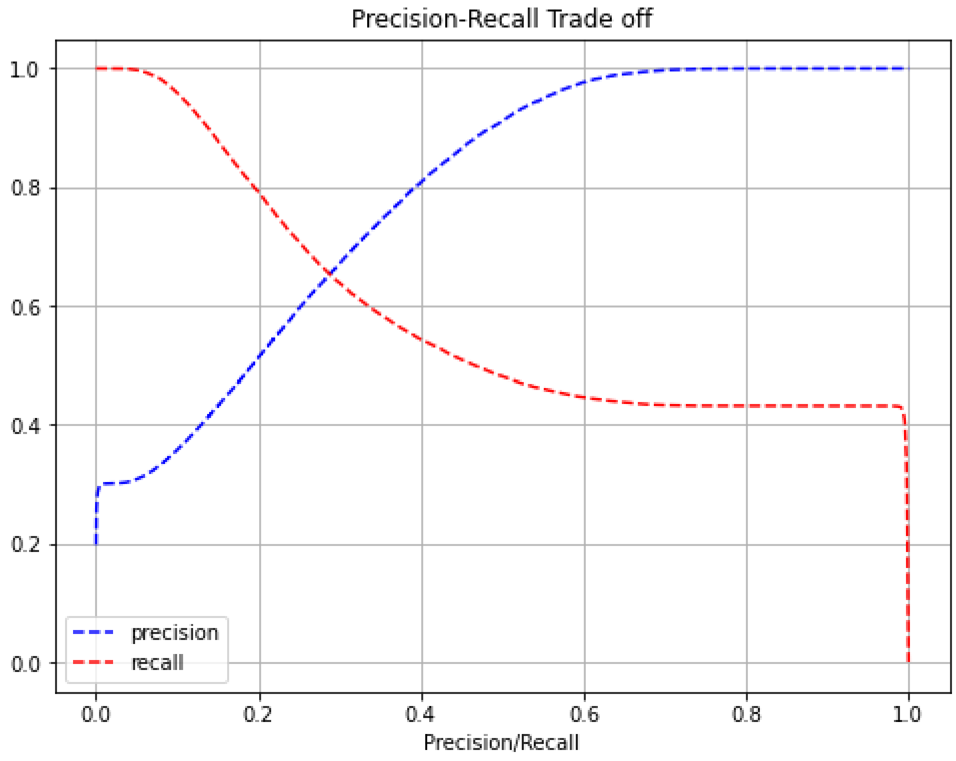
In [ ]:
```
show_metrics_summary(y_test, y_pred, y_pred_proba)
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.93    117931
           1       0.91      0.48      0.63     29067

    accuracy                           0.89    146998
   macro avg       0.90      0.73      0.78    146998
weighted avg       0.89      0.89      0.87    146998

-------------------------------------------------------------------------------------
--
[[116581   1350]
 [ 15082  13985]]
-------------------------------------------------------------------------------------
--
[[79.30788174  0.91837984]
 [10.26000354  9.51373488]]
-------------------------------------------------------------------------------------
--
```


116581
1350
15082
13985
True label
Predicted label
0
1
100000
80000
60000
40000
20000


ROC curve
True Positive Rate
False Positive rate
Logistic Regression (ROC Area Score: 0.91)
Random Model (ROC Area Score: 0.5)


PR curve
Precisions
Recalls
Logistic Regression (PR score: 0.78)
Random Model (ROC score: 0.5)



In [ ]:
```
metrics.fbeta_score(y_test, y_pred, beta=2)
```

Out[ ]: 0.5313328723509343

Insights:-

1. Metrics summary of Logistic Regression without handling balancing, regularization and multi-collinearity check:
   - Accuracy: 0.89
   - Precision: 0.91
   - Recall: 0.48
   - F1-score: 0.63
   - ROC-AUC: 0.91
   - PR-AUC: 0.78
   - fbeta(beta=2): 0.53
2. Confusion Matrix:
   - TN: 79.3%
   - FP: 0.91%
   - FN: 10.26%
   - TP: 9.51%

Multi-Collinearity check and its removal

In [ ]:
```
X = df.drop('loan_status', axis=1)
y = df['loan_status']
```

In [ ]:
```
def remove_features_by_vif(X: pd.DataFrame, vif_threshold = 10):
    features = X.columns.to_list()
```

```
    dropped_features = list()

    while True:
        # calculate vifs
        vifs = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
        vifs = list(zip(features, vifs))

        # sort vifs
        vifs.sort(key= lambda x: x[1], reverse=True)

        # top vif
        top_vif = vifs[0][1]
        top_vif_feature = vifs[0][0]

        if top_vif < vif_threshold:
            break

        # drop top vif feature
        dropped_features.append(top_vif_feature)
        X.drop(columns=[top_vif_feature], inplace=True)
        features.remove(top_vif_feature)

    return dropped_features
```

In [ ]:

```
dropped_features = remove_features_by_vif(X, vif_threshold= 10)
dropped_features
```

Out[ ]:

```
['issue_d_year',
 'earliest_cr_line_year',
 'int_rate',
 'application_type_INDIVIDUAL',
 'term',
 'purpose_debt_consolidation',
 'open_acc']
```

Logistic Regression after removal of multi-collinearity

train-test split

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10, test_size

print(X_train.shape)
print(X_test.shape)
```

```
(220497, 41)
(146998, 41)
```

Scaling

In [ ]:

```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Classification Model

In [ ]:

```
model = LogisticRegression(max_iter=1000)  # Adjust iterations based on need

model.fit(X_train, y_train)
```

Out[ ]:

```
▾         LogisticRegression
LogisticRegression(max_iter=1000)
```

Model Performace

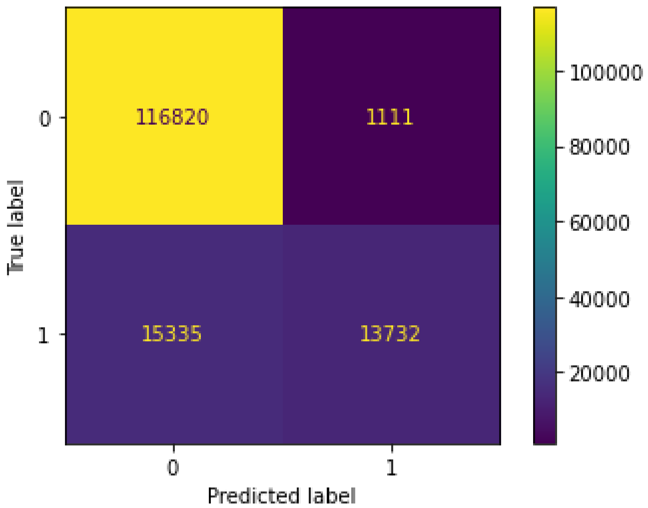
In [ ]:

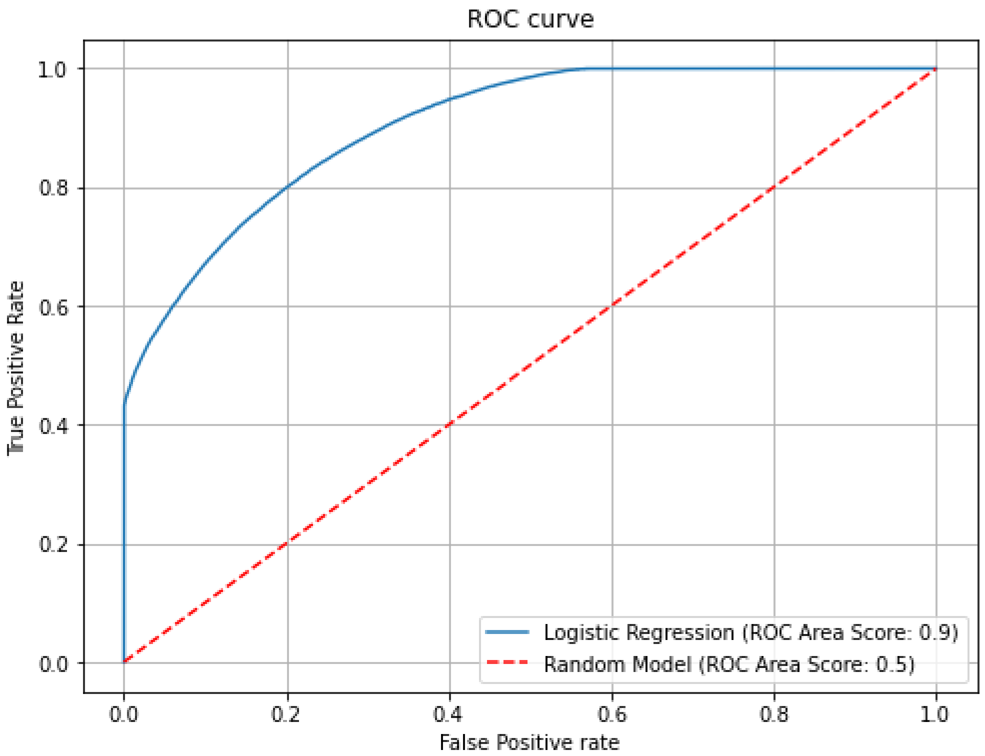
```python
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

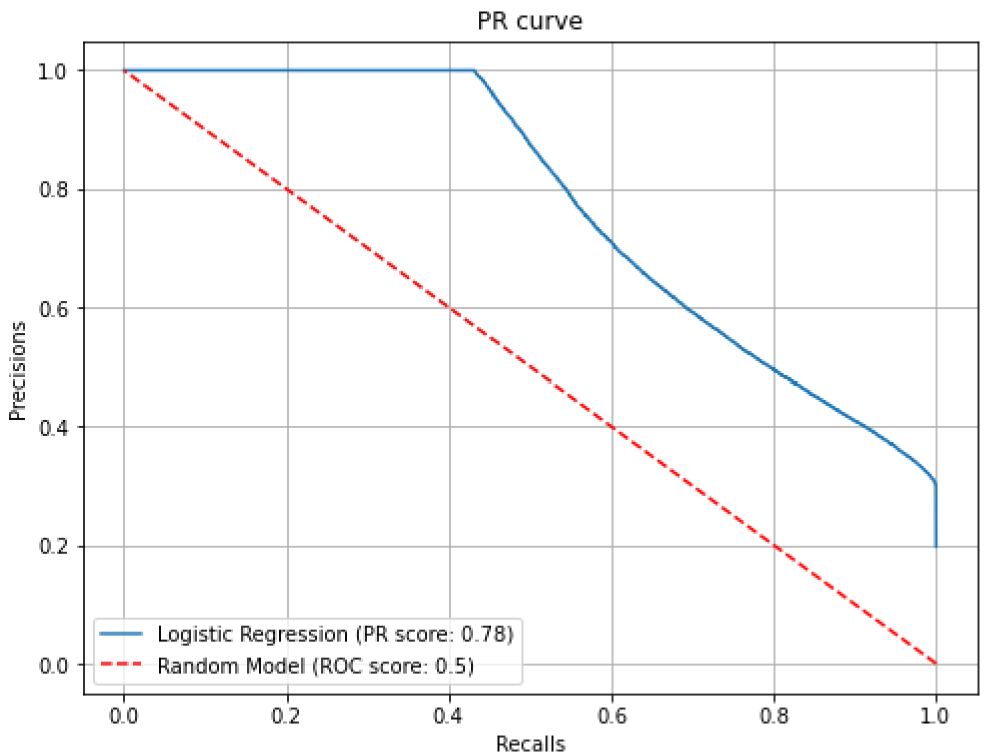
In [ ]:

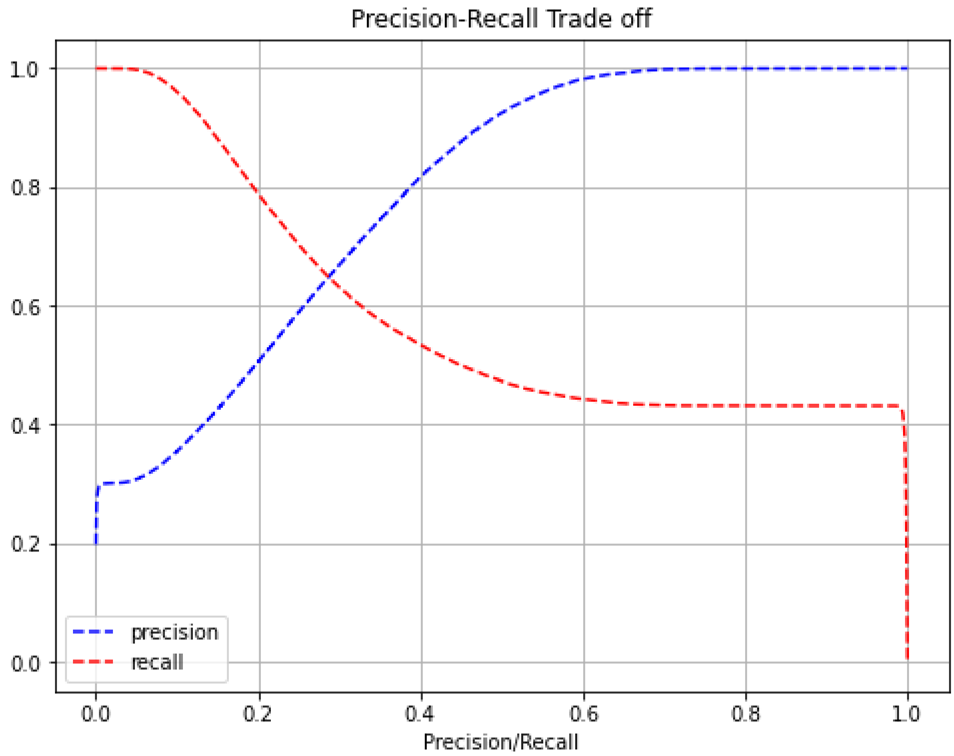
```python
show_metrics_summary(y_test, y_pred, y_pred_proba)
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93    117931
           1       0.93      0.47      0.63     29067

    accuracy                           0.89    146998
   macro avg       0.90      0.73      0.78    146998
weighted avg       0.89      0.89      0.87    146998

-----------------------------------------------------------------------------------
--
[[116820   1111]
 [ 15335  13732]]
-----------------------------------------------------------------------------------
--
[[79.47046899  0.7557926 ]
 [10.43211472  9.3416237 ]]
-----------------------------------------------------------------------------------
--
```




ROC curve
True Positive Rate
False Positive rate
Logistic Regression (ROC Area Score: 0.9)
Random Model (ROC Area Score: 0.5)


PR curve
Precisions
Recalls
Logistic Regression (PR score: 0.78)
Random Model (ROC score: 0.5)



In [ ]:

```
metrics.fbeta_score(y_test, y_pred, beta=2)
```

Out[ ]: 0.5236784098969576

Insights:-

1. Metrics summary of Logistic Regression after multi-collinearity check:

- Accuracy: 0.89
- Precision: 0.93
- Recall: 0.47
- F1-score: 0.63
- ROC-AUC: 0.9
- PR-AUC: 0.78
- fbeta(beta=2): 0.52

2. Confusion Matrix:

- TN: 79.47%
- FP: 0.75%
- FN: 10.43%
- TP: 9.34%

Handling Imbalance using class-weights

X and y

In [ ]:

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

Drop non-important features

In [ ]:
```
X.drop(columns=dropped_features, inplace=True)
```

train-test split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10, test_size

print(X_train.shape)
print(X_test.shape)
```

```
(220497, 41)
(146998, 41)
```

Scaling

In [ ]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Classification Model

In [ ]:
```
model = LogisticRegression(max_iter=1000, class_weight='balanced')  # Adjust iterati

model.fit(X_train, y_train)
```

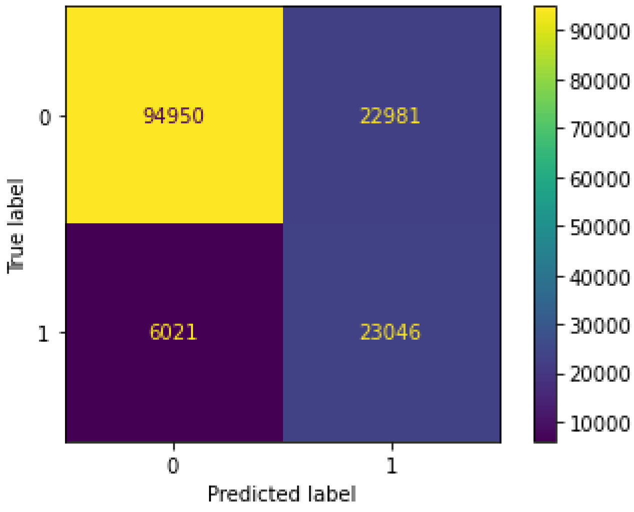
Out[ ]:
```
▾                    LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=1000)
```

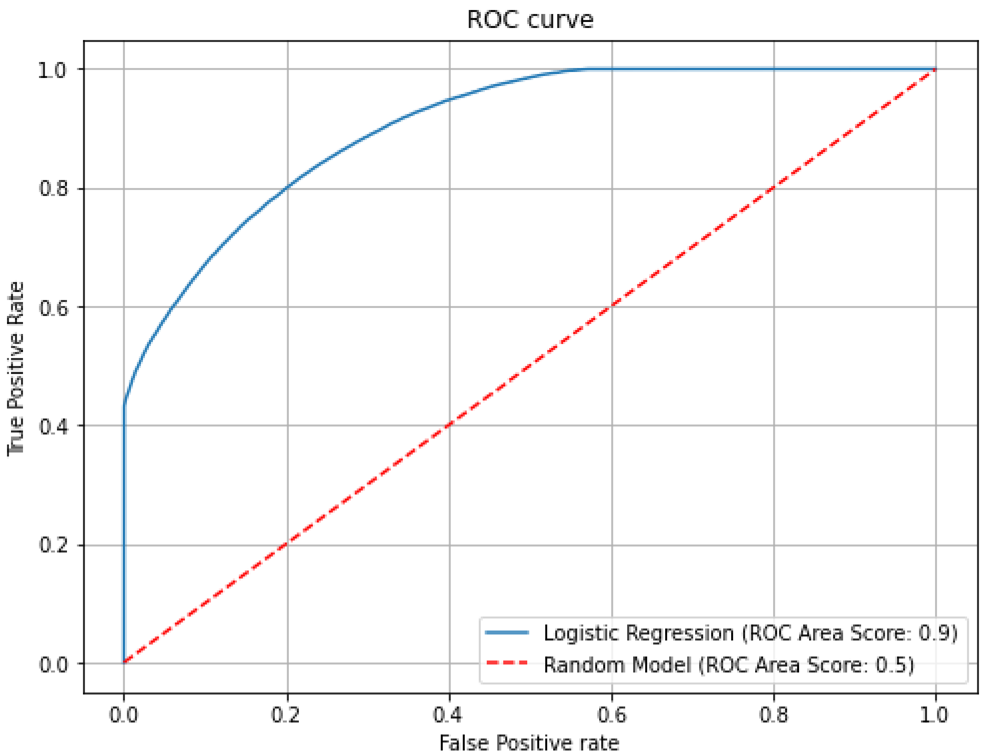
Model Performace

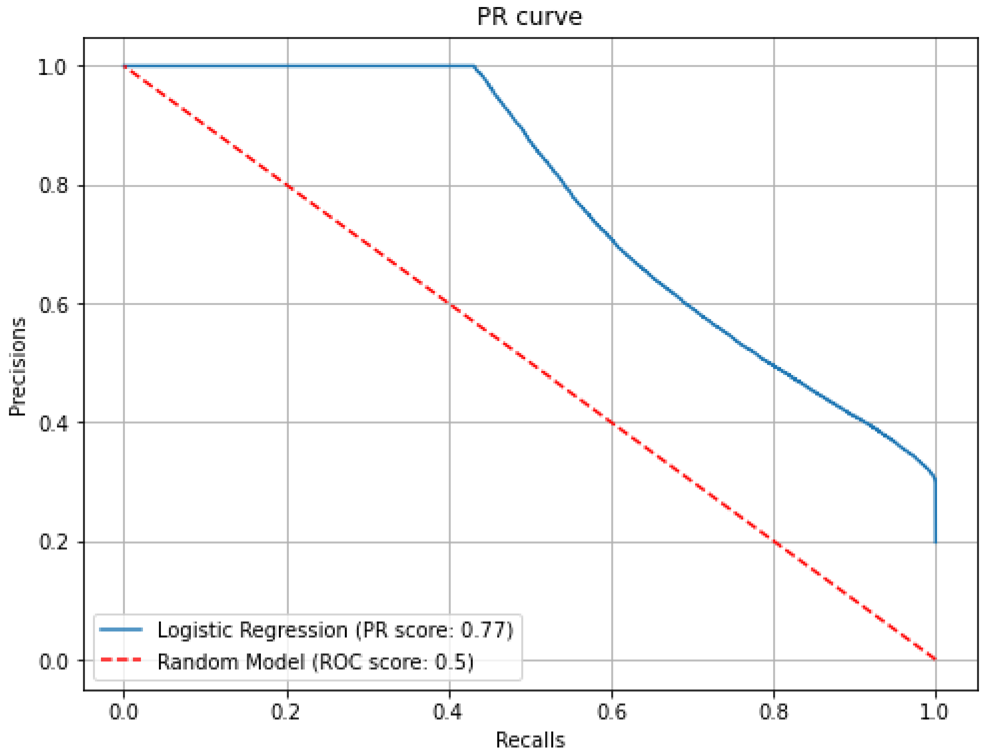
In [ ]:
```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

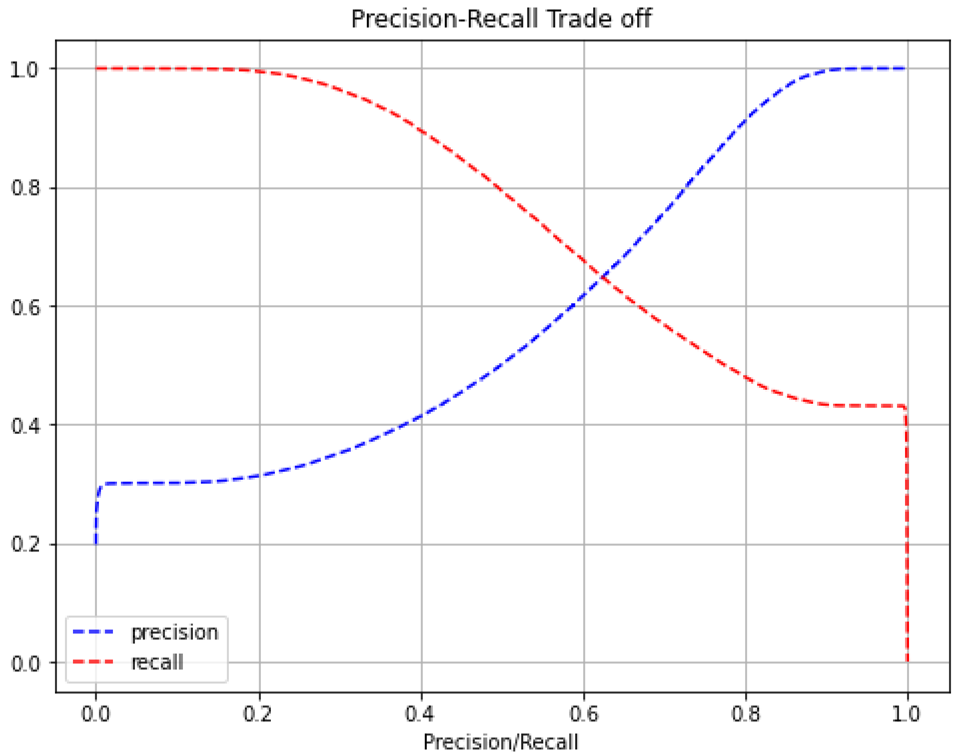
In [ ]:
```
show_metrics_summary(y_test, y_pred, y_pred_proba)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87    117931
           1       0.50      0.79      0.61     29067

    accuracy                           0.80    146998
   macro avg       0.72      0.80      0.74    146998
weighted avg       0.85      0.80      0.82    146998

--------------------------------------------------------------------------------
--
[[94950 22981]
 [ 6021 23046]]
--------------------------------------------------------------------------------
--
[[64.59271555 15.63354603]
 [ 4.09597409 15.67776432]]
--------------------------------------------------------------------------------
--
```


94950
22981
6021
23046
True label
Predicted label
0
1
90000
80000
70000
60000
50000
40000
30000
20000
10000


ROC curve
True Positive Rate
False Positive rate
Logistic Regression (ROC Area Score: 0.9)
Random Model (ROC Area Score: 0.5)


PR curve
Precisions
Recalls
Logistic Regression (PR score: 0.77)
Random Model (ROC score: 0.5)



In [ ]:
```
metrics.fbeta_score(y_test, y_pred, beta=2)
```

Out[ ]: 0.7100033888906004

Insights:-

1. Metrics summary of Logistic Regression after handling balancing and multi-collinearity check:
    - Accuracy: 0.8
    - Precision: 0.5
    - Recall: 0.79
    - F1-score: 0.61
    - ROC-AUC: 0.9
    - PR-AUC: 0.77
    - fbeta(beta=2): 0.71
2. Confusion Matrix:
    - TN: 64.59%
    - FP: 15.63%
    - FN: 4.09%
    - TP: 15.67%

Regularization (L2 regularizer) and Cross Validation

X and y

In [ ]:
```
X = df.drop('loan_status', axis=1)
y = df['loan_status']
```

Drop non-important features

In [ ]:
```
X.drop(columns=dropped_features, inplace=True)
```

train-val-test split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10, test_size
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, random_state=10,

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(183747, 41)
(91874, 41)
(91874, 41)
```

scaling

In [ ]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

cross-validation

In [ ]:
```
lamdas = [0.01, 0.1, 1, 10]
train_scores = list()
val_scores = list()

for lamda in lamdas:
    model = LogisticRegression(max_iter=1000, class_weight='balanced', penalty='l2',

    # train the model
    model.fit(X_train, y_train)

    # training score
    y_train_pred = model.predict(X_train)
    train_scores.append(metrics.recall_score(y_train, y_train_pred))

    # validation score
    y_val_pred = model.predict(X_val)
    val_scores.append(metrics.recall_score(y_val, y_val_pred))

print(lamdas)
print(train_scores)
print(val_scores)
```

```
[0.01, 0.1, 1, 10]
[0.7946626719928983, 0.7946071904127829, 0.794329782512206, 0.7925821127385708]
[0.7881546476556074, 0.7881546476556074, 0.7878256100904854, 0.7863997806416233]
```

- Not much difference in using different lambdas
- let use lambda = 0.01

Test score/summary

Classification Model

In [ ]:
```
model = LogisticRegression(max_iter=1000, class_weight='balanced', penalty='l2', C=

model.fit(X_train, y_train)
```

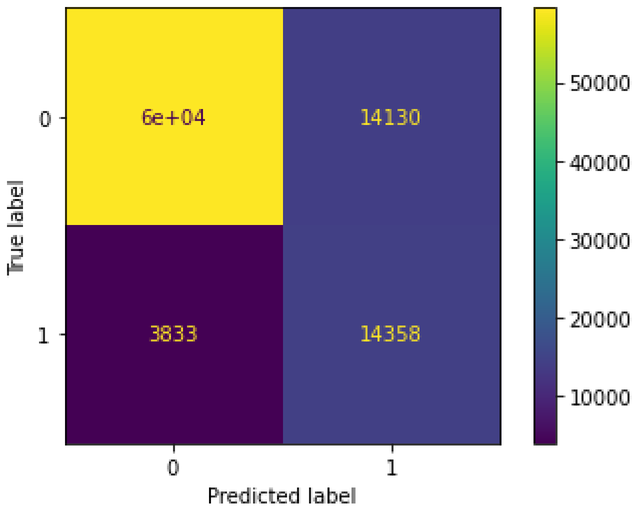
Out[ ]:
```
LogisticRegression

LogisticRegression(C=100.0, class_weight='balanced', max_iter=1000)
```

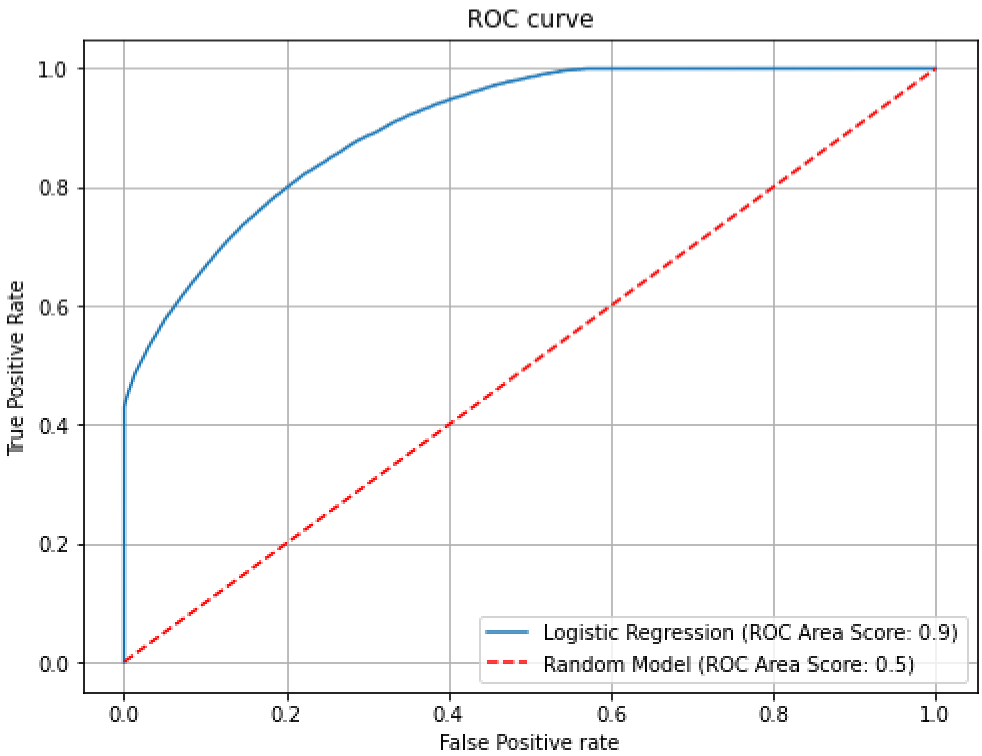
Model Performace

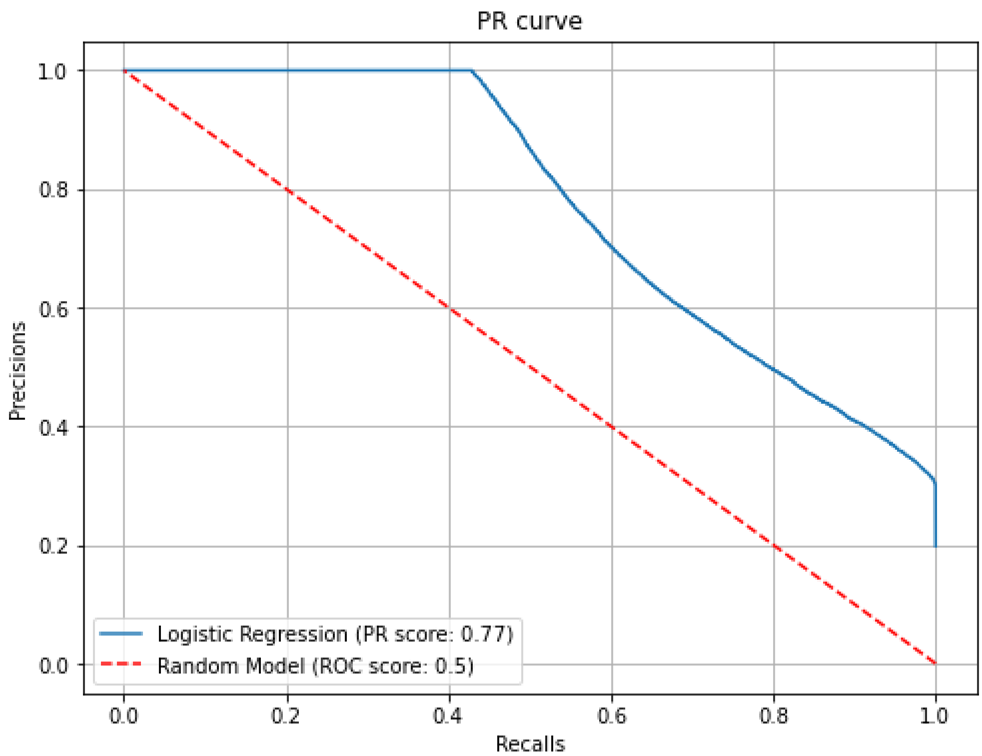
In [ ]:
```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

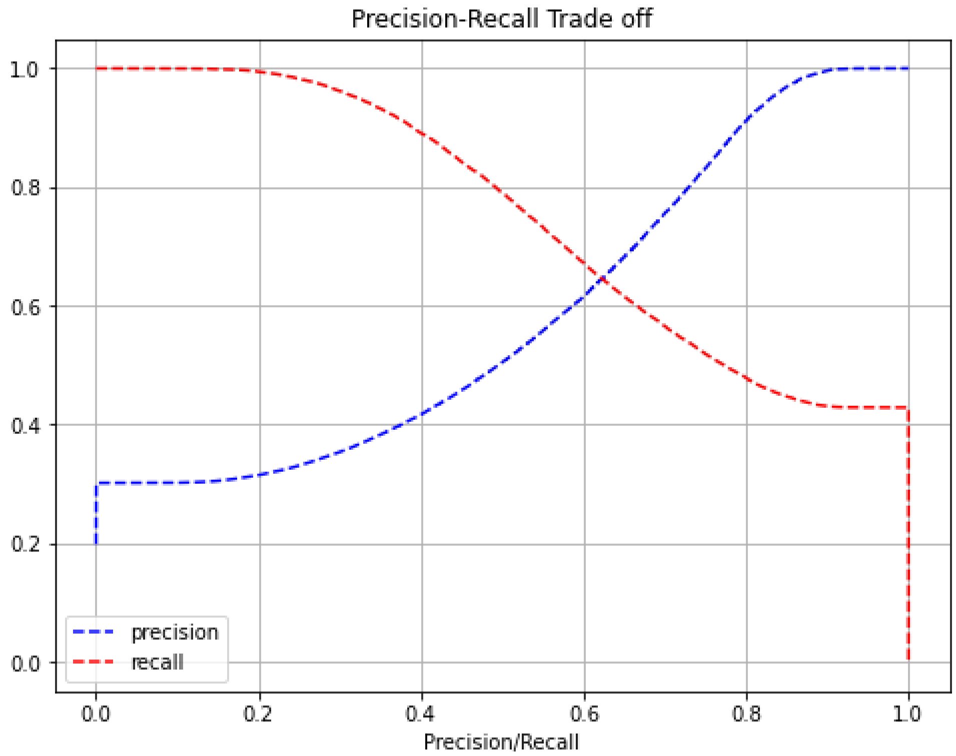
In [ ]:
```
show_metrics_summary(y_test, y_pred, y_pred_proba)
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87     73683
           1       0.50      0.79      0.62     18191

    accuracy                           0.80     91874
   macro avg       0.72      0.80      0.74     91874
weighted avg       0.85      0.80      0.82     91874

-----------------------------------------------------------------------------------
--
[[59553 14130]
 [ 3833 14358]]
-----------------------------------------------------------------------------------
--
[[64.82029736 15.37975924]
 [ 4.1720182  15.6279252 ]]
-----------------------------------------------------------------------------------
--
```




ROC curve
True Positive Rate
False Positive rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (ROC Area Score: 0.9)
Random Model (ROC Area Score: 0.5)


PR curve
Precisions
Recalls
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (PR score: 0.77)
Random Model (ROC score: 0.5)



In [ ]:
```
metrics.fbeta_score(y_test, y_pred, beta=2)
```

Out[ ]: 0.7090230316438193

Insights:-

1. Metrics summary of Logistic Regression after handling balancing, multi-collinearity check, cross-validation and regularization:
   - Accuracy: 0.8
   - Precision: 0.5
   - Recall: 0.79
   - F1-score: 0.62
   - ROC-AUC: 0.9
   - PR-AUC: 0.77
   - fbeta(beta=2): 0.71
2. Confusion Matrix:
   - TN: 64.82%
   - FP: 15.37%
   - FN: 4.17%
   - TP: 15.62%

Logistic Regression Model - (final)

X and y

In [ ]:
```
X = df.drop('loan_status', axis=1)
y = df['loan_status']
```

Drop non-important features

In [ ]:
```
X.drop(columns=dropped_features, inplace=True)
```

train-test split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10, test_size

print(X_train.shape)
print(X_test.shape)
```

```
(220497, 41)
(146998, 41)
```

Scaling

In [ ]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Classification Model

In [ ]:
```
model = LogisticRegression(max_iter=1000, class_weight='balanced', penalty='l2', C=1

model.fit(X_train, y_train)
```

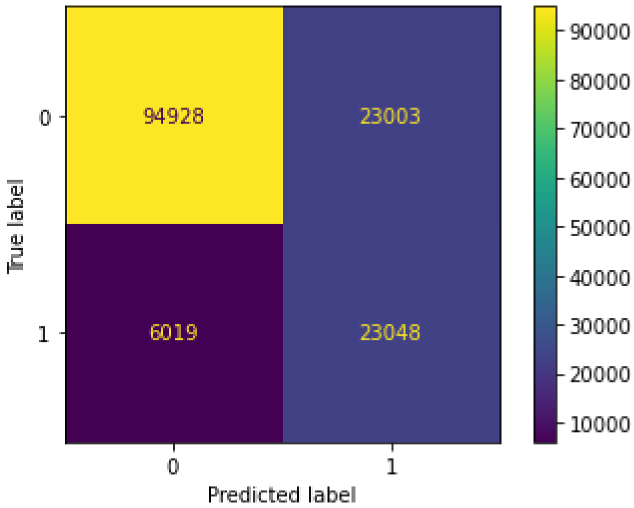
Out[ ]:
```
▾                     LogisticRegression
LogisticRegression(C=100.0, class_weight='balanced', max_iter=1000)
```

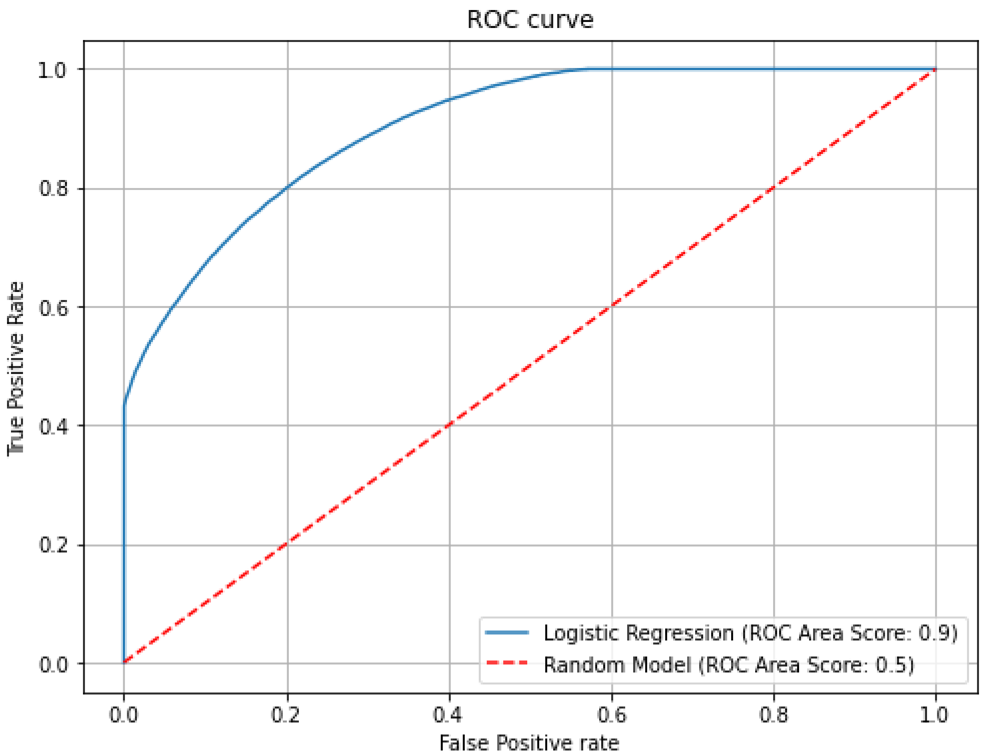
Model Performace

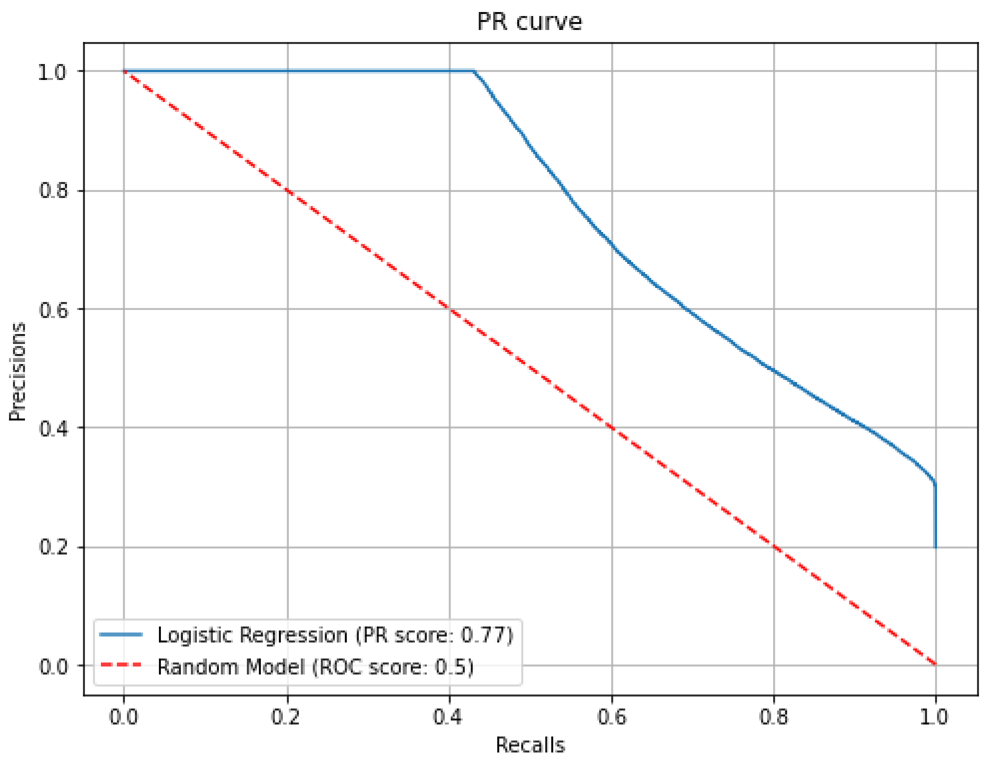
In [ ]:
```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

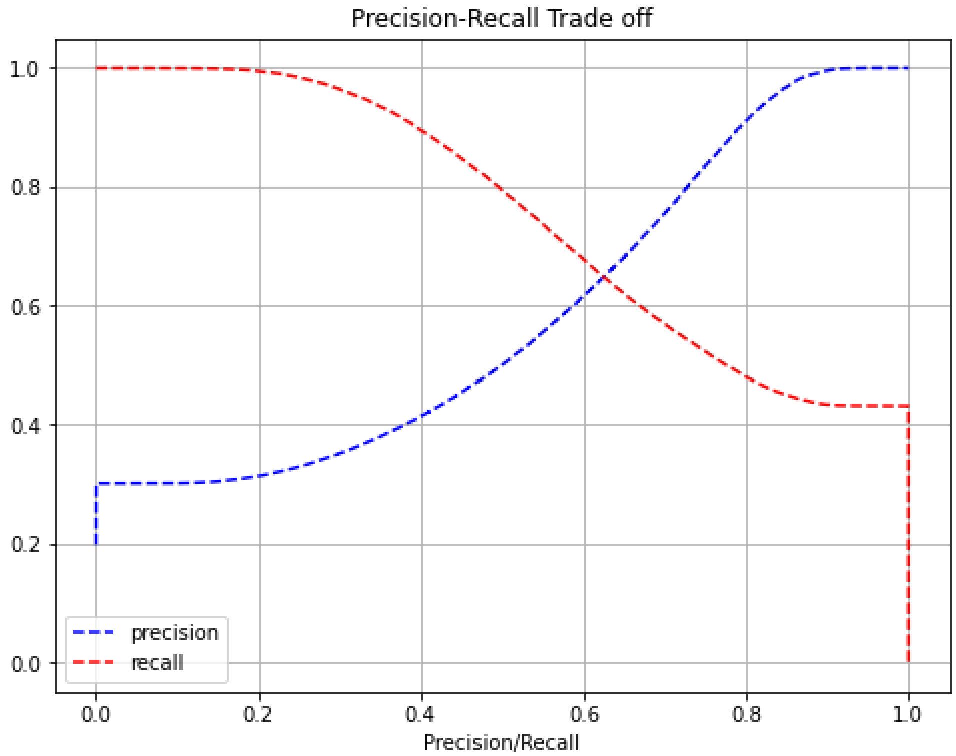
In [ ]:
```
show_metrics_summary(y_test, y_pred, y_pred_proba)
```

```
              precision    recall  f1-score   support

           0       0.94      0.80      0.87    117931
           1       0.50      0.79      0.61     29067

    accuracy                           0.80    146998
   macro avg       0.72      0.80      0.74    146998
weighted avg       0.85      0.80      0.82    146998

-----------------------------------------------------------------------------------
--
[[94928 23003]
 [ 6019 23048]]
-----------------------------------------------------------------------------------
--
[[64.57774936 15.64851222]
 [ 4.09461353 15.67912489]]
-----------------------------------------------------------------------------------
--
```


94928
23003
6019
23048
True label
0
1
Predicted label
0
1
90000
80000
70000
60000
50000
40000
30000
20000
10000


ROC curve
True Positive Rate
False Positive rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (ROC Area Score: 0.9)
Random Model (ROC Area Score: 0.5)


PR curve
Precisions
Recalls
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (PR score: 0.77)
Random Model (ROC score: 0.5)



In [ ]:

```
metrics.fbeta_score(y_test, y_pred, beta=2)
```

Out[ ]: `0.7099600170035547`

comparing weights

In [ ]:

```
feature_with_weights = list(zip(X.columns, model.coef_.flatten().tolist()))
feature_with_weights.sort(key=lambda x: x[1], reverse=True)

feature_with_weights[:10]
```

Out[ ]:

```
[('zip_code_93700', 20.507644059351133),
 ('zip_code_11650', 20.49416052939432),
 ('zip_code_86630', 20.35131438231251),
 ('zip_code_48052', 9.785400878146039),
 ('zip_code_70466', 9.753018661142576),
 ('zip_code_30723', 9.747578592989196),
 ('zip_code_22690', 9.734376695171463),
 ('grade', 2.6427442289162046),
 ('dti', 1.7199235354775095),
 ('loan_amnt', 0.7445231314973701)]
```

Insights:-

1. Metrics summary of my final Logistic Regression model:
   - Accuracy: 0.8
   - Precision: 0.5
   - Recall: 0.79
   - F1-score: 0.61
   - ROC-AUC: 0.9
   - PR-AUC: 0.77
   - fbeta(beta=2): 0.71
2. Confusion Matrix:

- TN: 64.57%
- FP: 15.64%
- FN: 4.09%
- TP: 15.67%

3. Zip-code (address), grade, dti and loan_amnt are the most important features by weights.

Conclusion

Summarizing Insights

Exploratory Data Analysis (EDA) Insights:

- **Dataset Overview:**
  - Total Rows: 396,030
  - Total Columns: 27
  - No duplicates; some missing values were noted.
- **Grade and Repayment:**
  - Grades A, B, C, and D have a higher number of fully-paid applicants compared to defaulters.
  - Grade B has the highest count of fully-paid loans.
- **Loan Term Insights:**
  - The 36-month period is the most common choice among applicants.
  - Loans with a 36-month term have a higher success rate compared to 60-month loans.
- **Applicant Characteristics:**
  - Most applicants report home ownership as 'Mortgage' or 'Rent.'
  - Individual applicants are more likely to apply for loans.
  - Applicants with no derogatory remarks or bankruptcies, and those with mortgage accounts, have a higher chance of receiving loans and successfully repaying them.
- **Loan Purpose:**
  - Debt consolidation and credit card repayment are the most common reasons for applying.
  - Applicants with longer employment histories have a higher likelihood of loan approval.
- **Occupation and Location Insights:**
  - Teachers and Managers are the occupations most likely to be granted loans.
  - Certain ZIP codes (e.g., 05113, 00813) are associated with high loan repayment rates, while others (e.g., 11650, 86630) have high default rates.
- **Feature Correlations:**
  - Strong correlations were found between `(loan_amnt, installment)` and `(pub_rec_bankruptcies, pub_rec)`.

Logistic Regression Model Insights:

- **Accuracy:** 0.8
- **Precision:** 0.5
- **Recall:** 0.79
- **F1-Score:** 0.61
- **ROC-AUC:** 0.9
- **PR-AUC:** 0.77
- **Confusion Matrix:** TN: 64.57%, FP: 15.64%, FN: 4.09%, TP: 15.67%
- **Key Features:** ZIP code, grade, debt-to-income ratio (DTI), and loan amount were identified as the most important features based on their weights.

Some Questions

1. Comment on Final Model Statistics (including confusion matrix):

The final logistic regression model demonstrates a balanced performance with a strong emphasis on recall. The key statistics are as follows:

- **Accuracy (0.8):** The model correctly predicts 80% of the cases, indicating a good overall performance.
- **Precision (0.5):** Precision is moderate, reflecting the trade-off between false positives and true positives. This indicates that while the model identifies a reasonable number of actual defaulters, it also misclassifies a notable proportion of non-defaulters as defaulters.
- **Recall (0.79):** The recall is high, which is crucial for minimizing false negatives—cases where a defaulter is incorrectly classified as a non-defaulter. This metric is particularly important in credit risk scenarios, where missing a defaulter can lead to significant financial losses.
- **F1-Score (0.61):** The F1-score balances precision and recall, providing a single metric to evaluate the model’s effectiveness. The score of 0.61 indicates a reasonable balance between the two.
- **ROC-AUC (0.9):** The high ROC-AUC value suggests that the model has a strong ability to distinguish between defaulters and non-defaulters.
- **PR-AUC (0.77):** The PR-AUC score further confirms the model's effectiveness in handling the imbalanced nature of the dataset.
- **Fbeta (0.71):** With a beta value of 2, the fbeta score prioritizes recall over precision, and a score of 0.71 reflects the model’s strength in correctly identifying defaulters.

The confusion matrix reveals a higher true negative rate (64.57%) compared to the true positive rate (15.67%), indicating the model is more conservative in predicting defaults. The false positive rate (15.64%) is slightly higher than desired, which suggests room for improvement in precision. However, the low false negative rate (4.09%) highlights the model's effectiveness in minimizing missed defaults, aligning with the business goal of reducing financial risk.

Overall, this model provides a robust framework for LoanTap to make informed lending decisions, balancing the need to identify high-risk applicants while minimizing potential losses.

2. ROC-AUC curve and its comment

Comment on ROC Curve

- The ROC curve illustrates the True Positive Rate (TPR) against the False Positive Rate (FPR) for various threshold values.
- The area under the ROC curve (AUC) is 0.9, which signifies that the logistic regression model has a strong ability to differentiate between the positive (defaulter) and negative (non-defaulter) classes.

3. PR curve and its comment

Comment on PR Curve

- The PR curve is particularly useful for evaluating model performance in cases of class imbalance.
- The area under the PR curve (PR AUC) is 0.77, indicating that the model effectively balances precision and recall.

Comment on PR Trade-Off Curve

- The PR trade-off curve illustrates the inverse relationship between precision and recall.
- As precision increases, recall typically decreases, and vice versa.
- The curve demonstrates that at various thresholds, the model can achieve either high precision with a trade-off in recall or high recall with a trade-off in precision.
- Understanding this trade-off is crucial for selecting the optimal threshold based on whether the priority is minimizing false positives (favoring precision) or minimizing false negatives (favoring recall).

4. How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.

Ensuring the Model Accurately Detects Real Defaulters and Minimizes False Positives

To improve the detection of real defaulters while reducing false positives, several strategies can be employed:

1. **Adjust the Decision Threshold:**

   - The default threshold for classifying defaulters might be 0.5, but this can be adjusted to balance recall and precision. By lowering the threshold, the model can become more sensitive to potential defaulters, reducing false negatives. Conversely, raising the threshold can help minimize false positives.
2. **Incorporate More Granular Features:**

   - Consider adding or engineering features that provide more detailed insights into an applicant's financial behavior, such as credit utilization ratios, recent spending patterns, or more specific employment stability indicators. These can help the model differentiate between high-risk and low-risk applicants more effectively.
3. **Implement a Cost-Sensitive Learning Approach:**

- Introduce a cost-sensitive learning framework where the cost of misclassifying a non-defaulter (false positive) is higher. This would encourage the model to be more cautious in predicting defaulters, reducing the number of false positives.

4. **Use Ensemble Methods:**

   - Combine multiple models, such as a random forest or gradient boosting, with logistic regression to leverage the strengths of each. Ensemble methods often provide better generalization and can reduce both false positives and false negatives.

5. **Apply More Rigorous Cross-Validation:**

   - Ensure that the model is tested across various subsets of the data through techniques like k-fold cross-validation. This helps in identifying any biases or inconsistencies in the model's predictions, leading to a more reliable and accurate model.

6. **Regularly Update the Model with New Data:**

   - Continuously update the model with the latest data to ensure it captures recent trends and changes in borrower behavior. A model trained on outdated data might not perform well on new applicants, leading to higher false positives.

7. **Monitor and Fine-Tune the Model Post-Deployment:**

   - After deployment, keep track of the model's performance metrics, especially the false positive rate. Fine-tune the model based on real-world data and outcomes to ensure it continues to meet the desired balance between detecting defaulters and minimizing false positives.

8. **Leverage Alternative Data Sources:**

   - Incorporate alternative data sources such as social media activity, digital footprints, or utility payment history. These can provide additional context that traditional credit metrics might miss, helping to refine the model's predictions.

9. **Develop a Two-Stage Model:**

   - Consider implementing a two-stage model where the first stage identifies potential defaulters with high recall, and the second stage refines these predictions with a focus on minimizing false positives. This approach can help balance the competing objectives of recall and precision.

10. **Involve Human Review for Borderline Cases:**

    - For cases where the model's prediction is uncertain or falls within a specific confidence range, involve human underwriters for final review. This can help catch errors that the model might make, particularly in borderline cases.

5. Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone. Comment on this.

Comment on Avoiding Risky Loan Disbursements Due to NPA Concerns

Given the critical issue of Non-Performing Assets (NPA) in the lending industry, it is crucial to adopt a conservative approach when disbursing loans. NPAs can significantly impact a financial institution's profitability and liquidity, as they represent loans that borrowers are unlikely to repay. This makes it imperative to minimize exposure to high-risk applicants.

Key Considerations:

1. **Prioritize Conservative Risk Management:**
    - Given the financial implications of NPAs, the lending strategy should prioritize minimizing risk over maximizing loan disbursements. The cost of approving a high-risk applicant can outweigh the potential earnings from interest if the loan turns into an NPA.
2. **Rigorous Applicant Screening:**
    - Implementing stringent criteria for applicant evaluation is essential. By thoroughly analyzing applicants' credit histories, income stability, and other financial behaviors, the model can help identify and filter out high-risk applicants more effectively.
3. **Focus on Long-Term Stability:**
    - Disbursing loans to applicants with a proven track record of financial responsibility, even if it means rejecting a larger number of applications, will help ensure long-term financial stability. This approach helps prevent short-term gains from turning into long-term losses due to NPAs.
4. **Iterative Model Improvement:**
    - Continuously refine the predictive model based on emerging trends in borrower behavior and economic conditions. This will allow for better anticipation of potential defaults and reduce the likelihood of disbursing loans that could become NPAs.
5. **Balance Caution with Opportunity:**
    - While it is crucial to avoid disbursing loans to high-risk individuals, it’s also important not to be overly conservative. A balanced approach that includes regular updates to the model and consideration of new data sources can help identify safe lending opportunities without excessive risk.
6. **Regular Portfolio Monitoring:**
    - Continuously monitor the loan portfolio to identify early signs of potential defaults. Early intervention strategies can be developed to mitigate losses, thereby preventing loans from becoming NPAs.

Conclusion: In light of the NPA problem, playing it safe by being selective in loan disbursements is a prudent strategy. This approach helps protect the financial institution from potential losses while ensuring that the loans granted are more likely to be repaid.

Recommendations

Based on the analysis and model insights, the following recommendations can be made to improve loan disbursement practices while minimizing risk:

1. **Focus on High-Grade Applicants:**

- Prioritize lending to applicants with Grade A, B, and C ratings, as they have shown a higher likelihood of fully repaying their loans. This will reduce the overall risk in the loan portfolio.

2. **Prefer Shorter Loan Tenures:**

   - Encourage applicants to choose 36-month loan terms, as these loans have demonstrated a higher success rate compared to 60-month loans. Shorter tenures reduce the likelihood of default, ensuring better cash flow and lower risk.

3. **Enhance Creditworthiness Checks:**

   - Give more weight to applicants with stable employment, no derogatory remarks, and no history of bankruptcies. These factors are strong indicators of an applicant's ability to repay the loan, reducing the chances of default.

4. **Target Debt Consolidation Loans:**

   - Focus on applicants seeking loans for debt consolidation, as this purpose often reflects a proactive approach to managing finances. Such loans are more likely to be repaid in full, contributing to a healthier loan portfolio.

5. **Consider Zip-Code Risk Factors:**

   - Implement stricter lending criteria for applicants from high-risk zip codes with a history of defaults. Conversely, offer more favorable terms to applicants from low-risk areas where the likelihood of loan repayment is higher.

6. **Regularly Update the Model:**

   - Continuously retrain and update the logistic regression model with new data to reflect changes in economic conditions and borrower behavior. Regular updates will ensure the model remains accurate and relevant.

7. **Monitor Key Features:**

   - Pay close attention to the most important features identified by the model, such as zip code, grade, debt-to-income ratio (DTI), and loan amount. These features should be regularly monitored to refine loan approval criteria.

8. **Consider Implementing a Two-Stage Model:**

   - Utilize a two-stage model where the first stage identifies potential defaulters with high recall, and the second stage focuses on reducing false positives. This approach balances the need for accurate default detection with the goal of minimizing missed opportunities for lending.

9. **Balance Loan Approvals with NPA Risks:**

   - While aiming to reduce NPAs, it is also important not to be overly conservative in loan disbursements. Striking the right balance will ensure that opportunities for profitable lending are not missed.

10. **Implement Human Review for High-Risk Cases:**

    - For applicants who are borderline in terms of risk, consider involving human underwriters to review the cases. This additional step can help prevent potential NPAs while still allowing for reasonable loan approvals.

Questionnaire:

1. What percentage of customers have fully paid their Loan Amount?
   - **`80%`**
2. Comment about the correlation between Loan Amount and Installment features.
   - **High Co-relation: `0.97`**
3. The majority of people have home ownership as ___.
   - **MORTGAGE**
4. People with grades 'A' are more likely to fully pay their loan. (T/F)
   - **True.** *Although grade B has highest applicants but grade A high loan payback rate.*
5. Name the top 2 afforded job titles.
   - **Teacher and Manager**
6. Thinking from a bank's perspective, which metric should our primary focus be on (ROC AUC, Precision, Recall, F1 Score)
   - In my opinion **Recall**. Because higher number of defaulters(False Negative) cause high financial loss.
   - But if Loan Tap is financially strong and wants to gain more interest than **F1-score** is also a good choice.
7. How does the gap in precision and recall affect the bank?
   - High Recall and Low Precision indicates lower number of Non-Performing Assest at the cost of gaining more interest.
   - Low Recall and High Precision indicates chance of gaining more interest at the cost of high number of Non-Performing Assest.
8. Which were the features that heavily affected the outcome?
   - **Address (Zip codes), Grade, DTI (debt-to-income ratio) and loan amount**
   - **Installment** is also a feature that heavily affected the outcome as it highly depends on Loan Amount.
9. Will the results be affected by geographical location? (Yes/No)
   - **Yes**
   - Zip-code: **`05113, 00813, 29597`** are the areas where almost every applicant has paid their loan successfully.
   - Zip-code **`11650, 86630, 93700`** are the areas where almost every applicant has defaulted.