```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('logistic_regression.csv')
```

```
df
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title
0	10000.0	36 months	11.44	329.48	B	B4	Marketing
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst
2	15600.0	36 months	10.49	506.97	B	B3	Statistician
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.
...	...	...	...	...	...	...	...
396025	10000.0	60 months	10.99	217.38	B	B4	licensed bankere
396026	21000.0	36 months	12.29	700.42	C	C1	Agent
396027	5000.0	36 months	9.99	161.32	B	B1	City Carrier
396028	21000.0	60 months	15.31	503.02	C	C2	Gracon Services, Inc
396029	2000.0	36 months	13.61	67.98	C	C2	Internal Revenue Service

396030 rows × 27 columns

```
df.iloc[:,9:18]
```

	annual_inc	verification_status	issue_d	loan_status	purpose
0	117000.0	Not Verified	Jan-2015	Fully Paid	vacation
1	65000.0	Not Verified	Jan-2015	Fully Paid	debt_consolidation
2	43057.0	Source Verified	Jan-2015	Fully Paid	credit_card
3	54000.0	Not Verified	Nov-2014	Fully Paid	credit_card
4	55000.0	Verified	Apr-2013	Charged Off	credit_card
...	...	...	...	...	...
396025	40000.0	Source Verified	Oct-2015	Fully Paid	debt_consolidation
396026	110000.0	Source Verified	Feb-2015	Fully Paid	debt_consolidation

```
df.iloc[:,18:]
```

	pub_rec	revol_bal	revol_util	total_acc	initial_list_status	applica
0	0.0	36369.0	41.8	25.0	w	
1	0.0	20131.0	53.3	27.0	f	
2	0.0	11987.0	92.2	26.0	f	
3	0.0	5472.0	21.5	13.0	f	
4	0.0	24584.0	69.8	43.0	f	
...	...	...	...	...	...	
396025	0.0	1990.0	34.3	23.0	w	
396026	0.0	43263.0	95.7	8.0	f	

```
df.shape
```

```
(396030, 27)
```

```
df.info()
```

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

```
df.describe()
```

	loan_amnt	int_rate	installment	annual_inc	dti	op
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	39603(
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	1
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	(
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	1(
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	9(

```
#For categoric variables, let us see their value counts
df['term'].value_counts()
```

```
term
 36 months    302005
 60 months     94025
Name: count, dtype: int64
```

Data Cleaning - Converting categoric variable to numeric variable

```
df['term'].replace({" 36 months":36, " 60 months":60}, inplace = True)
```

```
df['term']
```

```
0         36
1         36
2         36
3         36
4         60
          ..
396025    60
396026    36
396027    36
396028    60
396029    36
Name: term, Length: 396030, dtype: int64
```

```
df['grade'].value_counts()
```

```
grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
```

```
#We are going to convert subgrade from Ordinal values to Numeric values
grade = df['sub_grade'].value_counts().index.sort_values()
grade
```

```
Index(['A1', 'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5', 'C1', 'C2',
       'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D5', 'E1', 'E2', 'E3', 'E4',
       'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5'],
      dtype='object', name='sub_grade')
```

```
len(grade)
```

```
35
```

```
# We will assign more points to better grades and it decreases continuously to G5
pd.DataFrame({"grade_name": list(grade), "grade_value": list(range(len(grade),0,-1))})
```

	grade_name	grade_value
0	A1	35
1	A2	34
2	A3	33
3	A4	32
4	A5	31
5	B1	30
6	B2	29
7	B3	28
8	B4	27
9	B5	26
10	C1	25
11	C2	24
12	C3	23
13	C4	22
14	C5	21
15	D1	20
16	D2	19
17	D3	18
18	D4	17
19	D5	16
20	E1	15
21	E2	14
22	E3	13
23	E4	12
24	E5	11
25	F1	10
26	F2	9
27	F3	8
28	F4	7
29	F5	6
30	G1	5
31	G2	4
32	G3	3
33	G4	2
34	G5	1

```
df["sub_grade"].replace(list(grade), list(range(len(grade),0,-1)), inplace = True)
```

```
# Since we already have the subgrade data, we will no more require the Grade data
df.drop(columns = ["grade"], inplace= True)
```

```
df['emp_title'].value_counts()
```

```
emp_title
Teacher                                 4389
Manager                                 4250
Registered Nurse                        1856
RN                                      1846
Supervisor                              1830
                                        ...
Plus One Health Managment                  1
Comcast Corporate office                   1
Regional Counsel                           1
Social Work/Care Manager                   1
Director Bureau of Equipment Inventory     1
Name: count, Length: 173105, dtype: int64
```

```
# THis is the output variable, so we will convert it into 0 and 1.
df['loan_status'].replace(['Fully Paid', 'Charged Off'], [1,0], inplace = True)
```

```
emp_title_target_enc = df.groupby('emp_title')['loan_status'].agg(['mean', 'count']).reset_index()
emp_title_target_enc
```

	emp_title	mean	count
0	NSA Industries llc	1.0	1
1	Fibro Source	1.0	1
2	Long Ilsand College Hospital	0.0	1
3	mortgage banker	0.0	1
4	Credit rev specialist	0.0	1
...	...	...	...
173100	zozaya officiating	0.0	1
173101	zs backroom	1.0	1
173102	zueck transportation	1.0	1
173103	zulily	0.0	1
173104	License Compliance Investigator	1.0	1

173105 rows × 3 columns

Feature Engineering on employee title. Converting categoric to numeric value using Target Encoding.

```
# Target Encoding has a tendency to overfit so we will be smoothening
# Formula : (n*mean_of_category + alpha*global_mean)/(n + alpha)
# ---- where n is the count of rows in the category and alpha is a hyperparameter

global_mean = df['loan_status'].mean()
alpha = 10
emp_title_target_enc['target_enc'] = (emp_title_target_enc['count']*emp_title_target_enc['mean'] + alpha*global_mean)/(emp_t
```

```
emp_title_target_enc.sort_values( by = 'target_enc')
```

	emp_title	mean	count	target_enc
53991	G4S Secure Solutions	0.200000	10	0.501935
90660	Nurse assistant	0.000000	6	0.502419
51893	Floorhand	0.222222	9	0.528353
132550	Technition	0.142857	7	0.531689
158311	housekeeping	0.472222	36	0.544320
...	...	...	...	...
69179	Judge	1.000000	24	0.942315
10794	Associate Attorney	0.967742	62	0.944982
49643	Federal Bureau of Prisons	1.000000	26	0.945520
118392	Senior Systems Administrator	1.000000	28	0.948387
49992	Fidelity Investments	0.984615	65	0.960516

173105 rows × 4 columns

```
df = df.merge(emp_title_target_enc, left_on = 'emp_title', right_on = 'emp_title', how = 'left')
df
```

	loan_amnt	term	int_rate	installment	sub_grade	emp_title	emp_lengt
0	10000.0	36	11.44	329.48	27	Marketing	10+ yea
1	8000.0	36	11.99	265.68	26	Credit analyst	4 yea
2	15600.0	36	10.49	506.97	28	Statistician	< 1 ye
3	7200.0	36	6.49	220.65	34	Client Advocate	6 yea
4	24375.0	60	17.27	609.33	21	Destiny Management Inc.	9 yea
...	...	...	...	...	...	...	
396025	10000.0	60	10.99	217.38	27	licensed bankere	2 yea
396026	21000.0	36	12.29	700.42	25	Agent	5 yea
396027	5000.0	36	9.99	161.32	30	City Carrier	10+ yea
396028	21000.0	60	15.31	503.02	24	Gracon Services, Inc	10+ yea
396029	2000.0	36	13.61	67.98	24	Internal Revenue Service	10+ yea

396030 rows × 29 columns

```
df.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_type',
       'mort_acc', 'pub_rec_bankruptcies', 'address', 'mean', 'count',
       'target_enc'],
      dtype='object')
```

```
df.drop(columns = ['emp_title', 'mean', 'count'], inplace= True)
```

```
df.rename(columns = {'target_enc': 'emp_title'}, inplace = True)
```

```
df['emp_length']
```

```
0         10+ years
1           4 years
2          < 1 year
3           6 years
4           9 years
            ...
396025      2 years
396026      5 years
396027    10+ years
396028    10+ years
396029    10+ years
Name: emp_length, Length: 396030, dtype: object
```

```
df['emp_length'] = df['emp_length'].str.extract('(\d+)')
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:1: SyntaxWarning: invalid escape sequence '\d'
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_1735/3304813693.py:1: SyntaxWarning: invalid escape sequence
  df['emp_length'] = df['emp_length'].str.extract('(\d+)')
```

```
df
```

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	home_own
0	10000.0	36	11.44	329.48	27	10	
1	8000.0	36	11.99	265.68	26	4	MOR
2	15600.0	36	10.49	506.97	28	1	
3	7200.0	36	6.49	220.65	34	6	
4	24375.0	60	17.27	609.33	21	9	MOR
...	...	...	...	...	...	...	
396025	10000.0	60	10.99	217.38	27	2	
396026	21000.0	36	12.29	700.42	25	5	MOR
396027	5000.0	36	9.99	161.32	30	10	
396028	21000.0	60	15.31	503.02	24	10	MOR
396029	2000.0	36	13.61	67.98	24	10	

396030 rows × 26 columns

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

```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64
```

One Hot Encoding on 'Home Ownership' column

```
df['mortage_one_hot'] = 0
df.loc[df['home_ownership'] == 'MORTGAGE','mortage_one_hot'] = 1

df['rent_one_hot'] = 0
df.loc[df['home_ownership'] == 'RENT','rent_one_hot'] = 1

df['own_home'] = 0
df.loc[df['home_ownership'] == 'OWN','own_home'] = 1
df.drop(columns = ['home_ownership', 'mort_acc'], inplace = True)
```

```
df
```

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	annual_i
0	10000.0	36	11.44	329.48	27	10	11700C
1	8000.0	36	11.99	265.68	26	4	6500C
2	15600.0	36	10.49	506.97	28	1	43057
3	7200.0	36	6.49	220.65	34	6	5400C
4	24375.0	60	17.27	609.33	21	9	5500C
...	...	...	...	...	...	...	
396025	10000.0	60	10.99	217.38	27	2	4000C
396026	21000.0	36	12.29	700.42	25	5	11000C
396027	5000.0	36	9.99	161.32	30	10	5650C
396028	21000.0	60	15.31	503.02	24	10	6400C
396029	2000.0	36	13.61	67.98	24	10	42996

396030 rows × 27 columns

```
df['verification_status'].value_counts()
```

```
verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64
```

```
df['verification_status'].replace(['Verified', 'Source Verified', 'Not Verified'], [1,1,0], inplace = True)
```

```
df['verification_status'].value_counts()
```

```
verification_status
1    270948
0    125082
Name: count, dtype: int64
```

```
# Issue date is ordinal values, so we will convert it to numeric values
df['issue_d'].value_counts()
```

```
issue_d
Oct-2014    14846
Jul-2014    12609
Jan-2015    11705
Dec-2013    10618
Nov-2013    10496
            ...
Jul-2007       26
Sep-2008       25
Nov-2007       22
Sep-2007       15
Jun-2007        1
Name: count, Length: 115, dtype: int64
```

```
df['year'] = df['issue_d'].str.extract('(\d+)').astype('float64')
df['month'] = df['issue_d'].str.slice(0,3).replace(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'N
                                        , [1,2,3,4,5,6,7,8,9,10,11,12])/12
df['date']= df['year'] + df['month']
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:1: SyntaxWarning: invalid escape sequence '\d'
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_1735/3579227957.py:1: SyntaxWarning: invalid escape sequence
```

```
  df['year'] = df['issue_d'].str.extract('(\d+)').astype('float64')
```

```
df.drop(columns = ['issue_d', 'year', 'month'], inplace = True)
```

```
df.iloc[:,9:18]
```

	purpose	title	dti	earliest_cr_line	open_acc	pub_rec
0	vacation	Vacation	26.24	Jun-1990	16.0	0.0
1	debt_consolidation	Debt consolidation	22.05	Jul-2004	17.0	0.0
2	credit_card	Credit card refinancing	12.79	Aug-2007	13.0	0.0
3	credit_card	Credit card refinancing	2.60	Sep-2006	6.0	0.0
4	credit_card	Credit Card Refinance	33.95	Mar-1999	13.0	0.0
...	...	...	...	...	...	...
396025	debt_consolidation	Debt consolidation	15.63	Nov-2004	6.0	0.0
396026	debt_consolidation	Debt consolidation	21.45	Feb-2006	6.0	0.0

```
df.drop(columns = ['purpose'], inplace = True)
```

```
df.drop(columns = ['title'], inplace = True)
```

```
df['year'] = df['earliest_cr_line'].str.extract('(\d+)').astype('float64')
df['month'] = df['earliest_cr_line'].str.slice(0,3).replace(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
                                     , [1,2,3,4,5,6,7,8,9,10,11,12])/12
df['earliest_cr_line_date']= df['year'] + df['month']
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:1: SyntaxWarning: invalid escape sequence '\d'
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_1735/4040646995.py:1: SyntaxWarning: invalid escape sequence
  df['year'] = df['earliest_cr_line'].str.extract('(\d+)').astype('float64')
```

```
df.drop(columns = ['earliest_cr_line', 'year', 'month'], inplace = True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   loan_amnt              396030 non-null  float64
 1   term                   396030 non-null  int64  
 2   int_rate               396030 non-null  float64
 3   installment            396030 non-null  float64
 4   sub_grade              396030 non-null  int64  
 5   emp_length             377729 non-null  object 
 6   annual_inc             396030 non-null  float64
 7   verification_status    396030 non-null  int64  
 8   loan_status            396030 non-null  int64  
 9   dti                    396030 non-null  float64
 10  open_acc               396030 non-null  float64
 11  pub_rec                396030 non-null  float64
 12  revol_bal              396030 non-null  float64
 13  revol_util             395754 non-null  float64
 14  total_acc              396030 non-null  float64
 15  initial_list_status    396030 non-null  object 
 16  application_type       396030 non-null  object 
 17  pub_rec_bankruptcies   395495 non-null  float64
 18  address                396030 non-null  object 
 19  emp_title              373103 non-null  float64
 20  mortage_one_hot        396030 non-null  int64  
 21  rent_one_hot           396030 non-null  int64  
 22  own_home               396030 non-null  int64  
 23  date                   396030 non-null  float64
 24  earliest_cr_line_date  396030 non-null  float64
dtypes: float64(14), int64(7), object(4)
```

```
memory usage: 75.5+ MB
```

```
# We will convert pub rec and pub rec bankruptcy into just 0 and 1
df.loc[df['pub_rec'] >= 1, 'pub_rec'] = 1
df.loc[df['pub_rec'] != 1, 'pub_rec'] = 0
```

```
df.loc[df['pub_rec_bankruptcies'] >= 1, 'pub_rec_bankruptcies'] = 1
df.loc[df['pub_rec_bankruptcies'] != 1, 'pub_rec_bankruptcies'] = 0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   loan_amnt              396030 non-null  float64
 1   term                   396030 non-null  int64  
 2   int_rate               396030 non-null  float64
 3   installment            396030 non-null  float64
 4   sub_grade              396030 non-null  int64  
 5   emp_length             377729 non-null  object 
 6   annual_inc             396030 non-null  float64
 7   verification_status    396030 non-null  int64  
 8   loan_status            396030 non-null  int64  
 9   dti                    396030 non-null  float64
 10  open_acc               396030 non-null  float64
 11  pub_rec                396030 non-null  float64
 12  revol_bal              396030 non-null  float64
 13  revol_util             395754 non-null  float64
 14  total_acc              396030 non-null  float64
 15  initial_list_status    396030 non-null  object 
 16  application_type       396030 non-null  object 
 17  pub_rec_bankruptcies   396030 non-null  float64
 18  address                396030 non-null  object 
 19  emp_title              373103 non-null  float64
 20  mortage_one_hot        396030 non-null  int64  
 21  rent_one_hot           396030 non-null  int64  
 22  own_home               396030 non-null  int64  
 23  date                   396030 non-null  float64
 24  earliest_cr_line_date  396030 non-null  float64
dtypes: float64(14), int64(7), object(4)
memory usage: 75.5+ MB
```

```
# One hot encoding
df['initial_list_status'].replace(['w', 'f'], [0,1], inplace = True)
```

```
df['application_type'].value_counts()
```

```
application_type
INDIVIDUAL    395319
JOINT            425
DIRECT_PAY       286
Name: count, dtype: int64
```

```
df['application_type'].replace(['INDIVIDUAL', 'JOINT', 'DIRECT_PAY'], [1,0,1], inplace = True)
```

```
df
```

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	annual_i
0	10000.0	36	11.44	329.48	27	10	11700C
1	8000.0	36	11.99	265.68	26	4	6500C
2	15600.0	36	10.49	506.97	28	1	43057
3	7200.0	36	6.49	220.65	34	6	5400C
4	24375.0	60	17.27	609.33	21	9	5500C
...	...	...	...	...	...	...	
396025	10000.0	60	10.99	217.38	27	2	4000C
396026	21000.0	36	12.29	700.42	25	5	11000C
396027	5000.0	36	9.99	161.32	30	10	5650C
396028	21000.0	60	15.31	503.02	24	10	6400C
396029	2000.0	36	13.61	67.98	24	10	4299E

396030 rows × 25 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  int64  
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   sub_grade             396030 non-null  int64  
 5   emp_length            377729 non-null  object 
 6   annual_inc            396030 non-null  float64
 7   verification_status   396030 non-null  int64  
 8   loan_status           396030 non-null  int64  
 9   dti                   396030 non-null  float64
 10  open_acc              396030 non-null  float64
 11  pub_rec               396030 non-null  float64
 12  revol_bal             396030 non-null  float64
 13  revol_util            395754 non-null  float64
 14  total_acc             396030 non-null  float64
 15  initial_list_status   396030 non-null  int64  
 16  application_type      396030 non-null  int64  
 17  pub_rec_bankruptcies  396030 non-null  float64
 18  address               396030 non-null  object 
 19  emp_title             373103 non-null  float64
 20  mortage_one_hot       396030 non-null  int64  
 21  rent_one_hot          396030 non-null  int64  
 22  own_home              396030 non-null  int64  
 23  date                  396030 non-null  float64
 24  earliest_cr_line_date 396030 non-null  float64
dtypes: float64(14), int64(9), object(2)
memory usage: 75.5+ MB
```

Treating missing values

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

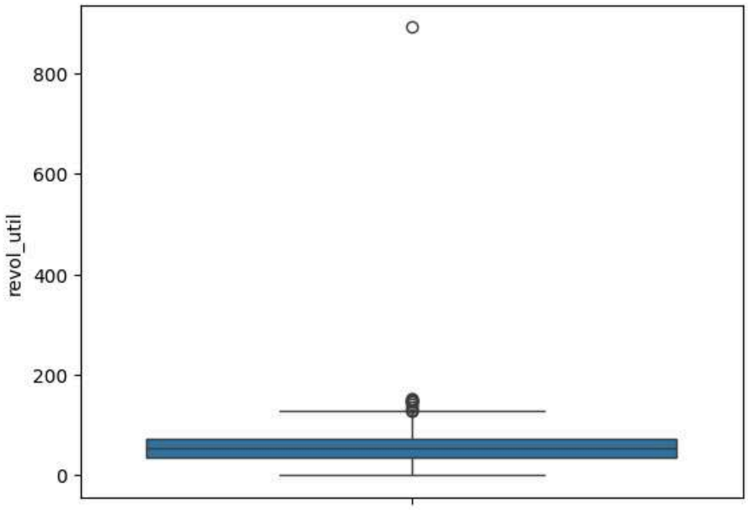
```
loan_amnt                0
term                     0
int_rate                 0
installment              0
```

```
sub_grade                   0
emp_length              18301
annual_inc                  0
verification_status         0
loan_status                 0
dti                         0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
pub_rec_bankruptcies        0
address                     0
emp_title               22927
mortage_one_hot             0
rent_one_hot                0
own_home                    0
date                        0
earliest_cr_line_date       0
dtype: int64
```

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

```
sns.boxplot(y = df['revol_util'])
```

```
<Axes: ylabel='revol_util'>
```




```
df['revol_util'].fillna(df['revol_util'].median(),inplace = True)
```

Data preparation and preprocessing

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.drop(columns = ['address']))
```

```
scaled_data = pd.DataFrame(scaled_data, columns = df.drop(columns = ['address']).columns)
```

```
scaled_data
```

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	annua
0	-0.492243	-0.557975	-0.491799	-0.408291	0.467127	1.130888	0.(
1	-0.731551	-0.557975	-0.368816	-0.662750	0.315634	-0.575068	-0.'
2	0.177819	-0.557975	-0.704225	0.299609	0.618620	-1.428046	-0.!
3	-0.827274	-0.557975	-1.598649	-0.842348	1.527580	-0.006416	-0.:
4	1.227783	1.792196	0.811824	0.707861	-0.441833	0.846562	-0.:
...	...	...	...	...	...	...	...
396025	-0.492243	1.792196	-0.592422	-0.855390	0.467127	-1.143720	-0.!
396026	0.823951	-0.557975	-0.301734	1.071164	0.164140	-0.290742	0.!
396027	-1.090513	-0.557975	-0.816028	-1.078979	0.921607	1.130888	-0.:
396028	0.823951	1.792196	0.373556	0.283855	0.012647	1.130888	-0.'
396029	-1.449475	-0.557975	-0.006574	-1.451256	0.012647	1.130888	-0.!

396030 rows × 24 columns

Filling missing values using KNN Imputer

The reason we have done normalisation before imputing is because KNN is a distance based algorithm. So we want all features to be equally important. We don't want a single feature to dominate.

```
from sklearn.impute import KNNImputer
```

```
imputer = KNNImputer(n_neighbors = 12)
df_imputed = imputer.fit_transform(scaled_data)
df_imputed = pd.DataFrame(df_imputed, columns = scaled_data.columns)
df_imputed
```

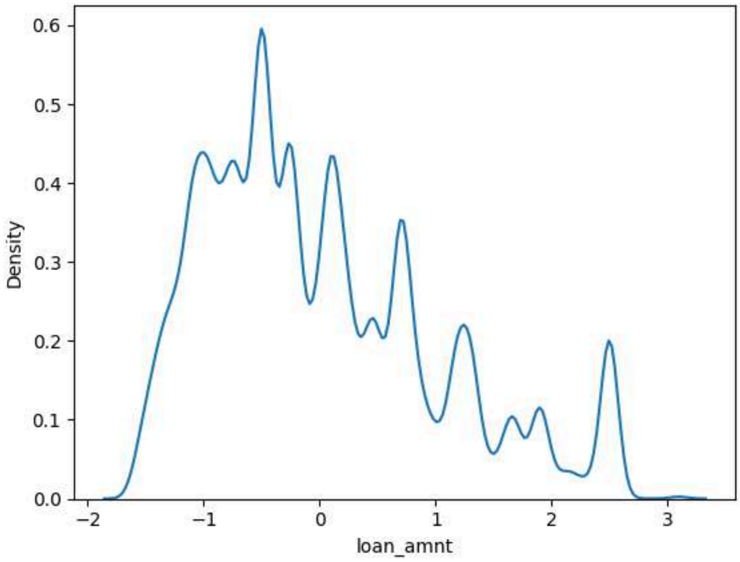
	loan_amnt	term	int_rate	installment	sub_grade	emp_length	annua
0	-0.492243	-0.557975	-0.491799	-0.408291	0.467127	1.130888	0.(
1	-0.731551	-0.557975	-0.368816	-0.662750	0.315634	-0.575068	-0.'
2	0.177819	-0.557975	-0.704225	0.299609	0.618620	-1.428046	-0.!
3	-0.827274	-0.557975	-1.598649	-0.842348	1.527580	-0.006416	-0.:
4	1.227783	1.792196	0.811824	0.707861	-0.441833	0.846562	-0.:
...	...	...	...	...	...	...	...
396025	-0.492243	1.792196	-0.592422	-0.855390	0.467127	-1.143720	-0.!
396026	0.823951	-0.557975	-0.301734	1.071164	0.164140	-0.290742	0.!
396027	-1.090513	-0.557975	-0.816028	-1.078979	0.921607	1.130888	-0.:
396028	0.823951	1.792196	0.373556	0.283855	0.012647	1.130888	-0.'
396029	-1.449475	-0.557975	-0.006574	-1.451256	0.012647	1.130888	-0.!

396030 rows × 24 columns

Univariate Analysis

```
sns.kdeplot(x = df_imputed['loan_amnt'])
```

```
<Axes: xlabel='loan_amnt', ylabel='Density'>
```

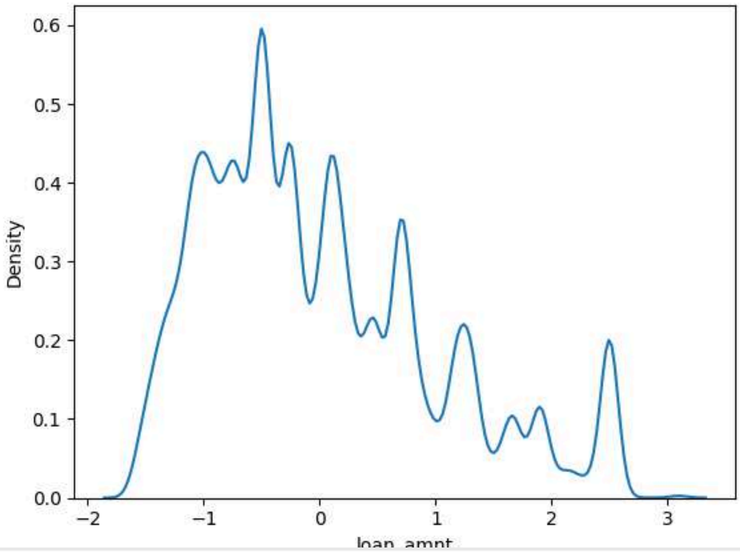


Treating outliers by clipping

```
df['loan_amnt'][df['loan_amnt'] >=3] = 3
sns.kdeplot(x = df_imputed['loan_amnt'])
```

```
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_1735/970610851.py:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df['loan_amnt'][df['loan_amnt'] >=3] = 3
<Axes: xlabel='loan_amnt', ylabel='Density'>
```




```
df['loan_amnt'].max()
```

```
3.0
```

```
for columns in df_imputed.columns:
    sns.kdeplot(x = df_imputed[columns])
    plt.show()
```

Visualising outliers using Box plots

```
for columns in df_imputed.columns:
    sns.boxplot(y = df_imputed[columns])
    plt.show()
```

Show hidden output

```
df_imputed.columns
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'sub_grade',
       'emp_length', 'annual_inc', 'verification_status', 'loan_status', 'dti',
       'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', 'application_type', 'pub_rec_bankruptcies',
       'emp_title', 'mortage_one_hot', 'rent_one_hot', 'own_home', 'date',
       'earliest_cr_line_date'],
      dtype='object')
```

Feature Engineering - Transforming the data

For variables which have skewed distribution, we will use box-cox transformation

```
from scipy.stats import boxcox
scaler = StandardScaler()
def trans(df):
    df = df - np.min(df) +1
    for col in df.columns:
        df[col], lambda_ = boxcox(df[col])
        df_ = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)

    return df_
```

For variables which are normally distributed, we will perform clipping to remove outliers

```
def clip(df):
    for col in df.columns:
        df.loc[df[col]>=3,col] = 3
        df.loc[df[col]<=-3,col] = -3
    return df
```

```
df_boc_cox = trans(df_imputed[['annual_inc','dti', 'open_acc','revol_bal', 'revol_util', 'total_acc']])
df_clip = clip(df_imputed[['loan_amnt', 'int_rate', 'installment', 'sub_grade','emp_title', 'date', 'earliest_cr_line_date']
df_lef_out = df[['term', 'verification_status', 'loan_status', 'pub_rec',
                'initial_list_status','pub_rec_bankruptcies',
                'mortage_one_hot', 'rent_one_hot', 'own_home', 'application_type']]

df_new = pd.concat([df_boc_cox, df_clip, df_lef_out], axis = 1)
for columns in df_new.columns:
    sns.kdeplot(x = df_new[columns])
    plt.show()
```

Show hidden output

```
df_new[['annual_inc','dti', 'open_acc','revol_bal', 'revol_util', 'total_acc']] = clip(df_new[['annual_inc','dti', 'open_acc'
for columns in df_new.columns:
    sns.kdeplot(x = df_new[columns])
    plt.show()
```

Show hidden output

Treating Outliers in multi-dimensional plane using Gaussian Mixture Models

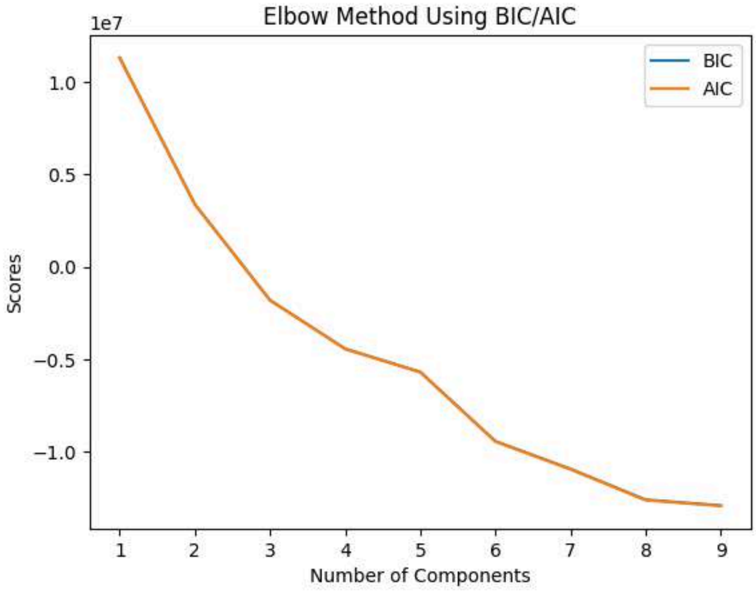
```
from sklearn.mixture import GaussianMixture
```

Before we perform GMM, we need to understand how many classes exist in the dataset. To find the optimum number we find out the AIC and BIC score and find the elbow point.

```
bics = []
aics = []
for n in range(1,10):
    gmm = GaussianMixture(n_components = n)
    gmm.fit(df_new.drop(columns = 'loan_status'))
    bics.append(gmm.bic(df_new.drop(columns = 'loan_status')))
    aics.append(gmm.aic(df_new.drop(columns = 'loan_status')))

plt.plot( np.arange(1,10), bics, label='BIC')
plt.plot( np.arange(1,10), aics, label='AIC')
```

```
plt.xlabel('Number of Components')
plt.ylabel('Scores')
plt.legend()
plt.title('Elbow Method Using BIC/AIC')
plt.show()
```



As we can see we don't find any significant decline in AIC/BIC. But our intuition tells us that it should be 2 classes - the ones who fully paid the loan and ones who didn't.

```
gmm = GaussianMixture(n_components = 2)
gmm.fit(df_new.drop(columns = 'loan_status'))
prob = gmm.score_samples(df_new.drop(columns = 'loan_status'))
```

```
threshold = np.percentile(prob, 1)
```

Let us remove the data points whose probability of lying inside the cluster is less than 1%

```
df_new = df_new[prob >= threshold]
```

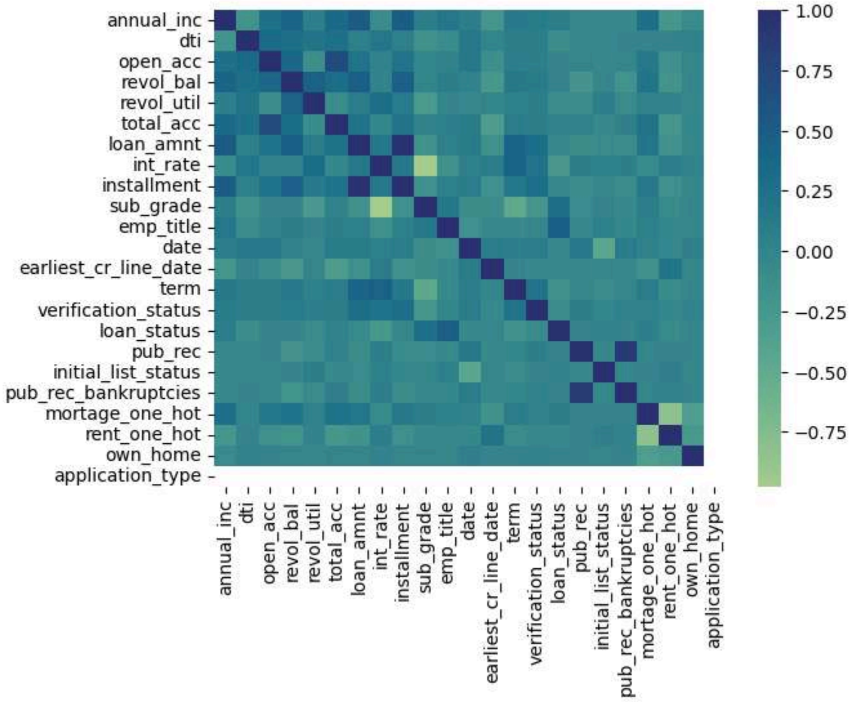
Bivariate Analysis

```
sns.pairplot(df_new, size = 4)
```

Show hidden output

```
sns.heatmap(df_new.corr(), cmap ="crest")
```

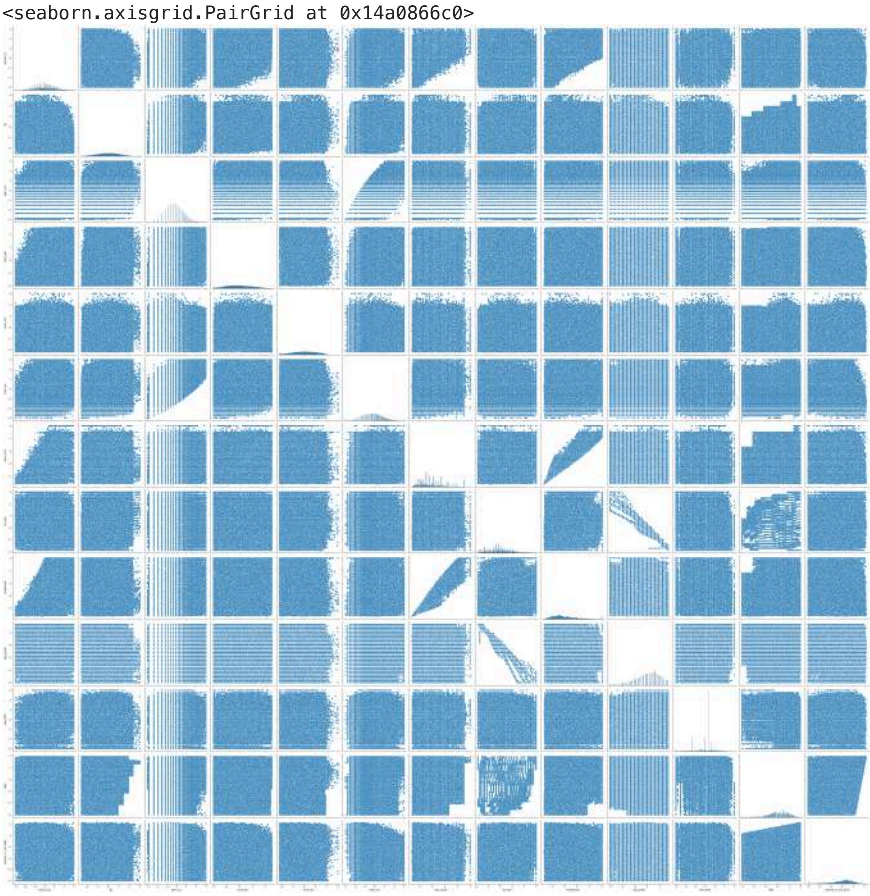
```
<Axes: >
```




```
# The above pairplot contains too many columns, so let us only focus on what is relavant.
```

```
sns.pairplot(df_new[['annual_inc','dti', 'open_acc','revol_bal', 'revol_util', 'total_acc','loan_amnt', 'int_rate', 'install
```

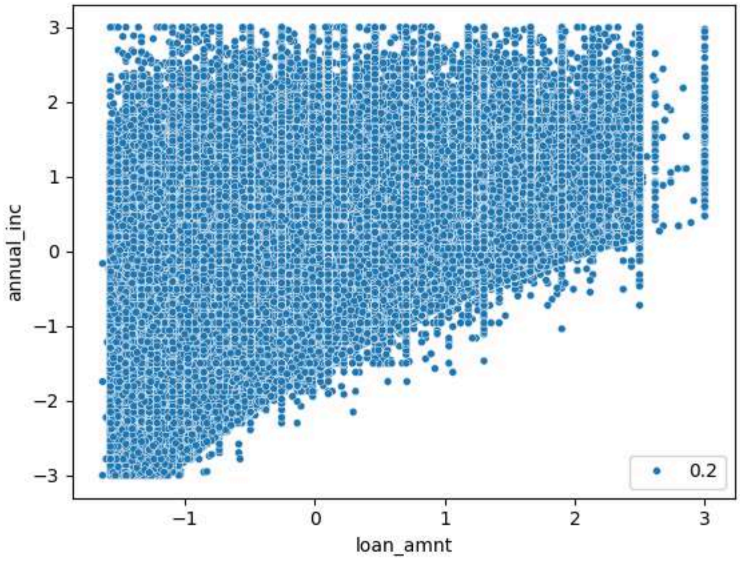
```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.PairGrid at 0x14a0866c0>
```



```
# This still looks unreadable, so let us play around with point sizes.
```

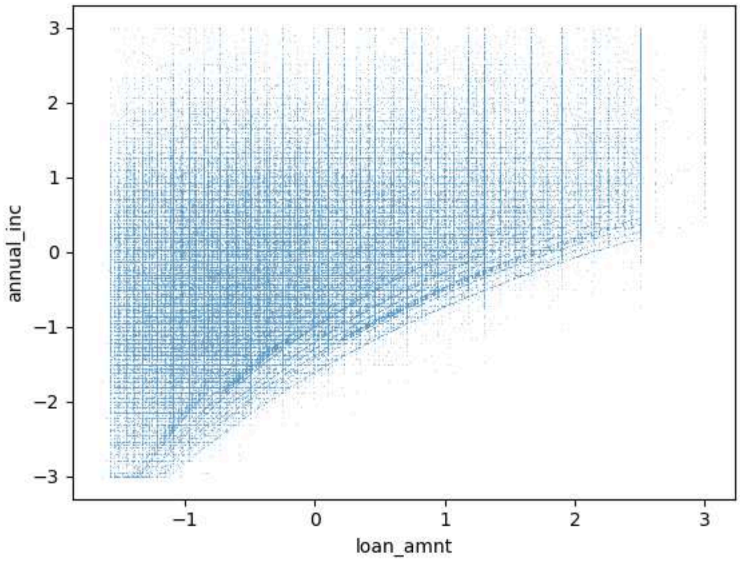
```
sns.scatterplot(x = df_new['loan_amnt'], y = df_new['annual_inc'], size = 0.2)
```

<Axes: xlabel='loan_amnt', ylabel='annual_inc'>




```
sns.scatterplot(x = df_new['loan_amnt'], y = df_new['annual_inc'], s = 0.2)
```

<Axes: xlabel='loan_amnt', ylabel='annual_inc'>




```
df2 = df_new[['annual_inc','dti', 'open_acc','revol_bal', 'revol_util', 'total_acc','loan_amnt', 'int_rate', 'installment',
n = len(df2.columns)
for i in range(n):
    if i == (n-1):
        break
    for j in range(i+1,n):
        sns.scatterplot(x = df2.iloc[:,i], y = df2.iloc[:,j], s = 0.4)
        plt.show()
```

Show hidden output

```
n = len(df2.columns)
df2.iloc[:,3]
```

```
0         1.546721
1         0.771260
2         0.047435
3        -0.855094
4         1.049019
            ...
396025   -1.526823
396026    1.736422
396027    1.420199
396028    0.420028
396029   -1.064561
Name: revol_bal, Length: 392069, dtype: float64
```

```
df_new.drop(columns = ['installment', 'sub_grade', 'pub_rec_bankruptcies'],inplace = True)
```

Let us drop duplicate values

```
df_new.drop_duplicates(inplace = True)
```

```
df_new
```

	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	loan_a
0	1.221826	1.081457	1.000450	1.546721	-0.498104	0.118440	-0.492
1	0.040022	0.628910	1.146471	0.771260	-0.028143	0.287092	-0.731
2	-0.812425	-0.505303	0.506486	0.047435	1.584173	0.203864	0.177
3	-0.348535	-2.043472	-1.179456	-0.855094	-1.317655	-1.133916	-0.827
4	-0.310274	1.836211	0.506486	1.049019	0.651903	1.390271	1.227
...	...	...	...	...	...	...	
396025	-0.958631	-0.135229	-1.179456	-1.526823	-0.802534	-0.059481	-0.492
396026	1.106544	0.561320	-1.179456	1.736422	1.730664	-1.842867	0.823
396027	-0.254052	0.104430	0.845793	1.420199	0.531935	-0.059481	-1.090
396028	0.007453	-0.103666	-0.332918	0.420028	-0.007631	-0.346147	0.823
396029	-0.815265	-1.135266	-2.361137	-1.064561	1.546538	-0.447630	-1.449

392068 rows × 20 columns

Checking for Multicollinearity

```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Add a constant to the predictors
df_new = sm.add_constant(df_new)

# Calculate VIF scores
vif_data = pd.DataFrame()
vif_data['Feature'] = df_new.columns
vif_data['VIF'] = [variance_inflation_factor(df_new.values, i) for i in range(df_new.shape[1])]

# Display the VIF scores
print(vif_data)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/statsmodels/stats/outliers_influence.py:
  vif = 1. / (1. - r_squared_i)
                Feature       VIF
0            annual_inc  1.916954
1                   dti  1.528316
2              open_acc  2.401096
3             revol_bal  2.343151
4            revol_util  1.753073
5             total_acc  2.299320
6             loan_amnt  1.909349
7              int_rate  1.627638
8             emp_title  1.333348
9                  date  1.386505
10  earliest_cr_line_date  1.244496
11                  term  1.499687
12   verification_status  1.154925
13           loan_status  1.352833
14               pub_rec  1.113557
15   initial_list_status  1.273909
16       mortage_one_hot       inf
17         rent_one_hot       inf
18             own_home       inf
19      application_type  0.000000
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/statsmodels/regression/linear_model.py:1
  return 1 - self.ssr/self.centered_tss
```

```
# Looks like rent_one_hot column is easily predictable by other vatiables, so lets drop it and again run the vif
df_new2 = df_new.drop(columns = [ 'rent_one_hot'])

# Calculate VIF scores
vif_data = pd.DataFrame()
vif_data['Feature'] = df_new2.columns
vif_data['VIF'] = [variance_inflation_factor(df_new2.values, i) for i in range(df_new2.shape[1])]

# Display the VIF scores
print(vif_data)
```

```
                  Feature        VIF
0              annual_inc   1.916954
1                     dti   1.528316
2                open_acc   2.401096
3               revol_bal   2.343151
4              revol_util   1.753073
5               total_acc   2.299320
6               loan_amnt   1.909349
7                int_rate   1.627638
8               emp_title   1.333348
9                    date   1.386505
10   earliest_cr_line_date   1.244496
11                   term   1.499687
12    verification_status   1.154925
13            loan_status   1.352833
14                pub_rec   1.113557
15    initial_list_status   1.273909
16        mortage_one_hot   1.279780
17               own_home   1.141232
18       application_type  39.618399
```

```
df_new2 = df_new.drop(columns = [ 'rent_one_hot', 'application_type'])

# Calculate VIF scores
vif_data = pd.DataFrame()
vif_data['Feature'] = df_new2.columns
vif_data['VIF'] = [variance_inflation_factor(df_new2.values, i) for i in range(df_new2.shape[1])]

# Display the VIF scores
print(vif_data)
```

```
                  Feature       VIF
0              annual_inc  1.916938
1                     dti  1.528184
2                open_acc  2.388082
3               revol_bal  2.339744
4              revol_util  1.731083
5               total_acc  2.285436
6               loan_amnt  1.797020
7                int_rate  1.451097
8               emp_title  1.296468
9                    date  1.377008
10   earliest_cr_line_date  1.242594
11                   term  8.373868
12    verification_status  3.480185
13            loan_status  5.635092
14                pub_rec  1.293214
15    initial_list_status  2.868932
16        mortage_one_hot  2.531061
17               own_home  1.247506
```

```
df_new2 = df_new.drop(columns = [ 'rent_one_hot', 'application_type','term'])

# Calculate VIF scores
vif_data = pd.DataFrame()
vif_data['Feature'] = df_new2.columns
vif_data['VIF'] = [variance_inflation_factor(df_new2.values, i) for i in range(df_new2.shape[1])]

# Display the VIF scores
print(vif_data)
```

```
                  Feature       VIF
0              annual_inc  1.913093
1                     dti  1.527863
2                open_acc  2.387958
3               revol_bal  2.339650
4              revol_util  1.727972
5               total_acc  2.284524
6               loan_amnt  1.776380
7                int_rate  1.364235
8               emp_title  1.226035
9                    date  1.371504
10   earliest_cr_line_date  1.239720
11    verification_status  2.901844
12            loan_status  3.905802
```

```
13               pub_rec  1.290547
14   initial_list_status  2.673744
15       mortage_one_hot  2.283420
16              own_home  1.213701
```

```
# We see that these 3 columns have a high vif, so lets drop them and
df_new.drop(columns = [ 'rent_one_hot', 'application_type','term'], inplace = True)
```

```
df_new
```

	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	loan_a
0	1.221826	1.081457	1.000450	1.546721	-0.498104	0.118440	-0.49
1	0.040022	0.628910	1.146471	0.771260	-0.028143	0.287092	-0.73
2	-0.812425	-0.505303	0.506486	0.047435	1.584173	0.203864	0.17
3	-0.348535	-2.043472	-1.179456	-0.855094	-1.317655	-1.133916	-0.82
4	-0.310274	1.836211	0.506486	1.049019	0.651903	1.390271	1.22
...	...	...	...	...	...	...	
396025	-0.958631	-0.135229	-1.179456	-1.526823	-0.802534	-0.059481	-0.49
396026	1.106544	0.561320	-1.179456	1.736422	1.730664	-1.842867	0.82
396027	-0.254052	0.104430	0.845793	1.420199	0.531935	-0.059481	-1.09
396028	0.007453	-0.103666	-0.332918	0.420028	-0.007631	-0.346147	0.82
396029	-0.815265	-1.135266	-2.361137	-1.064561	1.546538	-0.447630	-1.44

392068 rows × 17 columns

Building the base model

Let us evaluate the model on recall, precision, accuracy, f1 score and specifity using K Folf cross validation.

```
from sklearn.model_selection import KFold, cross_validate
from sklearn.metrics import f1_score, recall_score, precision_score, accuracy_score, make_scorer
```

```
def specificity(y_true, y_pred):
    tn = np.sum(y_pred[y_true == 0]==0)
    fp = np.sum(y_pred[y_true == 0]==1)
    return tn/(tn+fp)
```

```
model = LogisticRegression()
kfold = KFold(n_splits = 5)
x_train, x_test, y_train, y_test = train_test_split(df_new.drop(columns = 'loan_status'), df_new['loan_status'], test_size =
specificity = make_scorer(specificity)
scoring = {
    'recall': 'recall',
    'precision': 'precision',
    'accuracy': 'accuracy',
    'f1_score': 'f1',
    'specificity': specificity
}

results = cross_validate(estimator = model, X = x_train, y = y_train, scoring = scoring, cv = kfold)
```

```
results
```

```
{'fit_time': array([0.41666675, 0.41862988, 0.44450092, 0.3990798 , 0.39305496]),
 'score_time': array([0.03732204, 0.03639603, 0.03877211, 0.03370905, 0.03579593]),
 'test_recall': array([0.94888496, 0.94754995, 0.9465958 , 0.94755723, 0.94683359]),
 'test_precision': array([0.86610477, 0.86652769, 0.8686138 , 0.86701607, 0.86669326]),
 'test_accuracy': array([0.84121088, 0.84044571, 0.84138624, 0.84092395, 0.84001275]),
 'test_f1_score': array([0.9056071 , 0.90522947, 0.90592974, 0.90549921, 0.90499271]),
 'test_specificity': array([0.40302247, 0.40055361, 0.40191467, 0.40265559, 0.39968972])}
```

```
print('recall', np.mean(results['test_recall']))
print('precision', np.mean(results['test_precision']))
print('accuracy', np.mean(results['test_accuracy']))
print('f1 score', np.mean(results['test_f1_score']))
print('specificity', np.mean(results['test_specificity']))
```

```
recall 0.947484306818626
precision 0.8669911170052262
accuracy 0.8407959063708631
f1 score 0.9054516442534428
specificity 0.40156721011208896
```

We see that model is performing really well on recall, precision, f1 score. But very poorly on specificity. This shows that the model is not able to classifiy negative class (here it is 0).

```
# Let us save the final dataset
df_new.to_csv('df_new_loan_tab', index = False)
```

```
df_new = pd.read_csv('df_new_loan_tap')
```

```
df_new
```

	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	loan_a
0	1.221826	1.081457	1.000450	1.546721	-0.498104	0.118440	-0.49
1	0.040022	0.628910	1.146471	0.771260	-0.028143	0.287092	-0.73
2	-0.812425	-0.505303	0.506486	0.047435	1.584173	0.203864	0.17
3	-0.348535	-2.043472	-1.179456	-0.855094	-1.317655	-1.133916	-0.82
4	-0.310274	1.836211	0.506486	1.049019	0.651903	1.390271	1.22
...	...	...	...	...	...	...	
392063	-0.958631	-0.135229	-1.179456	-1.526823	-0.802534	-0.059481	-0.49
392064	1.106544	0.561320	-1.179456	1.736422	1.730664	-1.842867	0.82
392065	-0.254052	0.104430	0.845793	1.420199	0.531935	-0.059481	-1.09
392066	0.007453	-0.103666	-0.332918	0.420028	-0.007631	-0.346147	0.82
392067	-0.815265	-1.135266	-2.361137	-1.064561	1.546538	-0.447630	-1.44

392068 rows × 17 columns

```
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
model.coef_
```

```
array([[ 0.0947243 , -0.17269881, -0.18112777,  0.09063865, -0.14223899,
         0.08807614, -0.20165356, -0.48793512,  1.23014622,  0.10170187,
         0.05862076, -0.23038896,  0.04725662, -0.00738709,  0.29912027,
         0.15965224]])
```

```
df_new.columns
```

```
Index(['annual_inc', 'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc',
       'loan_amnt', 'int_rate', 'emp_title', 'date', 'earliest_cr_line_date',
       'verification_status', 'loan_status', 'pub_rec', 'initial_list_status',
       'mortage_one_hot', 'own_home'],
      dtype='object')
```

Printing the coefficients of the variables

```
pd.DataFrame({'columns': df_new.drop(columns = 'loan_status').columns, 'coefficients':model.coef_[0]})
```

	columns	coefficients
0	annual_inc	0.094724
1	dti	-0.172699
2	open_acc	-0.181128
3	revol_bal	0.090639
4	revol_util	-0.142239
5	total_acc	0.088076
6	loan_amnt	-0.201654
7	int_rate	-0.487935
8	emp_title	1.230146
9	date	0.101702
10	earliest_cr_line_date	0.058621
11	verification_status	-0.230389
12	pub_rec	0.047257
13	initial_list_status	-0.007387
14	mortage_one_hot	0.299120
15	own_home	0.159652

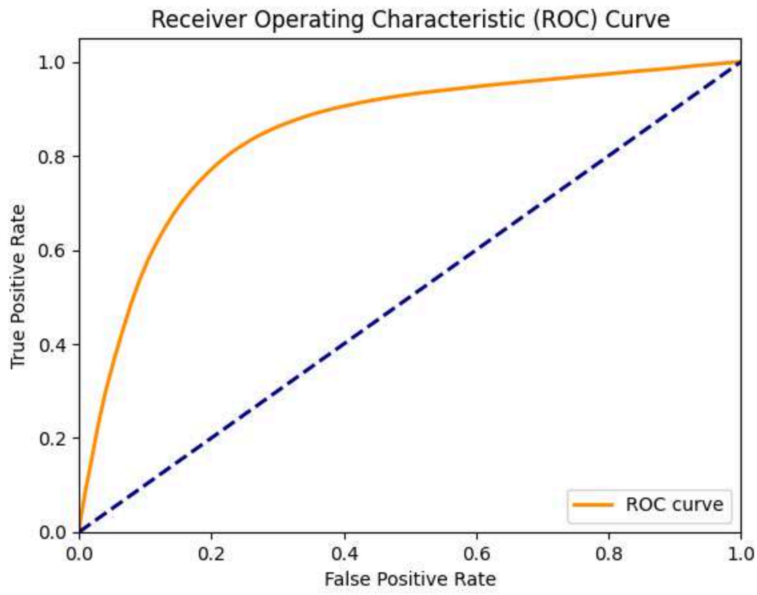
Let us find the optimum threshold for classification.

```
y_pred = model.predict_proba(x_train)[:,1]
```

```
threshold = np.linspace(0,1,101)
```

```
tpr = []
fpr = []
for t in threshold:
    y_tr = np.array(y_train)
    y_pr = np.where(y_pred>=t,1,0)
    tp = 0
    fp = 0
    tn = 0
    fn = 0
    for i in range(len(y_pr)):
        if ((y_pr[i] == 1) and (y_tr[i] == 1)):
            tp = tp +1
        elif ((y_pr[i] == 0) and (y_tr[i] == 1)):
            fn = fn +1
        elif ((y_pr[i] == 0) and (y_tr[i] == 0)):
            tn = tn +1
        else:
            fp = fp +1
    fpr.append(fp/(fp+tn))
    tpr.append(tp/(tp+fn))
```

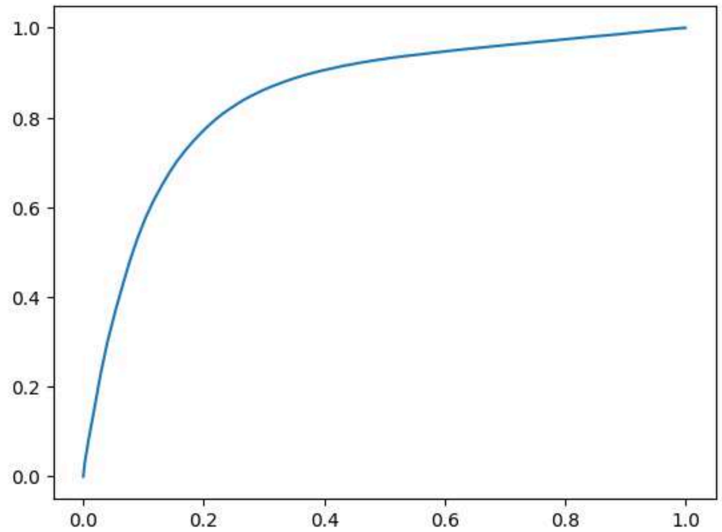
```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve' )
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



We see that the model is performing better than a randomly guessing model. The optimum threshold would be something near the top left corner which maximises TPR but minimises FPR.

```
plt.plot(fpr, tpr)
```

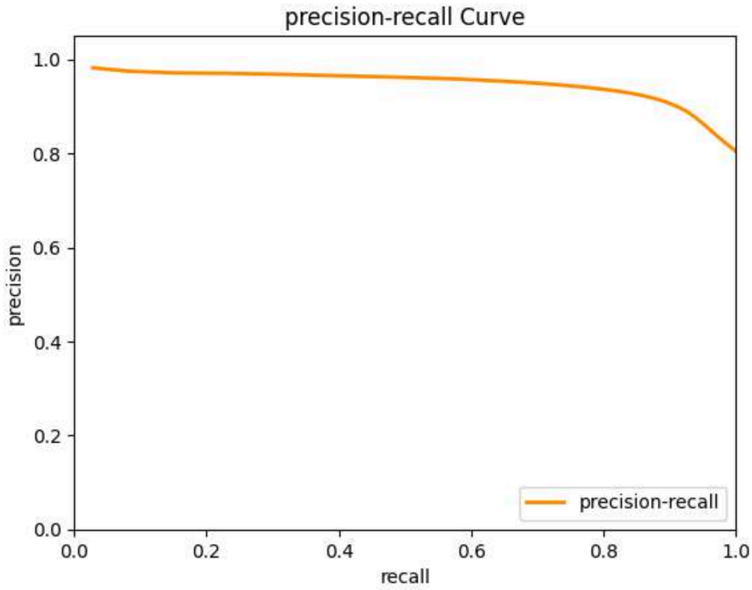
```
[<matplotlib.lines.Line2D at 0x126eecc20>]
```



Let us make precision recall curve.

```
recall = []
precision = []
for t in threshold[0:-1]:
    y_tr = np.array(y_train)
    y_pr = np.where(y_pred>=t,1,0)
    tp = 0
    fp = 0
    tn = 0
    fn = 0
    for i in range(len(y_pr)):
        if ((y_pr[i] == 1) and (y_tr[i] == 1)):
            tp = tp +1
        elif ((y_pr[i] == 0) and (y_tr[i] == 1)):
            fn = fn +1
        elif ((y_pr[i] == 0) and (y_tr[i] == 0)):
            tn = tn +1
        else:
            fp = fp +1
    precision.append(tp/(tp+fp))
    recall.append(tp/(tp+fn))
```

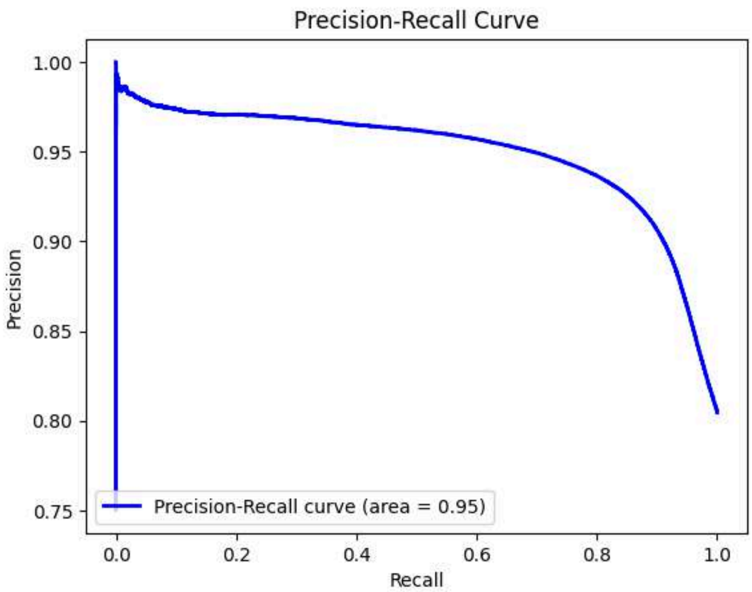
```
plt.figure()
plt.plot(recall, precision, color='darkorange', lw=2, label='precision-recall' )
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('recall')
plt.ylabel('precision')
plt.title('precision-recall Curve')
plt.legend(loc="lower right")
plt.show()
```



Let us make precision recall curve using sklearn

```
from sklearn.metrics import precision_recall_curve, average_precision_score
precision, recall, _ = precision_recall_curve(y_train, y_pred)
average_precision = average_precision_score(y_train, y_pred)

plt.figure()
plt.plot(recall, precision, color='b', lw=2, label='Precision-Recall curve (area = %0.2f)' % average_precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```



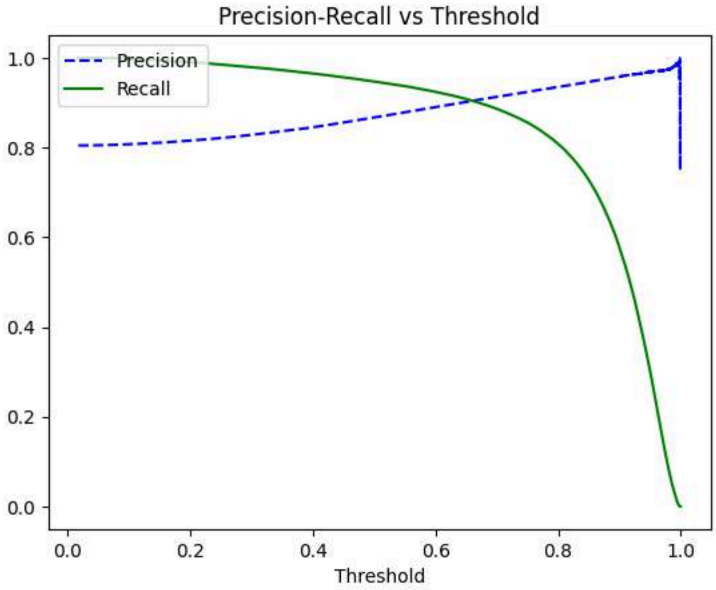
Let us find the optimum threshold for maximising the f1 score.

```
precision, recall, thresholds = precision_recall_curve(y_train, y_pred)
f1_scores = 2*recall*precision / (recall + precision)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]

print('Best Threshold by F1 Score:', optimal_threshold)

# Plotting Precision-Recall Curve
plt.plot(thresholds, precision[:-1], 'b--', label='Precision')
plt.plot(thresholds, recall[:-1], 'g-', label='Recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.title('Precision-Recall vs Threshold')
plt.show()
```

```
Best Threshold by F1 Score: 0.578998824546087
```




```
y_pred = model.predict_proba(x_test)[:,1]
y_tr = np.array(y_test)
y_pr = np.where(y_pred>=optimal_threshold,1,0)
tp = 0
fp = 0
tn = 0
fn = 0
for i in range(len(y_pr)):
    if ((y_pr[i] == 1) and (y_tr[i] == 1)):
        tp = tp +1
    elif ((y_pr[i] == 0) and (y_tr[i] == 1)):
        fn = fn +1
    elif ((y_pr[i] == 0) and (y_tr[i] == 0)):
        tn = tn +1
    else:
        fp = fp +1
recall = tp/(tp+fn)
precision = tp/(tp+fp)
specificity = tn/(tn + fp)
accuracy = (tp+tn)/(tp+fp+tn+fn)
f1_score = 2*recall*precision/(precision + recall)
```

```
recall, precision, specificity, accuracy, f1_score
```

```
(0.9311127693016019,
 0.8851993841881245,
 0.5069683023270889,
 0.8476675083531002,
 0.9075757692992162)
```

We see that tuning the threshold gives us a better model with respect to specificity and f1 score.

Confusion matrix for the base model

```
confusion_matrix = pd.DataFrame({"Actual positives": [tp,fn], "Actual negatives": [fp,tn]})
confusion_matrix.index = ["Predicted positives", "Predicted negatives"]
```

```
confusion_matrix
```

	Actual positives	Actual negatives
Predicted positives	58648	7606
Predicted negatives	4339	7821

```
100*confusion_matrix/(tp+tn+fp+fn)
```

	Actual positives	Actual negatives
Predicted positives	74.792767	9.699799
Predicted negatives	5.533451	9.973984

We observe one very important point here - the model is not able to classify negative class. This is shown by a large false positive value from the confusion matrix. This is also proved by low value of specificity. The key reason for this is that negative class is the minority class making up just 20% of the total data.

To overcome this we will have to do 2 things:

1. Oversample the minority class using SMOTE
2. Balancing the weight of minority class in the cost function.

```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/sit
Requirement already satisfied: numpy>=1.17.3 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-p
Requirement already satisfied: scipy>=1.5.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pa
Requirement already satisfied: scikit-learn>=1.0.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
Requirement already satisfied: joblib>=1.1.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-p
Requirement already satisfied: threadpoolctl>=2.0.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12

[notice] A new release of pip is available: 24.0 -> 24.1.2
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

```
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import SMOTE
```

We will only sample the minority class to 50% of the majority class. The reason being, if we had done too much oversampling, the model will overfit.

```
# We are using SMOTE + ENN model
smote = SMOTE(sampling_strategy = 0.5)
smote_enn = SMOTEENN(smote = smote)
```

```
x_train_s, y_train_s = smote_enn.fit_resample(x_train, y_train)
```

```
x_train
```

	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	loan_a
34680	0.134742	1.861027	1.284699	1.616962	-0.326887	1.560922	1.927
4557	-0.508132	-0.725103	-1.525402	0.523430	1.108524	-1.842867	0.225
73535	-0.025633	-0.270836	-0.332918	0.807576	0.850858	-1.399227	0.883
251815	-1.008226	-1.105712	-1.525402	-0.031858	0.263711	-1.540481	0.270
186849	1.053953	-0.921165	1.284699	0.293704	-0.927835	0.600118	-0.252
...	...	...	...	...	...	...	
210801	1.106544	-1.028049	-0.098196	-0.956983	-0.806581	-0.888663	1.302
374815	1.342036	0.262046	-0.869301	-1.441984	1.291829	-0.059481	1.778
197150	0.339573	-0.296752	-1.525402	0.082765	0.908975	-0.772655	-0.252
11556	-0.310274	-0.013638	0.118420	0.628501	0.070374	-1.008973	-0.731
39226	-0.508132	-1.286622	0.506486	-0.484329	0.395641	1.147736	-0.252

313654 rows × 16 columns

```
# We have the oversampled data
x_train_s
```

	annual_inc	dti	open_acc	revol_bal	revol_util	total_acc	loan_a
0	-1.008226	-1.105712	-1.525402	-0.031858	0.263711	-1.540481	0.270
1	0.713702	0.467747	1.000450	0.151194	-0.073249	0.745960	2.403
2	-1.468813	-0.747367	-0.869301	0.021530	0.560876	-1.008973	-0.611
3	-2.581226	-1.208251	-1.915469	-1.628079	-0.640389	-2.547860	-0.827
4	1.268269	-0.707071	-1.915469	-1.812563	1.158481	-2.356778	-0.492
...	...	...	...	...	...	...	
264021	-1.920064	-0.413382	-1.915469	-1.026202	0.921434	-1.842867	-1.090
264022	-0.958631	-0.398850	0.506486	0.298743	0.354388	-0.552393	-0.252
264023	-0.170479	0.277473	0.118420	0.686518	1.504737	0.447389	-0.492
264024	1.342036	0.262046	-0.869301	-1.441984	1.291829	-0.059481	1.778
264025	0.339573	-0.296752	-1.525402	0.082765	0.908975	-0.772655	-0.252

264026 rows × 16 columns

```
y_train.value_counts()
```

```
loan_status
1    252354
0     61300
Name: count, dtype: int64
```

```
y_train_s.value_counts()
```

```
loan_status
1    175584
0     88442
Name: count, dtype: int64
```

```
def specificity(y_true, y_pred):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    tn = np.sum(y_pred[y_true == 0]==0)
    fp = np.sum(y_pred[y_true == 0]==1)
    return tn/(tn+fp)
```

Let's again train a new model which is able to balance the minority class in cost function and hopefully give us better results

```
model_weight_balance = LogisticRegression(class_weight = "balanced")
kfold = KFold(n_splits = 5)
scoring = {
    'recall': 'recall'
```

```
    'recall': 'recall',
    'precision': 'precision',
    'accuracy': 'accuracy',
    'f1_score': 'f1',
    'specificity': specificity
}

results = cross_validate(estimator = model_weight_balance, X = x_train_s, y = y_train_s, scoring = scoring, cv = kfold)
results
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1471:
  _warn_prf(average, modifier, msg_start, len(result))
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_47161/4233513908.py:4: RuntimeWarning: invalid value encounte
  return tn/(tn+fp)
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_47161/4233513908.py:4: RuntimeWarning: invalid value encounte
  return tn/(tn+fp)
/var/folders/tx/1rbx7xzs2xn_hvqwj21v8cth0000gn/T/ipykernel_47161/4233513908.py:4: RuntimeWarning: invalid value encounte
  return tn/(tn+fp)
{'fit_time': array([0.40029788, 0.40426588, 0.38127017, 0.40839815, 0.40671206]),
 'score_time': array([0.29715204, 0.19349003, 0.05885482, 0.03536892, 0.03951192]),
 'test_recall': array([0.        , 0.92381618, 0.9172995 , 0.91889026, 0.91839788]),
 'test_precision': array([0.        , 0.75124331, 1.        , 1.        , 1.        ]),
 'test_accuracy': array([0.89715184, 0.87576934, 0.9172995 , 0.91889026, 0.91839788]),
 'test_f1_score': array([0.        , 0.82864009, 0.95686615, 0.95773091, 0.9574634 ]),
 'test_specificity': array([0.89715184, 0.85262095,        nan,        nan,        nan])}
```

This is giving us very good result, but the issue here is we are evaluating it on train data where there is a high chance of overfitting. Let us test on test data.

```
model_weight_balance = LogisticRegression(class_weight = "balanced")
model_weight_balance.fit(x_train_s, y_train_s)
```

```
▾            LogisticRegression
LogisticRegression(class_weight='balanced')
```

```
y_pred_train = model_weight_balance.predict(x_train_s)
y_pred_test = model_weight_balance.predict(x_test)
```

```
print("recall train = %f, recall test = %f" %(recall_score(y_train_s, y_pred_train), recall_score(y_test, y_pred_test)))
print("accuracy train = %f, accuracy test = %f" %(accuracy_score(y_train_s, y_pred_train), accuracy_score(y_test, y_pred_tes
print("precision train = %f, precision test = %f" %(precision_score(y_train_s, y_pred_train), precision_score(y_test, y_prec
print("f1 score train = %f, f1 score test = %f" %(f1_score(y_train_s, y_pred_train), f1_score(y_test, y_pred_test)))
print("specificity train = %f, specificity test = %f" %(specificity(y_train_s, y_pred_train), specificity(y_test, y_pred_tes
```

```
recall train = 0.918142, recall test = 0.789677
accuracy train = 0.904820, accuracy test = 0.790968
precision train = 0.937448, precision test = 0.940546
f1 score train = 0.927694, f1 score test = 0.858534
specificity train = 0.878372, specificity test = 0.796241
```

The model is now much better at specificity. But there is a big difference in train and test data metrics. This hints at overfitting.

Let us create another model without smote and just using the weight balance. This will hopefully give us better results without overfitting.

```
model2 = LogisticRegression(class_weight = "balanced")
model2.fit(x_train, y_train)
```

```
▾            LogisticRegression
LogisticRegression(class_weight='balanced')
```

```
y_pred_train = model2.predict(x_train)
y_pred_test = model2.predict(x_test)
```

```
print("recall train = %f, recall test = %f" %(recall_score(y_train, y_pred_train), recall_score(y_test, y_pred_test)))
print("accuracy train = %f, accuracy test = %f" %(accuracy_score(y_train, y_pred_train), accuracy_score(y_test, y_pred_test)
print("precision train = %f, precision test = %f" %(precision_score(y_train, y_pred_train), precision_score(y_test, y_pred_t
print("f1 score train = %f, f1 score test = %f" %(f1_score(y_train, y_pred_train), f1_score(y_test, y_pred_test)))
print("specificity train = %f, specificity test = %f" %(specificity(y_train, y_pred_train), specificity(y_test, y_pred_test)
```

```
recall train = 0.808379, recall test = 0.808412
accuracy train = 0.801278, accuracy test = 0.801898
precision train = 0.935896, precision test = 0.936250
f1 score train = 0.867476, f1 score test = 0.867647
specificity train = 0.772044, specificity test = 0.775308
```

Model 2 gives us very good result on both positive and negative class without overfitting.

```
# This is the base model metrics
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred_train = model.predict(x_train)
y_pred_test = model.predict(x_test)

print("recall train = %f, recall test = %f" %(recall_score(y_train, y_pred_train), recall_score(y_test, y_pred_test)))
print("accuracy train = %f, accuracy test = %f" %(accuracy_score(y_train, y_pred_train), accuracy_score(y_test, y_pred_test)
print("precision train = %f, precision test = %f" %(precision_score(y_train, y_pred_train), precision_score(y_test, y_pred_t
print("f1 score train = %f, f1 score test = %f" %(f1_score(y_train, y_pred_train), f1_score(y_test, y_pred_test)))
print("specificity train = %f, specificity test = %f" %(specificity(y_train, y_pred_train), specificity(y_test, y_pred_test)
```

```
recall train = 0.947550, recall test = 0.946542
accuracy train = 0.840907, accuracy test = 0.840641
precision train = 0.867056, precision test = 0.867207
f1 score train = 0.905518, f1 score test = 0.905139
specificity train = 0.401863, specificity test = 0.408360
```

Let us train another model without ENN and with just regular SMOTE.

```
smote = SMOTE(sampling_strategy = 0.5)
x_train_s, y_train_s = smote.fit_resample(x_train, y_train)
model_weight_balance = LogisticRegression(class_weight = "balanced")
kfold = KFold(n_splits = 5)
scoring = {
    'recall': 'recall',
    'precision': 'precision',
    'accuracy': 'accuracy',
    'f1_score': 'f1',
    'specificity': specificity
}

results = cross_validate(estimator = model_weight_balance, X = x_train_s, y = y_train_s, scoring = scoring, cv = kfold)
results
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:8
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_scorer.py", lin
    score = scorer(estimator, *args, **routed_params.get(name).score)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: specificity() takes 2 positional arguments but 3 were given

  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:8
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_scorer.py", lin
    score = scorer(estimator, *args, **routed_params.get(name).score)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: specificity() takes 2 positional arguments but 3 were given

  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:8
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_scorer.py", lin
    score = scorer(estimator, *args, **routed_params.get(name).score)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: specificity() takes 2 positional arguments but 3 were given

  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:8
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_scorer.py", lin
    score = scorer(estimator, *args, **routed_params.get(name).score)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: specificity() takes 2 positional arguments but 3 were given

  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:8
```

```
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_scorer.py", lin
    score = scorer(estimator, *args, **routed_params.get(name).score)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: specificity() takes 2 positional arguments but 3 were given

  warnings.warn(
{'fit_time': array([0.53799176, 0.57677698, 0.58870912, 0.5776062 , 0.57711601]),
 'score_time': array([0.05669618, 0.0458169 , 0.04612303, 0.04545593, 0.35125613]),
 'test_recall': array([0.81310887, 0.81227443, 0.80920342, 0.81100447, 0.80641453]),
 'test_precision': array([0.93376295, 0.93648692, 0.93677867, 0.93262525, 0.32829465]),
 'test_accuracy': array([0.80369054, 0.80449628, 0.80198661, 0.80100915, 0.78817018]),
 'test_f1_score': array([0.8692692 , 0.86996934, 0.86833022, 0.86757325, 0.46662454]),
 'test_specificity': array([nan, nan, nan, nan, nan])}
```

```
model_weight_balance.fit(x_train_s, y_train_s)
```

```
▾ LogisticRegression
LogisticRegression(class_weight='balanced')
```

```
y_pred_train = model_weight_balance.predict(x_train_s)
y_pred_test = model_weight_balance.predict(x_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 y_pred_train = model_weight_balance.predict(x_train_s)
      2 y_pred_test = model_weight_balance.predict(x_test)

NameError: name 'model_weight_balance' is not defined
```

```
print("recall train = %f, recall test = %f" %(recall_score(y_train_s, y_pred_train), recall_score(y_test, y_pred_test)))
print("accuracy train = %f, accuracy test = %f" %(accuracy_score(y_train_s, y_pred_train), accuracy_score(y_test, y_pred_tes
print("precision train = %f, precision test = %f" %(precision_score(y_train_s, y_pred_train), precision_score(y_test, y_prec
print("f1 score train = %f, f1 score test = %f" %(f1_score(y_train_s, y_pred_train), f1_score(y_test, y_pred_test)))
print("specificity train = %f, specificity test = %f" %(specificity(y_train_s, y_pred_train), specificity(y_test, y_pred_tes
```

```
recall train = 0.810847, recall test = 0.811031
accuracy train = 0.799746, accuracy test = 0.802790
precision train = 0.879373, precision test = 0.934814
f1 score train = 0.843721, f1 score test = 0.868535
specificity train = 0.777544, specificity test = 0.769151
```

This model is surprisingly performing really well, but we are compromising on specificity.

So the best model that gave us the best result is Model2 which is having weight balancing.

```
from sklearn.metrics import confusion_matrix
model2 = LogisticRegression(class_weight = "balanced")
model2.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(class_weight='balanced')
```

```
y_pred_train = model2.predict(x_train)
y_pred_test = model2.predict(x_test)
```

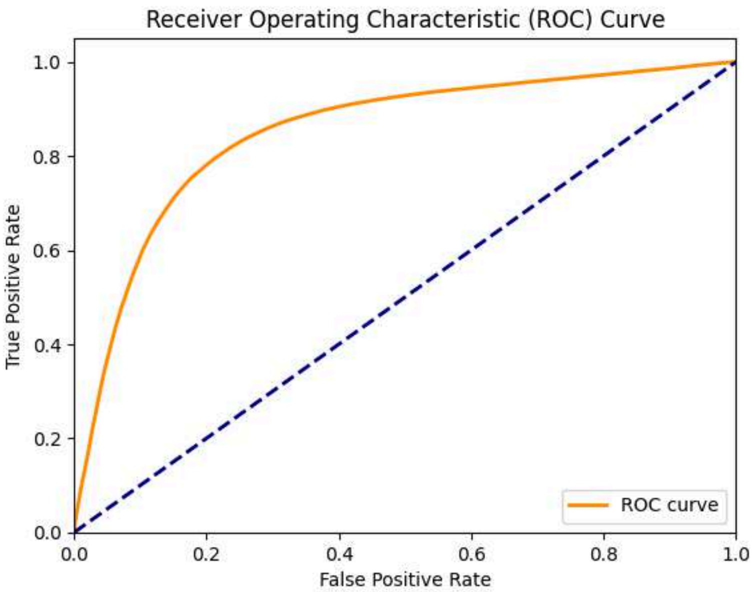
Let us tune the threshold.

```
y_pred = model2.predict_proba(x_train)[:,1]
```

```
tpr = []
fpr = []
for t in threshold:
    y_tr = np.array(y_train)
    y_pr = np.where(y_pred>=t,1,0)
    tp = 0
    fp = 0
    tn = 0
```

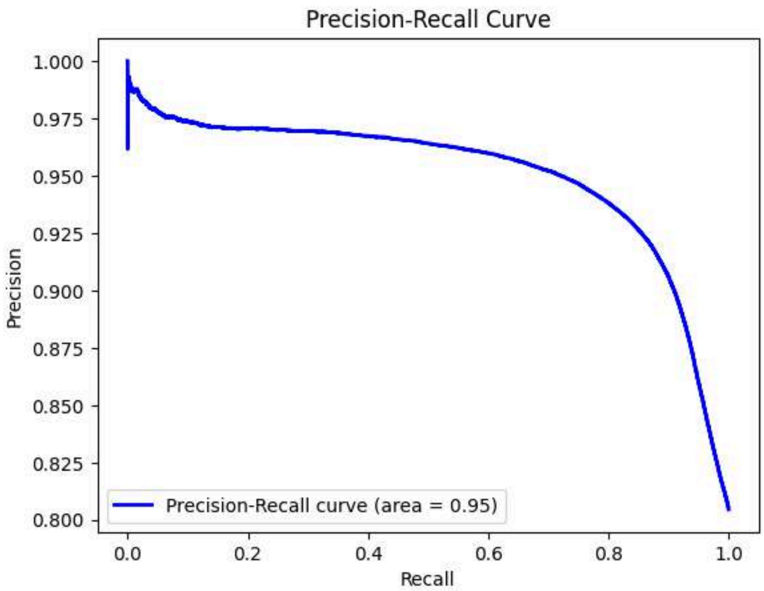
```
    fn = 0
    for i in range(len(y_pr)):
        if ((y_pr[i] == 1) and (y_tr[i] == 1)):
            tp = tp +1
        elif ((y_pr[i] == 0) and (y_tr[i] == 1)):
            fn = fn +1
        elif ((y_pr[i] == 0) and (y_tr[i] == 0)):
            tn = tn +1
        else:
            fp = fp +1
    fpr.append(fp/(fp+tn))
    tpr.append(tp/(tp+fn))


plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve' )
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```




```
from sklearn.metrics import precision_recall_curve, average_precision_score
precision, recall, _ = precision_recall_curve(y_train, y_pred)
average_precision = average_precision_score(y_train, y_pred)

plt.figure()
plt.plot(recall, precision, color='b', lw=2, label='Precision-Recall curve (area = %0.2f)' % average_precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```

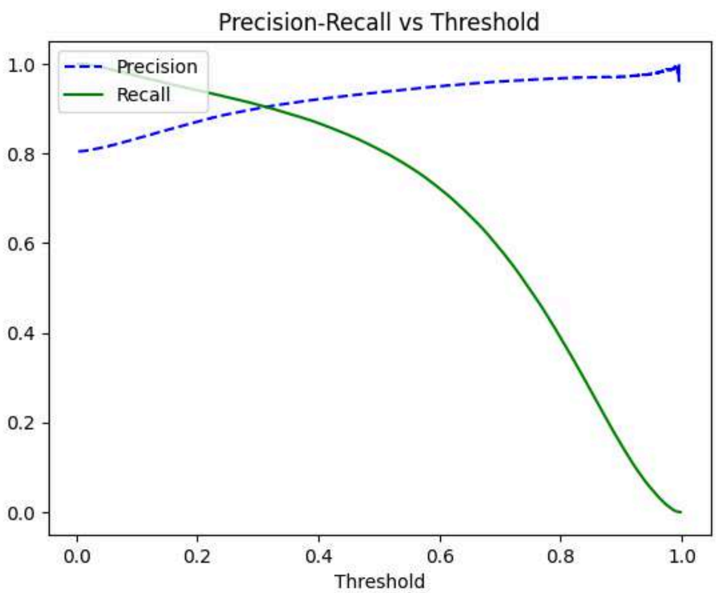



```
precision, recall, thresholds = precision_recall_curve(y_train, y_pred)
f1_scores = 2*recall*precision / (recall + precision)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]

print('Best Threshold by F1 Score:', optimal_threshold)

# Plotting Precision-Recall Curve
plt.plot(thresholds, precision[:-1], 'b--', label='Precision')
plt.plot(thresholds, recall[:-1], 'g-', label='Recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.title('Precision-Recall vs Threshold')
plt.show()
```

```
Best Threshold by F1 Score: 0.24148540285277867
```



The optimum threshold is 0.24 for this model.

```
y_pr = model2.predict(x_test)
y_tr = np.array(y_test)
tp = 0
fp = 0
tn = 0
fn = 0
for i in range(len(y_pr)):
    if ((y_pr[i] == 1) and (y_tr[i] == 1)):
        tp = tp +1
    elif ((y_pr[i] == 0) and (y_tr[i] == 1)):
        fn = fn +1
```

```
recall, precision, specificity, accuracy, f1_score
```

```
(0.8083353179328412,
 0.9345607401152759,
 0.7689416034739776,
 0.8005840793735812,
 0.8668772294254361)
```

```
accuracy = (tp+tn)/(tp+fp+tn+fn)
```

```
confusion_matrix = pd.DataFrame({"Actual positives": [tp,fn], "Actual negatives": [fp,tn]})
confusion_matrix.index = ["Predicted positives", "Predicted negatives"]
```

```
100*confusion_matrix/(tp + tn + fp + fn)
```

	Actual positives	Actual negatives
Predicted positives	64.928457	4.546382
Predicted negatives	15.395210	15.129951

From the confusion matrix we can see that we have significantly reduced the false positives but at the same time we are able to maintain the true positive and true negatives.

```
df
```

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...
...	...	...	...	...	...	...	...	...	...	...	...
396025	10000.0	60 months	10.99	217.38	B	B4	licensed bankere	2 years	RENT	40000.0	...
396026	21000.0	36 months	12.29	700.42	C	C1	Agent	5 years	MORTGAGE	110000.0	...
396027	5000.0	36 months	9.99	161.32	B	B1	City Carrier	10+ years	RENT	56500.0	...
396028	21000.0	60 months	15.31	503.02	C	C2	Gracon Services, Inc	10+ years	MORTGAGE	64000.0	...
396029	2000.0	36 months	13.61	67.98	C	C2	Internal Revenue Service	10+ years	RENT	42996.0	...

396030 rows × 27 columns

Questionnaire