In [1]:
```
pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages
(from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-pa
ckages (from category_encoders) (1.5.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages
(from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-pack
ages (from category_encoders) (0.14.3)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages
(from category_encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages
(from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
(from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
(from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from pats
y>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages
(from scikit-learn>=0.20.0->category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-pa
ckages (from scikit-learn>=0.20.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-package
s (from statsmodels>=0.9.0->category_encoders) (24.1)
Downloading category_encoders-2.6.4-py2.py3-none-any.whl (82 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.0/82.0 kB 1.9 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.4
```

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sbn
import matplotlib.pyplot as plt
import category_encoders as ce
from sklearn.preprocessing import OrdinalEncoder,OneHotEncoder,  LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

from lightgbm import LGBMClassifier, LGBMRegressor
import warnings
warnings.filterwarnings('ignore')
from sklearn.impute import SimpleImputer
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from scipy import stats
from statsmodels.graphics.gofplots import qqplot
```

```
from scipy.stats import pearsonr
from scipy.stats import shapiro
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
```

In [2]:

In [3]:
```
#!gdown https://drive.google.com/file/d/1ZPYj7CZCfxntE8p2Lze_4QO4MyEOy6_d/view?usp=shar
```

In [4]:
```
#dj = pd.read_csv('logistic_regression.csv')
#dj.head()
```

In [5]:
```
#To see all columns in dataset
pd.set_option('display.max_columns', None)
```

In [6]:
```
dl=pd.read_csv("/content/sample_data/logistic_regression.csv")
dl.head()
```

Out[6]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownersh
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	REI
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGA
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	REI
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	REI
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGA

In [7]:
```
dl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
```

```
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object
 5   sub_grade             396030 non-null  object
 6   emp_title             373103 non-null  object
 7   emp_length            377729 non-null  object
 8   home_ownership        396030 non-null  object
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object
 11  issue_d               396030 non-null  object
 12  loan_status           396030 non-null  object
 13  purpose               396030 non-null  object
 14  title                 394274 non-null  object
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

In [8]:
```
dl.describe()
```

Out[8]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000	396030
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153	
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649	
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000	
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000	
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000	
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000	
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000	86

In [9]:
```
dl.shape
```

Out[9]: (396030, 27)

In [10]:
```
dl['mort_acc'] = dl['mort_acc'].astype(object)
```

In [11]:
```
dl['pub_rec_bankruptcies'].unique()
```

Out[11]:
```
array([ 0.,  1.,  2.,  3., nan,  4.,  5.,  6.,  7.,  8.])
```

In [12]:
```
dl['revol_util'].unique()
```

Out[12]:
```
array([ 41.8 ,  53.3 ,  92.2 , ...,  56.26, 111.4 , 128.1 ])
```

In [13]:
```
C = dl.columns
missing_value = pd.DataFrame({
    'Missing Value': dl.isnull().sum(),
    'Percentage': (dl.isnull().sum() / len(dl))*100,
    'Unique Value': dl.nunique(),
    'data types': dl.dtypes,
})
missing_value[missing_value['Missing Value']>0].sort_values(by='Percentage', ascending=
```

Out[13]:

	Missing Value	Percentage	Unique Value	data types
mort_acc	37795	9.543469	33	object
emp_title	22927	5.789208	173105	object
emp_length	18301	4.621115	11	object
title	1756	0.443401	48816	object
pub_rec_bankruptcies	535	0.135091	9	float64
revol_util	276	0.069692	1226	float64

In [136…
```
successful_loans = dl[dl['loan_status'].isin(['Fully Paid'])]
```

In [137…
```
top_affording_job_titles = successful_loans['emp_title'].value_counts().head(2)

print(top_affording_job_titles)
```

```
emp_title
Teacher    3513
Manager    3304
Name: count, dtype: int64
```

In [14]:
```
dl.duplicated().sum()
```

Out[14]:
```
0
```

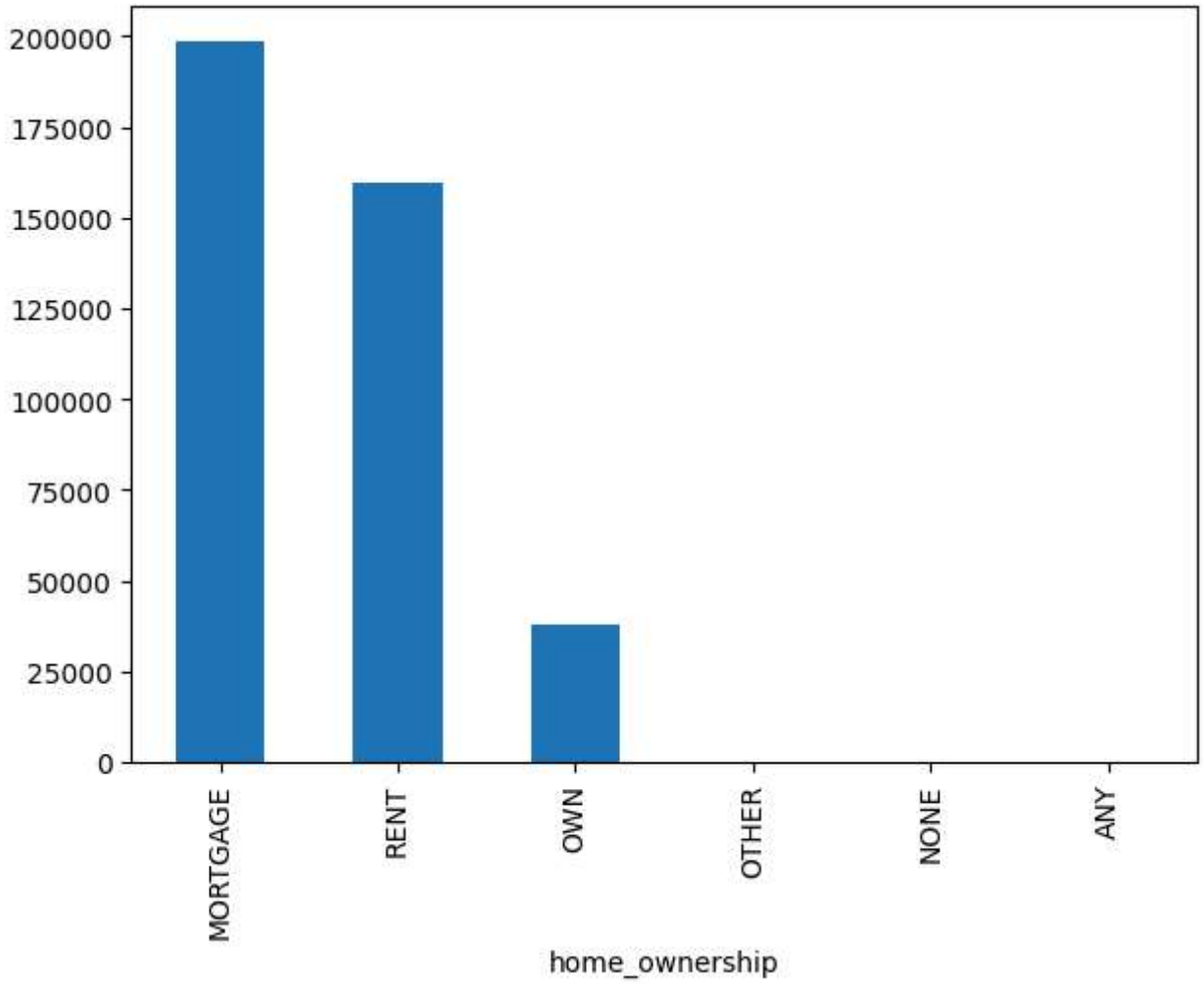
In [15]:
```
dls = dl['loan_status'].value_counts().sort_values(ascending=False)
```

In [16]:
```
dls[0]/(dls[0]+dls[1])
```

Out[16]: 0.8038709188697826

In [17]:
```
plt.figure(figsize=(7, 5))
dho = dl['home_ownership'].value_counts()
dho.plot(kind='bar')
```

Out[17]: <Axes: xlabel='home_ownership'>



In [18]:
```
dl.groupby('grade')['loan_status'].value_counts(normalize=True)
```

Out[18]:

		proportion
grade	**loan_status**	
A	**Fully Paid**	0.937121
	Charged Off	0.062879
B	**Fully Paid**	0.874270
	Charged Off	0.125730
C	**Fully Paid**	0.788191
	Charged Off	0.211809
D	**Fully Paid**	0.711322

		proportion
grade	loan_status	
	Charged Off	0.288678
E	Fully Paid	0.626366
	Charged Off	0.373634
F	Fully Paid	0.572120
	Charged Off	0.427880
G	Fully Paid	0.521611
	Charged Off	0.478389

dtype: float64

In [19]:
```
dl['emp_title'].value_counts()
```

Out[19]:

	count
emp_title	
Teacher	4389
Manager	4250
Registered Nurse	1856
RN	1846
Supervisor	1830
...	...
Postman	1
McCarthy & Holthus, LLC	1
jp flooring	1
Histology Technologist	1
Gracon Services, Inc	1

173105 rows × 1 columns

dtype: int64

In [20]:
```
dl.head()
```

Out[20]:

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

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

In [21]:
```
data = dl
data[['mort_acc','emp_title','emp_length','title']].describe([0.05,0.25,0.50,0.75,0.90,
```

Out[21]:

	mort_acc	emp_title	emp_length	title
count	358235.0	373103	377729	394274
unique	33.0	173105	11	48816
top	0.0	Teacher	10+ years	Debt consolidation
freq	139777.0	4389	126041	152472

In [22]:
```
data.dropna(inplace = True)
```

In [23]:
```
data['emp_title'].value_counts(dropna=False)
```

Out[23]:

emp_title	count
Teacher	4367
Manager	4231
Registered Nurse	1841
RN	1840
Supervisor	1812
...	...
Sales Managment	1
Dispatch supervor	1
Olympic companies	1

emp_title	count
Atrium Medical	1
Gracon Services, Inc	1

150832 rows × 1 columns

dtype: int64

In [24]:
```
data['title'].value_counts(dropna=False)
```

Out[24]:

title	count
Debt consolidation	144270
Credit card refinancing	48438
Home improvement	14154
Other	11951
Debt Consolidation	8986
...	...
Credit Card Refianace Loan	1
Reduce number of creditors to get ahead	1
PhD Debt Payoff	1
1999dodge	1
Loanforpayoff	1

32186 rows × 1 columns

dtype: int64

In [25]:
```
cols_with_nans = [x for x in data.columns if data[x].isnull().sum() > 200]
#for col in cols_with_nans:
 #   data[col].fillna(data[col].mode()[0], inplace=True)
```

In [26]:
```
cols_with_nans
```

Out[26]: []

In [27]:
```
C = data.columns
missing_value = pd.DataFrame({
    'Missing Value': data.isnull().sum(),
    'Percentage': (data.isnull().sum() / len(data))*100,
```

```
    'Unique Value': data.nunique(),
    'data types': data.dtypes,
})
missing_value[missing_value['Missing Value']>0].sort_values(by='Percentage', ascending=
```

Out[27]:

	Missing Value	Percentage	Unique Value	data types

In [28]:

```
#data['mort_acc'] = data['mort_acc'].fillna(data['mort_acc'].mode()[0])
#data['emp_length'] = data['emp_length'].fillna(data['emp_length'].mode()[0])
#data['emp_title'] = data['emp_title'].fillna(data['emp_title'].mode()[0])
#data['title'] = data['title'].fillna(data['title'].mode()[0])
```

In [29]:

```
#data['pub_rec_bankruptcies'].dropna(inplace = True)
#data['revol_util'].dropna(inplace = True)
#data['emp_length'] = pd.factorize(data['emp_length'])[0]
```

In [30]:

```
#cat_with_null = cols_with_nans
#for cat in cat_with_null:
 #   data[cat] = pd.factorize(data[cat])[0]***
```

In [31]:

```
'''for col in cat_with_null:
  nan_ixs = data[col][data[col].isnull()].index
  data['is_nan'] = 0
  data.loc[nan_ixs, 'is_nan'] = 1



  train = data[data['is_nan'] == 0]
  test = data[data['is_nan'] == 1]

  X_train = data.drop(['is_nan'], axis=1)
  y_train = data['is_nan']
  X_test = test.drop(['is_nan'], axis=1)
  model = LGBMClassifier(random_state=0)
  model.fit(X_train, y_train)

  y_pred = model.predict(X_test)
y_pred

freq_imputer = SimpleImputer(strategy = 'most_frequent') # mode
for col in cat_with_null:
    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[col])))
'''
```

Out[31]:

```
"for col in cat_with_null:\n  nan_ixs = data[col][data[col].isnull()].index\n  data['is_
nan'] = 0\n  data.loc[nan_ixs, 'is_nan'] = 1\n\n\n\n  train = data[data['is_nan'] == 0]
\n  test = data[data['is_nan'] == 1]\n\n  X_train = data.drop(['is_nan'], axis=1)\n  y_t
rain = data['is_nan']\n  X_test = test.drop(['is_nan'], axis=1)\n  model = LGBMClassifie
r(random_state=0)\n  model.fit(X_train, y_train)\n\n  y_pred = model.predict(X_test)\ny_
pred\n\nfreq_imputer = SimpleImputer(strategy = 'most_frequent') # mode\nfor col in cat_
with_null:\n    data[col] = pd.DataFrame(freq_imputer.fit_transform(pd.DataFrame(data[co
l])))\n"
```

In [32]:
```
C = data.columns
col_value = pd.DataFrame({
    'Missing Value': data.isnull().sum(),
    'Unique Value': data.nunique(),
    'data types': data.dtypes,
})
col_value.sort_values(by = 'Unique Value',ascending=False).head()
```

Out[32]:

	Missing Value	Unique Value	data types
address	0	334142	object
emp_title	0	150832	object
revol_bal	0	53310	float64
installment	0	48696	float64
title	0	32186	object

In [33]:
```
cat_cols  = data[col_value[(col_value['Unique Value'] <=100) | (col_value['data types']
cat_cols.head()
```

Out[33]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	issue_d	lc
0	36 months	B	B4	Marketing	10+ years	RENT	Not Verified	Jan-2015	
1	36 months	B	B5	Credit analyst	4 years	MORTGAGE	Not Verified	Jan-2015	
2	36 months	B	B3	Statistician	< 1 year	RENT	Source Verified	Jan-2015	
3	36 months	A	A2	Client Advocate	6 years	RENT	Not Verified	Nov-2014	
4	60 months	C	C5	Destiny Management Inc.	9 years	MORTGAGE	Verified	Apr-2013	

In [34]:
```
num_cols =  data[col_value[(col_value['Unique Value'] >=100) & (col_value['data types']
num_cols.head()
```

Out[34]:

	loan_amnt	int_rate	installment	annual_inc	dti	revol_bal	revol_util	total_acc
0	10000.0	11.44	329.48	117000.0	26.24	36369.0	41.8	25.0
1	8000.0	11.99	265.68	65000.0	22.05	20131.0	53.3	27.0
2	15600.0	10.49	506.97	43057.0	12.79	11987.0	92.2	26.0

	loan_amnt	int_rate	installment	annual_inc	dti	revol_bal	revol_util	total_acc
3	7200.0	6.49	220.65	54000.0	2.60	5472.0	21.5	13.0
4	24375.0	17.27	609.33	55000.0	33.95	24584.0	69.8	43.0

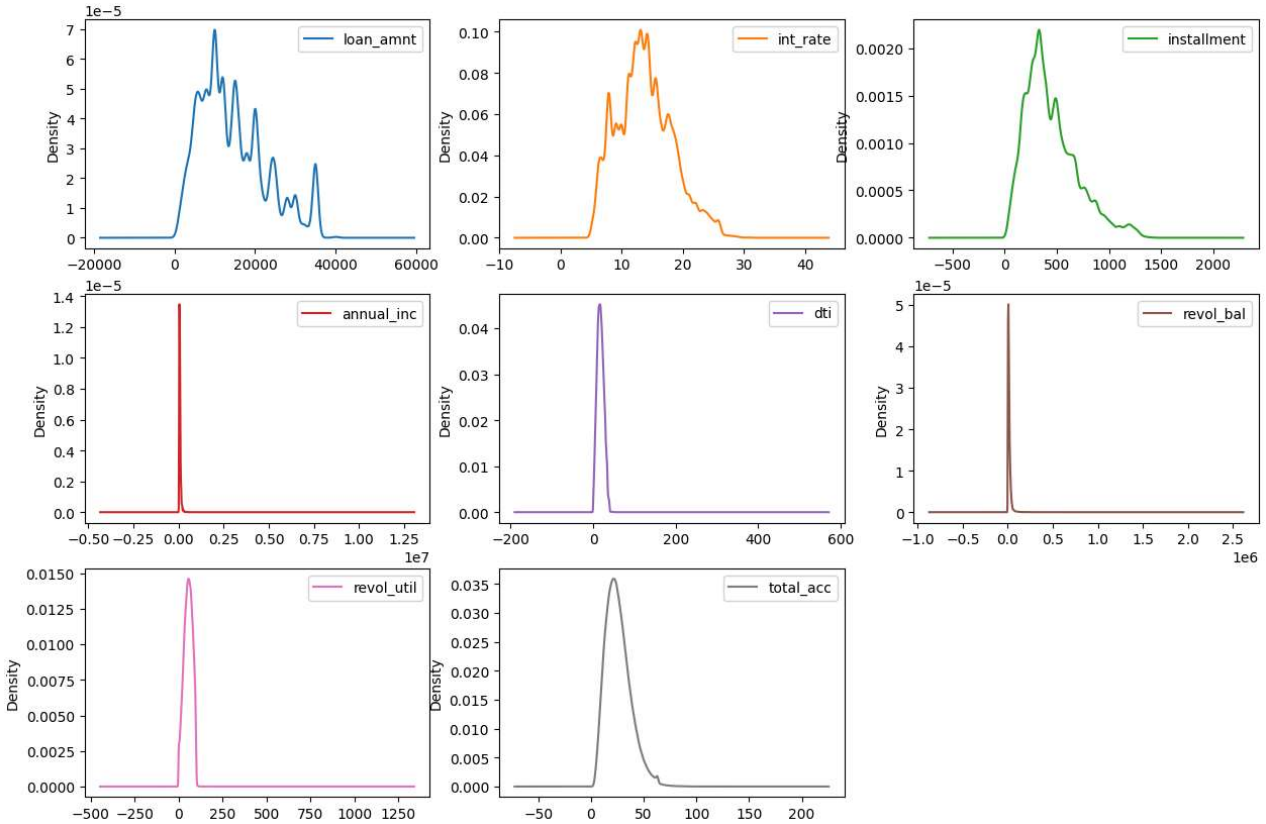
In [35]:
```
num_cols.columns
```

Out[35]:
```
Index(['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti',
       'revol_bal', 'revol_util', 'total_acc'],
      dtype='object')
```

In [36]:
```
cat_cols.columns
```

Out[36]:
```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'title', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

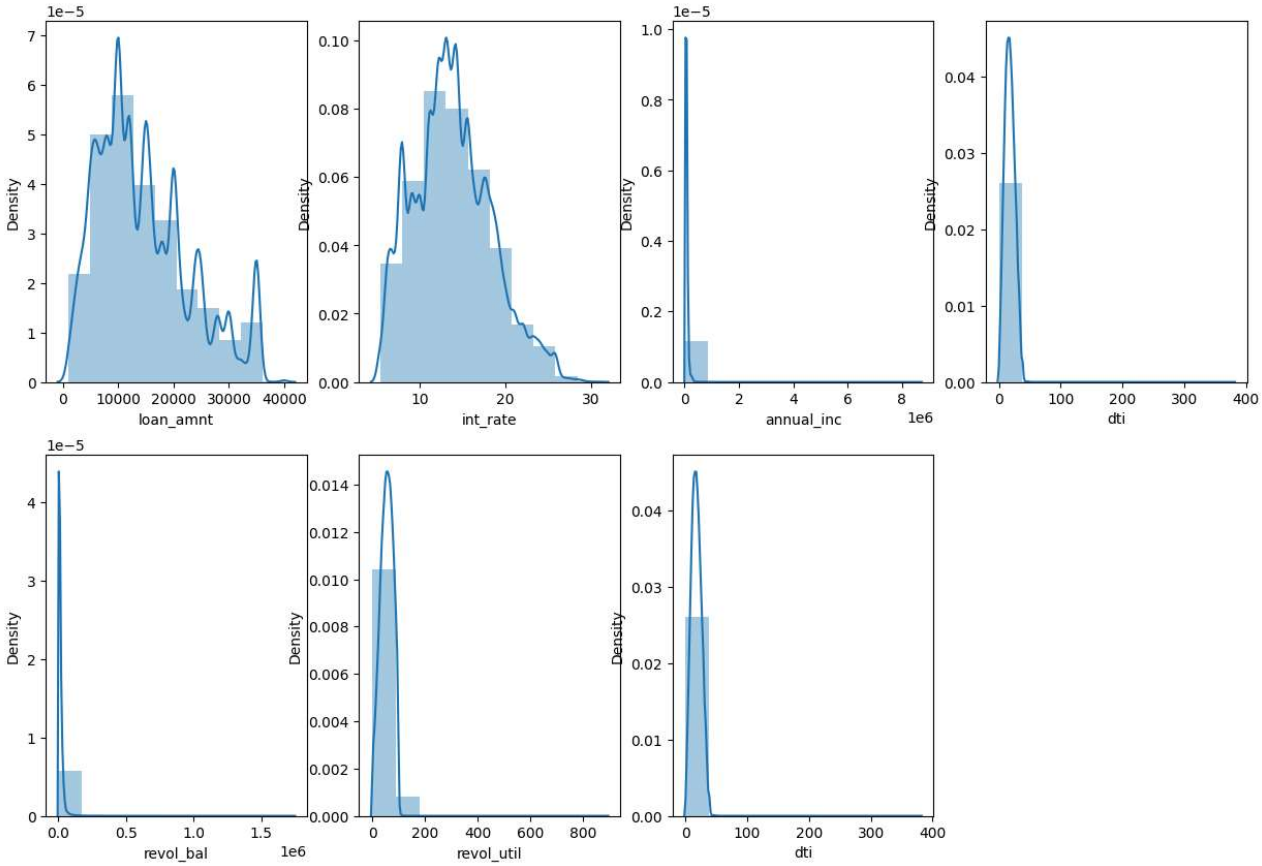
In [37]:
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [15,10]
num_cols.plot(kind = 'density', subplots = True, layout = (3,3), sharex = False)
plt.show()
```



In [38]:
```
plt.figure(figsize=(15, 10))

#Histogram
```

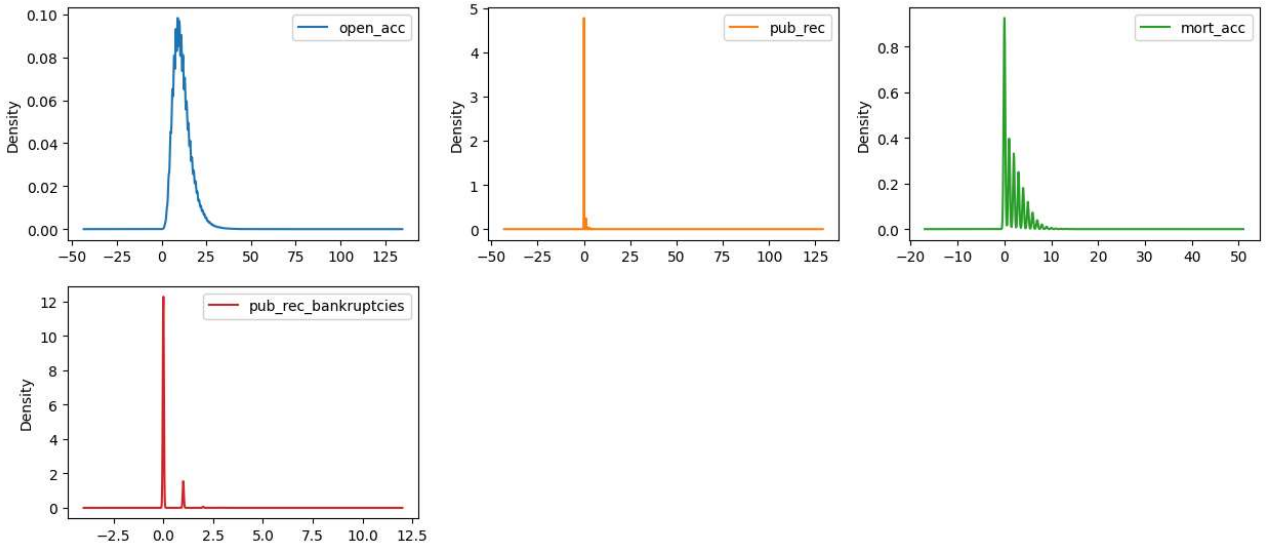
```
plt.subplot(2, 4 ,1)
sbn.distplot(num_cols['loan_amnt'], bins = 10)

plt.subplot(2, 4 ,2)
sbn.distplot(num_cols['int_rate'], bins = 10)

plt.subplot(2, 4 ,3)
sbn.distplot(num_cols['annual_inc'], bins = 10)

plt.subplot(2, 4,4)
sbn.distplot(num_cols['dti'], bins = 10)

plt.subplot(2, 4 ,5)
sbn.distplot(num_cols['revol_bal'], bins = 10)

plt.subplot(2, 4 ,6)
sbn.distplot(num_cols['revol_util'], bins = 10)

plt.subplot(2,4 ,7)
sbn.distplot(num_cols['dti'], bins = 10)
```

Out[38]: <Axes: xlabel='dti', ylabel='Density'>



In [39]:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = [15,10]
cat_cols.plot(kind = 'density', subplots = True, layout = (3,3), sharex = False)
plt.show()
```

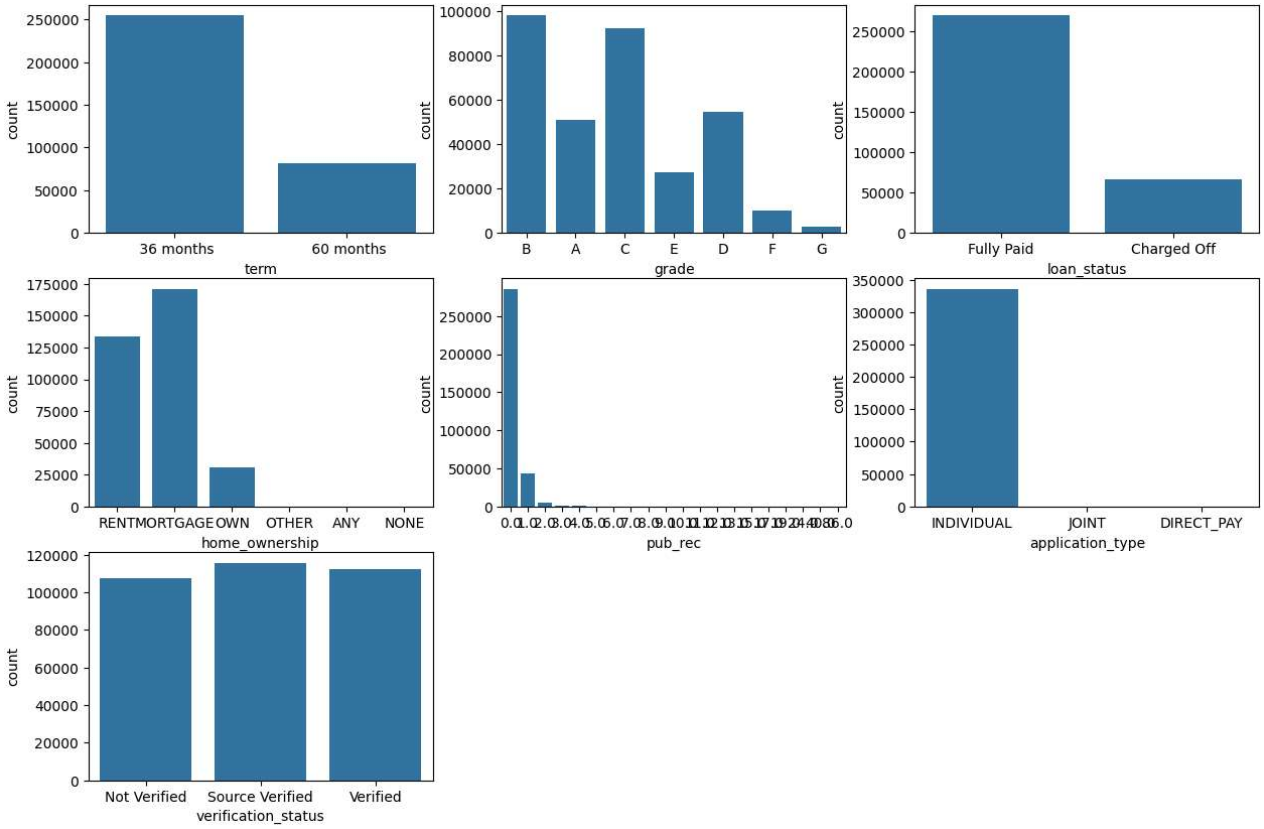


In [40]:
```
cat_cols.columns
```

Out[40]:
```
Index(['term', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       'home_ownership', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'title', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'initial_list_status', 'application_type', 'mort_acc',
       'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

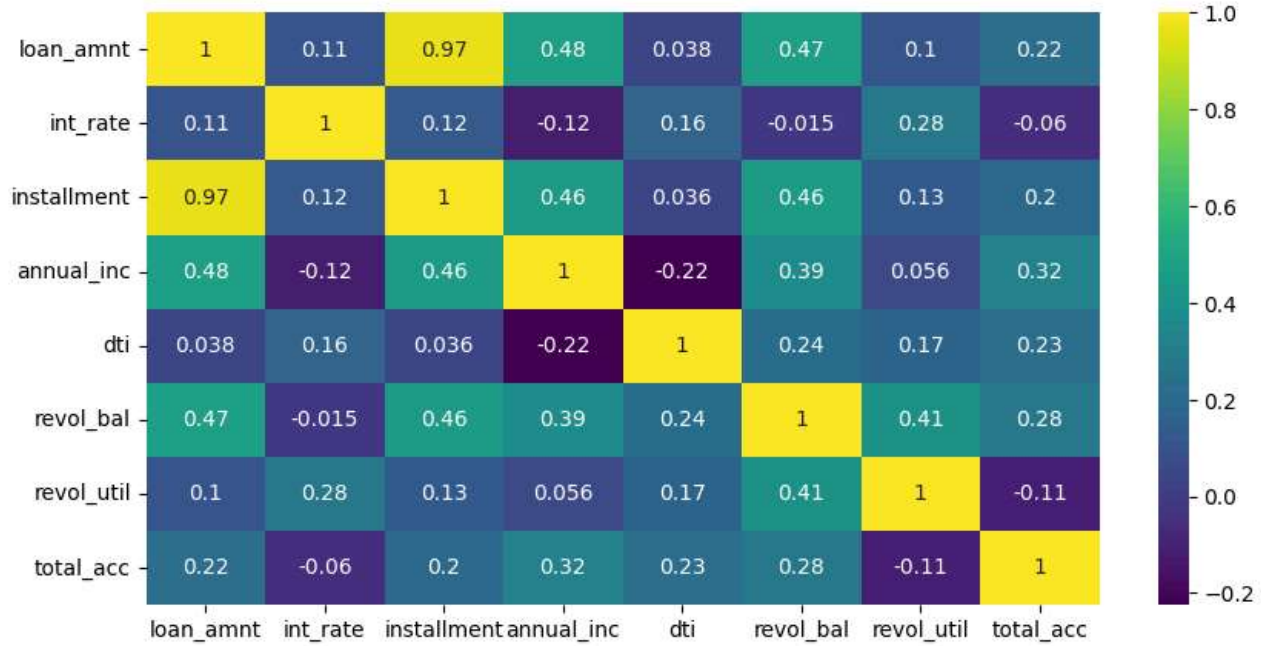
In [41]:
```
#for col in cat_cols:
 # sbn.countplot(data=cat_cols, x=col)

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

#Histogram
plt.subplot(3, 3 ,1)
sbn.countplot(data=cat_cols, x=cat_cols['term'])

plt.subplot(3, 3 ,2)
sbn.countplot(data=cat_cols, x=cat_cols['grade'])

plt.subplot(3, 3 ,3)
sbn.countplot(data=cat_cols, x=cat_cols['loan_status'])

plt.subplot(3, 3,4)
sbn.countplot(data=cat_cols, x=cat_cols['home_ownership'])

plt.subplot(3, 3,5)
sbn.countplot(data=cat_cols, x=cat_cols['pub_rec'])

plt.subplot(3, 3 ,6)
sbn.countplot(data=cat_cols, x=cat_cols['application_type'])

plt.subplot(3,3,7)
sbn.countplot(data=cat_cols, x=cat_cols['verification_status'])
```

Out[41]:
```
<Axes: xlabel='verification_status', ylabel='count'>
```



In [42]:

```
plt.figure(figsize=(10, 5))
sbn.heatmap(num_cols.corr(method='spearman'),
            annot=True, cmap='viridis')
plt.show()
```

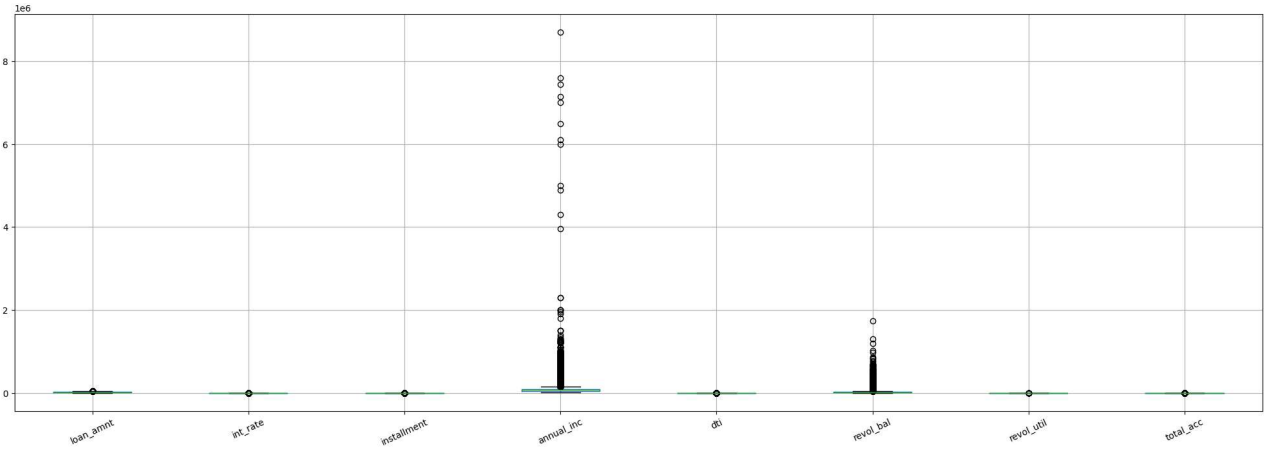


In [43]:

```
data[['issue_d_month', 'issue_d_year']] = data['issue_d'].str.split('-', expand=True)
data[['ecr_month', 'ecr_year']] = data['earliest_cr_line'].str.split('-', expand=True)
data.drop(['issue_d', 'earliest_cr_line'], axis=1, inplace=True)
```

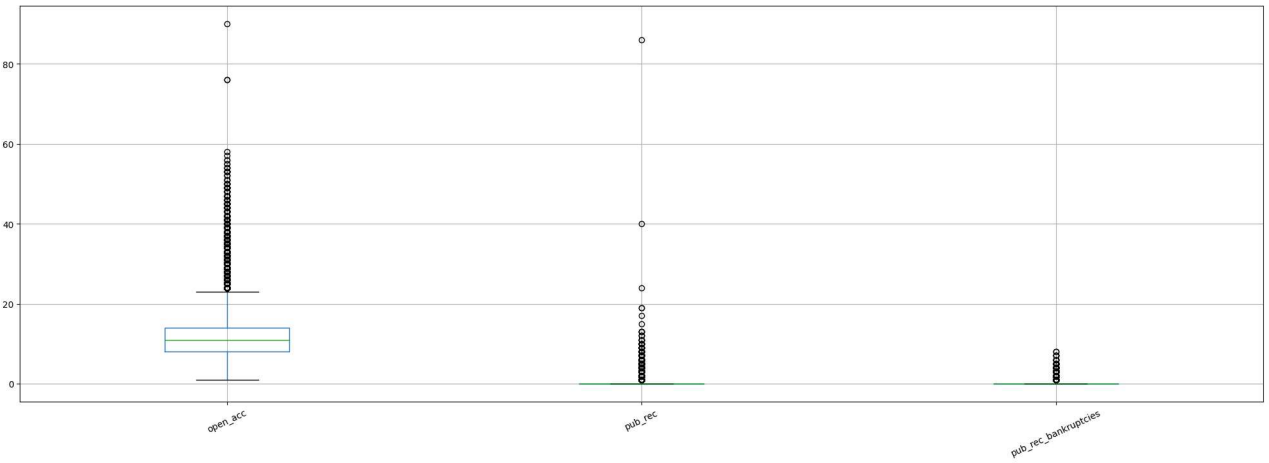
In [44]:
```
num_cols.boxplot(rot=25, figsize=(25,8))
```

Out[44]: <Axes: >



In [45]:
```
cat_cols.boxplot(rot=25, figsize=(25,8))
```

Out[45]: <Axes: >



In [46]:
```
# obtain the first quartile
Q1 = num_cols.quantile(0.3)

# obtain the third quartile
Q3 = num_cols.quantile(0.65)

# obtain the IQR
IQR = Q3 - Q1

# print the IQR
IQR
```

Out[46]:

	0
loan_amnt	6825.00
int_rate	4.17
installment	206.61

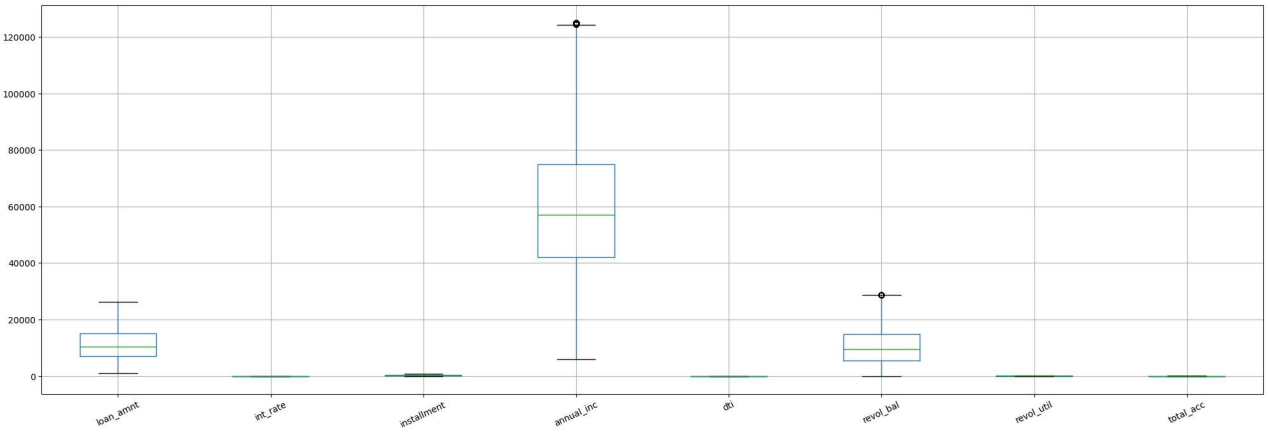
	0
annual_inc	30000.00
dti	7.94
revol_bal	8571.00
revol_util	24.80
total_acc	10.00

dtype: float64

In [47]:
```
dq = num_cols[~((num_cols < (Q1 - 1.5 * IQR)) |(num_cols > (Q3 + 1.5 * IQR))).any(axis=
```

In [48]:
```
dq.boxplot(rot=25, figsize=(25,8))
```

Out[48]: <Axes: >



In [49]:
```
data = data[~((num_cols < (Q1 - 1.5 * IQR)) |(num_cols > (Q3 + 1.5 * IQR))).any(axis=1)
```

Ranking: grade, sub_grade,

In [50]:
```
data['sub_grade'].unique()
```

Out[50]:
```
array(['B5', 'B3', 'A2', 'C3', 'A1', 'B2', 'B4', 'A5', 'C1', 'A3', 'D1',
       'C2', 'B1', 'C5', 'D5', 'D2', 'E2', 'A4', 'E3', 'D3', 'C4', 'D4',
       'E4', 'E1', 'E5', 'F5'], dtype=object)
```

In [51]:
```
def extract_state(address):
    # Clean the address by replacing newline characters
    clean_address = address.replace('\r\n', ', ')

    # Split the address to get the last part
    parts = clean_address.split(',')

    if len(parts) > 1:
        # Get the second to last part, which should be the state and ZIP
        state_zip = parts[-1].strip()
        # Split to get state
```

```
        state = state_zip.split()[0]  # First part is the state
        return state
    return None
```

In [52]:
```
data.isnull().sum()
d3 = data.copy()
```

In [53]:
```
data = d3
```

In [54]:
```
data['state'] = data['address'].apply(extract_state)
```

In [55]:
```
data.head()
```

Out[55]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownersh
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGA
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RE
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RE
5	20000.0	36 months	13.33	677.07	C	C3	HR Specialist	10+ years	MORTGA
6	18000.0	36 months	5.32	542.07	A	A1	Software Development Engineer	2 years	MORTGA

In [56]:
```
#df['make'] = df.groupby('make')['selling_price'].transform('mean')
#df['model'] = df.groupby('model')['selling_price'].transform('mean')

#from sklearn.preprocessing import
```

In [57]:
```
grade_order = data['grade'].unique().tolist()
grade_order.sort()
sub_grade_order = data['sub_grade'].unique().tolist()
sub_grade_order.sort()
```

In [58]:
```
ordinal_encoder = OrdinalEncoder(categories=[grade_order, sub_grade_order])

data[['grade','sub_grade']] = ordinal_encoder.fit_transform(data[['grade','sub_grade']]
```

In [59]:
```
data.isnull().sum().head()
```

Out[59]:

	0
loan_amnt	0
term	0
int_rate	0
installment	0
grade	0

dtype: int64

In [60]:
```
#OH_encoded = col_value[col_value['Unique Value']<=5].index.tolist()
```

In [61]:
```
# Apply one-hot encoding
#OHencoder = OneHotEncoder(sparse_output=False)  # Use sparse=True for large datasets
#for i in OH_encoded:
 # one_hot_encoded = OHencoder.fit_transform(data[[i]])
  #one_hot_encoded_df = pd.DataFrame(one_hot_encoded, columns=OHencoder.get_feature_name
  #data = pd.concat([data, one_hot_encoded_df], axis=1)
#data.head()
#data.drop(i, axis=1, inplace=True)
```

In [62]:
```
encoded2 = col_value[(col_value['Unique Value']>=5) & (col_value['Unique Value'] <= 30)
col_value.sort_values(by = 'Unique Value',ascending=False)#[col_value['Unique Value']>=
ENC = encoded2.index.tolist()
for i in ENC:
  print('Column= '+ i)
  print('Unique value = ',data[i].unique().tolist())
  print('Value count = ',data[i].value_counts(dropna=False))
  print('-----------------------------------')
```

```
Column= grade
Unique value =  [1.0, 0.0, 2.0, 3.0, 4.0, 5.0]
Value count =  grade
1.0    71316
2.0    63768
3.0    35892
0.0    33402
4.0    11733
5.0        2
Name: count, dtype: int64
-----------------------------------
Column= emp_length
Unique value =  ['4 years', '< 1 year', '6 years', '10+ years', '2 years', '7 years', '9
years', '8 years', '5 years', '3 years', '1 year']
Value count =  emp_length
10+ years    68415
2 years      21100
< 1 year     18365
3 years      18333
5 years      15607
```

```
1 year       15345
4 years      13573
7 years      12516
6 years      12371
8 years      11432
9 years       9056
Name: count, dtype: int64
---------------------------------
Column= home_ownership
Unique value =  ['MORTGAGE', 'RENT', 'OWN', 'OTHER', 'NONE']
Value count =  home_ownership
MORTGAGE    98708
RENT        97818
OWN         19545
OTHER          24
NONE           18
Name: count, dtype: int64
---------------------------------
Column= purpose
Unique value =  ['debt_consolidation', 'credit_card', 'home_improvement', 'small_busines
s', 'major_purchase', 'vacation', 'medical', 'other', 'car', 'moving', 'house', 'weddin
g', 'renewable_energy']
Value count =  purpose
debt_consolidation    132108
credit_card            48263
home_improvement       11106
other                  10377
major_purchase          4009
car                     2193
medical                 2042
small_business          1606
moving                  1405
vacation                1372
house                    840
wedding                  647
renewable_energy         145
Name: count, dtype: int64
---------------------------------
Column= pub_rec
Unique value =  [0.0, 1.0, 2.0, 4.0, 3.0, 6.0, 5.0, 10.0, 8.0, 11.0, 7.0, 19.0, 9.0, 13.
0, 86.0, 24.0, 12.0, 15.0]
Value count =  pub_rec
0.0     179561
1.0      31613
2.0       3462
3.0        903
4.0        314
5.0        129
6.0         65
7.0         27
8.0         17
10.0         7
9.0          4
11.0         3
19.0         2
12.0         2
13.0         1
86.0         1
24.0         1
15.0         1
Name: count, dtype: int64
---------------------------------
Column= pub_rec_bankruptcies
Unique value =  [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 7.0, 6.0]
Value count =  pub_rec_bankruptcies
```

```
0.0    186768
1.0     27793
2.0      1243
3.0       234
4.0        50
5.0        21
7.0         3
6.0         1
Name: count, dtype: int64
---------------------------------
```

In [63]:
```
ENC = encoded2.index.tolist()
```

In [64]:
```
data['term'].unique().tolist()
data['term'] = data['term'].apply(lambda x: 0 if x == ' 36 months' else 1)
```

In [65]:
```
def convert_employee_length(value):
    if '<' in value:  # Handle the "<1 year" case
        return 0
    elif '10+' in value:  # Handle the ">10 years" case
        return 11
    else:
        # Extract the number using regex
        return int(value.split()[0])
```

In [66]:
```
data['emp_length'] = data['emp_length'].apply(convert_employee_length)
```

In [67]:
```
data['emp_length'].unique()
```

Out[67]:
```
array([ 4,  0,  6, 11,  2,  7,  9,  8,  5,  3,  1])
```

In [68]:
```
data['loan_status'].unique()
```

Out[68]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [69]:
```
data['loan_status'] = data['loan_status'].apply(lambda x: 1 if x == 'Fully Paid' else 0
```

In [70]:
```
data['loan_status'].unique()
```

Out[70]:
```
array([1, 0])
```

In [71]:
```
data['verification_status'].unique()
```

Out[71]:
```
array(['Not Verified', 'Source Verified', 'Verified'], dtype=object)
```

In [72]:
```
data['verification_status'] = data['verification_status'].apply(lambda x: 1 if x == 'Ve
```

In [73]:
```
data['verification_status'].unique()
```

Out[73]:
```
array([0, 2, 1])
```

In [74]:
```
data
```

Out[74]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_own
1	8000.0	0	11.99	265.68	1.0	9.0	Credit analyst	4	MOR
2	15600.0	0	10.49	506.97	1.0	7.0	Statistician	0	
3	7200.0	0	6.49	220.65	0.0	1.0	Client Advocate	6	
5	20000.0	0	13.33	677.07	2.0	12.0	HR Specialist	11	MOR
6	18000.0	0	5.32	542.07	0.0	0.0	Software Development Engineer	2	MOR
...	...	...	...	...	...	...	...	...	
396020	10000.0	0	9.76	321.55	1.0	7.0	Retirement Counselor	11	
396022	12000.0	0	12.29	400.24	2.0	10.0	Data Center Specialist II	1	
396024	6000.0	0	13.11	202.49	1.0	8.0	Michael's Arts & Crafts	5	
396025	10000.0	1	10.99	217.38	1.0	8.0	licensed bankere	2	
396028	21000.0	1	15.31	503.02	2.0	11.0	Gracon Services, Inc	11	MOR

216113 rows × 30 columns

In [75]:
```
data['purpose'].unique()
```

Out[75]:
```
array(['debt_consolidation', 'credit_card', 'home_improvement',
       'small_business', 'major_purchase', 'vacation', 'medical', 'other',
```

```
       'car', 'moving', 'house', 'wedding', 'renewable_energy'],
      dtype=object)
```

In [76]:
```
data['application_type'].unique()
```

Out[76]:
```
array(['INDIVIDUAL', 'JOINT', 'DIRECT_PAY'], dtype=object)
```

In [77]:
```
data['application_type'] = data['application_type'].apply(lambda x: 1 if x == 'INDIVIDU
```

In [78]:
```
data['application_type'].unique()
```

Out[78]:
```
array([1, 2, 0])
```

In [79]:
```
data['initial_list_status'].unique()
```

Out[79]:
```
array(['f', 'w'], dtype=object)
```

In [80]:
```
data['initial_list_status'] = data['initial_list_status'].apply(lambda x: 1 if x == 'w'
```

In [81]:
```
data['initial_list_status'].unique()
```

Out[81]:
```
array([0, 1])
```

In [82]:
```
data['issue_d_year'].nunique()
```

Out[82]:
```
5
```

In [83]:
```
data['issue_d_month'].nunique()
```

Out[83]:
```
12
```

In [84]:
```
data['ecr_year'].nunique()
```

Out[84]:
```
63
```

In [85]:
```
data['ecr_month'].nunique()
```

Out[85]:
```
12
```

In [86]:
```
data.head()
```

Out[86]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership
1	8000.0	0	11.99	265.68	1.0	9.0	Credit analyst	4	MORTGAGE

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership
2	15600.0	0	10.49	506.97	1.0	7.0	Statistician	0	RENT
3	7200.0	0	6.49	220.65	0.0	1.0	Client Advocate	6	RENT
5	20000.0	0	13.33	677.07	2.0	12.0	HR Specialist	11	MORTGAGE
6	18000.0	0	5.32	542.07	0.0	0.0	Software Development Engineer	2	MORTGAGE

In [87]:
```
target = 'loan_status'
categorical_columns = ['emp_title', 'home_ownership', 'purpose','title', 'address','sta

# Initialize the TargetEncoder
encoder = ce.TargetEncoder(cols=categorical_columns)

# Fit and transform the encoder to your DataFrame
#df_encoded = data.copy()
data[categorical_columns] = encoder.fit_transform(data[categorical_columns], data[targe
```

In [88]:
```
data.head()
```

Out[88]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	a
1	8000.0	0	11.99	265.68	1.0	9.0	0.794106	4	0.843245	
2	15600.0	0	10.49	506.97	1.0	7.0	0.859658	0	0.792175	
3	7200.0	0	6.49	220.65	0.0	1.0	0.841147	6	0.792175	
5	20000.0	0	13.33	677.07	2.0	12.0	0.865638	11	0.843245	
6	18000.0	0	5.32	542.07	0.0	0.0	0.850700	2	0.843245	

In [89]:
```
month_mapping = {
    'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
    'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12
}
data['issue_d_month'] = data['issue_d_month'].map(month_mapping)
data['ecr_month'] = data['ecr_month'].map(month_mapping)
```

In [90]:
```
data.head()
```

Out[90]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	a
1	8000.0	0	11.99	265.68	1.0	9.0	0.794106	4	0.843245	
2	15600.0	0	10.49	506.97	1.0	7.0	0.859658	0	0.792175	
3	7200.0	0	6.49	220.65	0.0	1.0	0.841147	6	0.792175	
5	20000.0	0	13.33	677.07	2.0	12.0	0.865638	11	0.843245	
6	18000.0	0	5.32	542.07	0.0	0.0	0.850700	2	0.843245	

In [91]:

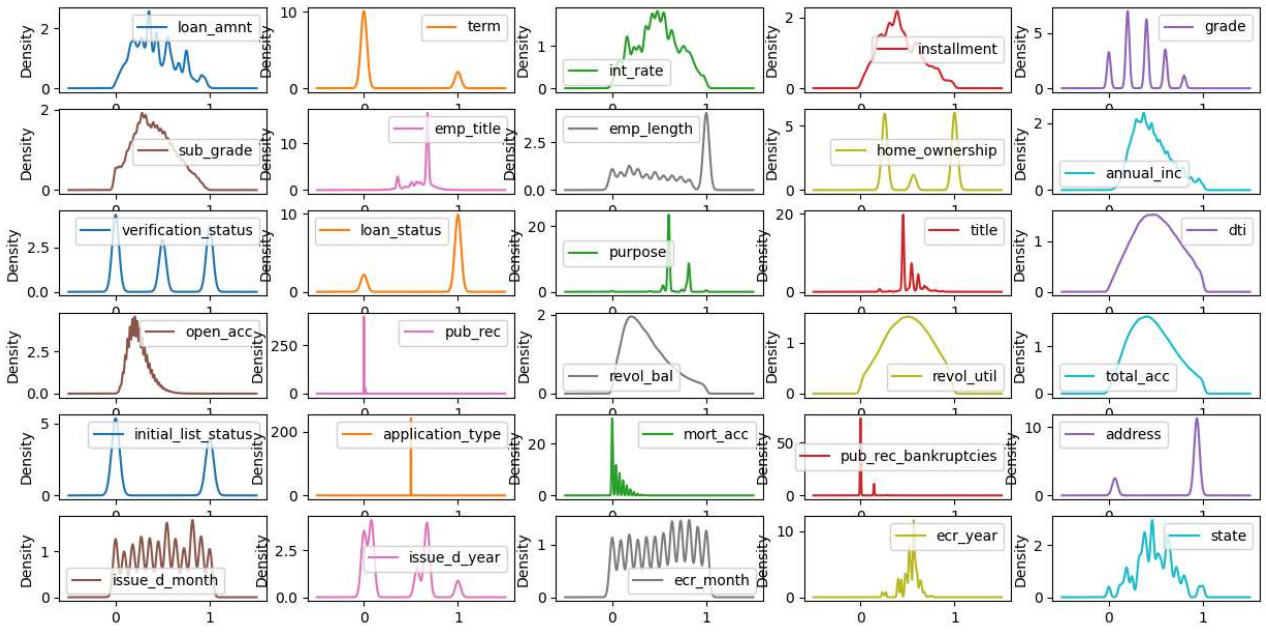
```
scaler = MinMaxScaler()
df = pd.DataFrame(scaler.fit_transform(data), columns = data.columns)
df.head()
```

Out[91]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership
0	0.277502	0.0	0.412492	0.304965	0.2	0.36	0.560749	0.363636	1.000000
1	0.578791	0.0	0.319728	0.615541	0.2	0.28	0.719652	0.000000	0.256604
2	0.245788	0.0	0.072356	0.247004	0.0	0.04	0.674780	0.545455	0.256604
3	0.753221	0.0	0.495362	0.834485	0.4	0.48	0.734150	1.000000	1.000000
4	0.673935	0.0	0.000000	0.660720	0.0	0.00	0.697939	0.181818	1.000000

In [92]:

```
plt.rcParams["figure.figsize"] = [15,10]
df.plot(kind = 'density', subplots = True, layout = (8,5), sharex = False)
plt.show()
```



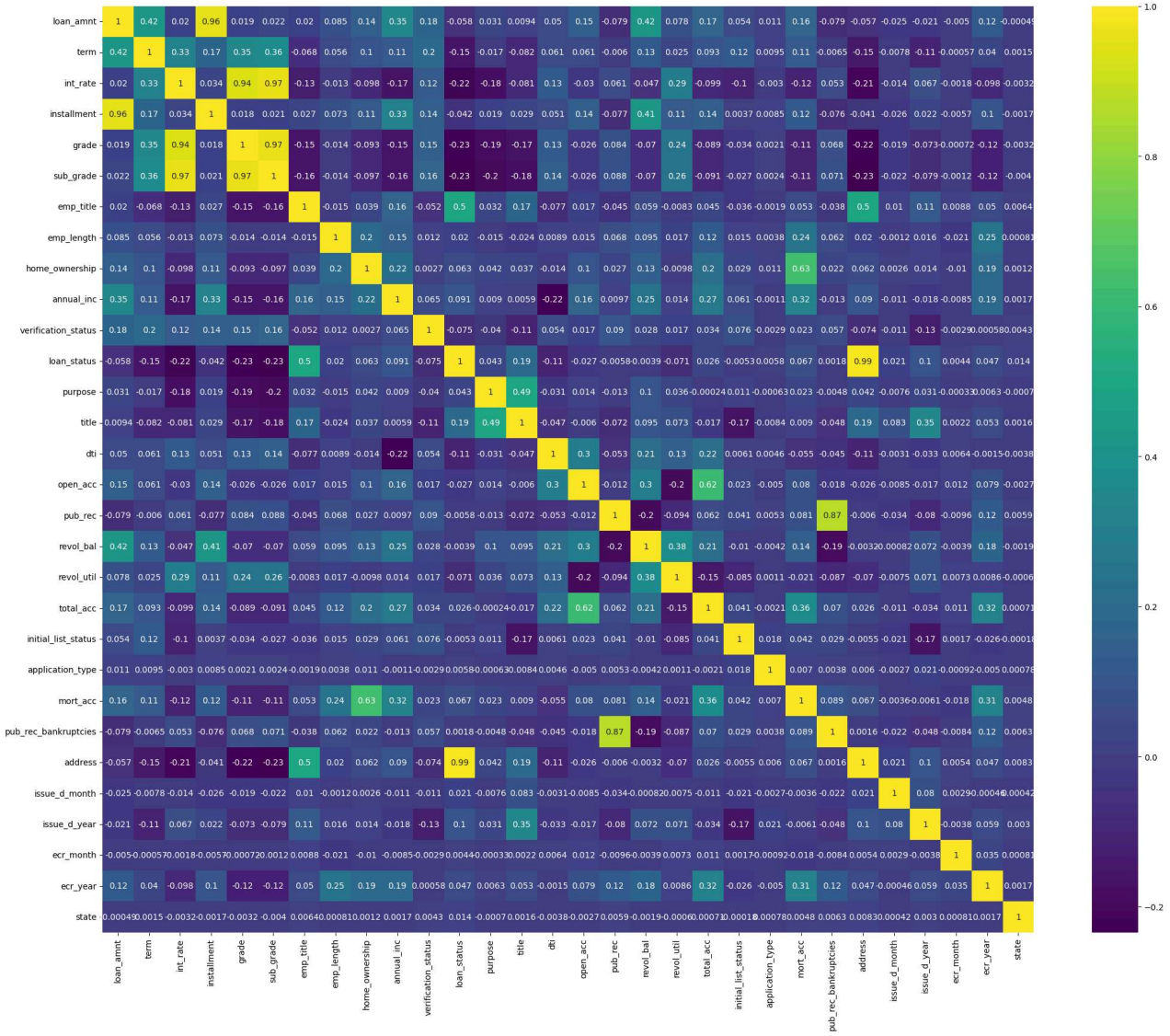
In [93]:

```
print(df.skew())
```

```
loan_amnt               0.417835
term                    1.715070
int_rate                0.033835
installment             0.410783
grade                   0.299544
sub_grade               0.261783
emp_title              -0.983796
emp_length             -0.047907
home_ownership          0.041952
annual_inc              0.596888
verification_status     0.113528
loan_status            -1.643020
purpose                -0.499409
title                   0.428292
dti                     0.085019
open_acc                0.876024
pub_rec                20.692216
revol_bal               0.691705
revol_util             -0.042505
total_acc               0.287279
initial_list_status     0.285125
application_type       39.571360
mort_acc                1.630761
pub_rec_bankruptcies    2.996681
address                -1.642568
issue_d_month          -0.077295
issue_d_year            0.381907
ecr_month              -0.141451
ecr_year               -1.101413
state                   0.105166
dtype: float64
```

In [94]:

```
plt.figure(figsize=(25, 20))
sbn.heatmap(df.corr(method='spearman'),
            annot=True, cmap='viridis')
plt.show()
```



In [95]:

```
X = df.drop('loan_status', axis=1)
# Add constant (bias term) for the model
X = add_constant(X)
y = df['loan_status']

vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display VIF values
print(vif_data)
```

	feature	VIF
0	const	1955.231024
1	loan_amnt	91.266671
2	term	11.693303
3	int_rate	29.166186
4	installment	74.317379
5	grade	15.367892
6	sub_grade	44.546271
7	emp_title	1.508334
8	emp_length	1.123960
9	home_ownership	1.388214
10	annual_inc	1.476835

```
11   verification_status     1.103014
12               purpose     1.255549
13                 title     1.509680
14                   dti     1.332098
15              open_acc     2.057723
16               pub_rec     2.054965
17             revol_bal     1.794016
18            revol_util     1.567375
19             total_acc     2.228613
20   initial_list_status     1.131606
21      application_type     1.003144
22             mort_acc     1.654897
23  pub_rec_bankruptcies     2.071734
24               address     1.552816
25         issue_d_month     1.014006
26          issue_d_year     1.558361
27             ecr_month     1.003846
28              ecr_year     1.332318
29                 state     1.000350
```

In [96]:
```
df = pd.DataFrame(data)

# Separate features and target variable
X = df.drop('loan_status', axis=1)
y = df['loan_status']

# Initialize a logistic regression model
model = LogisticRegression()

# Initialize RFE with the logistic regression model
rfe = RFE(model, n_features_to_select=2)  # Select 2 most important features

# Fit RFE
rfe = rfe.fit(X, y)

# Display selected features
print("Selected Features:", X.columns[rfe.support_])
print("Feature Ranking:", rfe.ranking_)
```

```
Selected Features: Index(['address', 'state'], dtype='object')
Feature Ranking: [26 10 12 25  8 15  3 24  6 27 11  2  7 18 14 17 28 19 21  9  5 20 13
1
 16  4 22 23  1]
```

In [97]:
```
vif_col = vif_data[vif_data["VIF"]>=5]
```

In [98]:
```
vif_col['feature'].tolist()[1:]
```

Out[98]:
```
['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade']
```

In [99]:
```
df.drop(vif_col['feature'].tolist()[1:], axis=1, inplace=True)
```

In [100…
```
def sigmoid(x):
    return 1/(1+np.e**-x)
```

In [101…
```
# Loss for a single point
def log_loss(y, y_hat):
  gain = y*np.log(y_hat)+(1-y)*np.log(1-y_hat)
  return -gain
```

In [102…
```
cols = df.columns.tolist()
cols.remove('loan_status')
y = df["loan_status"]
y = np.array(y).reshape(len(y), 1) #Reshaping our data to (m,1) shape
X = df[cols]
X.shape
```

Out[102… (216113, 23)

In [103…
```
from sklearn.model_selection import train_test_split

X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_state=1
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,rand
X_train.shape
```

Out[103… (129667, 23)

In [104…
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

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

X_train
```

Out[104…
```
array([[ 0.5998532 , -1.54137657, -1.03529485, ...,  0.35592307,
        -1.63665555,  2.50000654],
       [-2.07425273, -0.5417808 ,  1.05676204, ...,  0.64838197,
        -0.67542245,  0.79360069],
       [ 0.51530121,  1.2075118 , -1.03529485, ...,  0.35592307,
        -0.19872276,  0.05910377],
       ...,
       [-1.95800627, -1.29147763, -1.03529485, ...,  0.06346417,
        -0.67542245, -0.13608867],
       [ 0.15762143,  0.70771391,  1.05676204, ...,  1.23329978,
         0.58399678,  1.45423304],
       [-3.25376433, -1.04157869,  1.05676204, ...,  1.52575868,
         0.3684114 ,  0.41791292]])
```

In [105…
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

print("coef = ",model.coef_)
print("intercept = ",model.intercept_)
```

```
coef =  [[ 9.88086828e-01  3.05152524e-01 -1.00617151e-03 -3.47278727e-02
  -1.34453459e-01  2.21967085e-01  2.77019414e-01 -1.12768737e-01
```

```
   6.71497111e-02  1.09992601e-02 -1.67240684e-01 -1.57296847e-01
   2.33770257e-01  1.87159688e-01  1.21140536e-02  4.51868467e-03
  -3.78410843e-02  6.61513022e+00 -4.54697131e-03  1.75139488e-01
   1.34704029e-01  2.56195161e-02  3.43520226e-02]]
intercept =  [6.31204102]
```

In [106…
```
model.predict(X_train)
```

Out[106…
```
array([1, 1, 1, ..., 0, 1, 0])
```

In [107…
```
def accuracy(y_true, y_pred):
  y_true = y_true.reshape(len(y_true))
  return np.sum(y_true==y_pred)/y_true.shape[0]
```

In [108…
```
accuracy(y_train, model.predict(X_train))
```

Out[108…
```
0.9995912606908466
```

In [109…
```
accuracy(y_val, model.predict(X_val))
```

Out[109…
```
0.9993753325775628
```

In [110…
```
from sklearn.pipeline import make_pipeline
train_scores = []
val_scores = []
scaler = StandardScaler()
for la in np.arange(0.01, 5000.0, 100): # range of values of Lambda
  scaled_lr = make_pipeline(scaler, LogisticRegression(C=1/la))
  scaled_lr.fit(X_train, y_train)
  train_score = accuracy(y_train, scaled_lr.predict(X_train))
  val_score = accuracy(y_val, scaled_lr.predict(X_val))
  train_scores.append(train_score)
  val_scores.append(val_score)
```

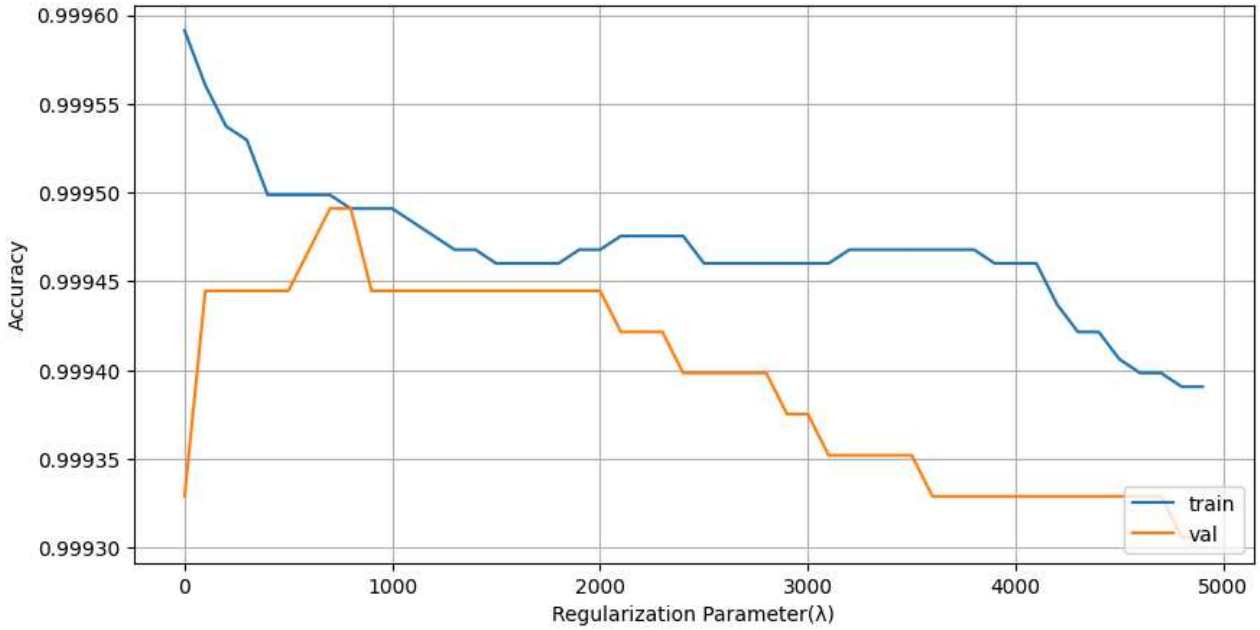
In [111…
```
(val_scores)
```

Out[111…
```
[0.9993290609166416,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994678758994054,
 0.9994910117298661,
 0.9994910117298661,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
 0.9994447400689448,
```

```
0.9994447400689448,
0.9994447400689448,
0.9994447400689448,
0.9994216042384841,
0.9994216042384841,
0.9994216042384841,
0.9993984684080235,
0.9993984684080235,
0.9993984684080235,
0.9993984684080235,
0.9993984684080235,
0.9993753325775628,
0.9993753325775628,
0.9993521967471022,
0.9993521967471022,
0.9993521967471022,
0.9993521967471022,
0.9993521967471022,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.9993290609166416,
0.999305925086181,
0.999305925086181]
```

In [112…

```
plt.figure(figsize=(10,5))
plt.plot(list(np.arange(0.01, 5000.0, 100)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 5000.0, 100)), val_scores, label="val")
plt.legend(loc='lower right')

plt.xlabel("Regularization Parameter(λ)")
plt.ylabel("Accuracy")
plt.grid()
plt.show()
```



In [113…
```
model = LogisticRegression(C=1/1000)

model.fit(X_train, y_train)

print("Train acc = ",accuracy(y_train, model.predict(X_train)))


print("Val acc = ",accuracy(y_val, model.predict(X_val)))
```

```
Train acc =  0.9994910038791675
Val acc =  0.9994447400689448
```

In [114…
```
accuracy(y_test, model.predict(X_test))
```

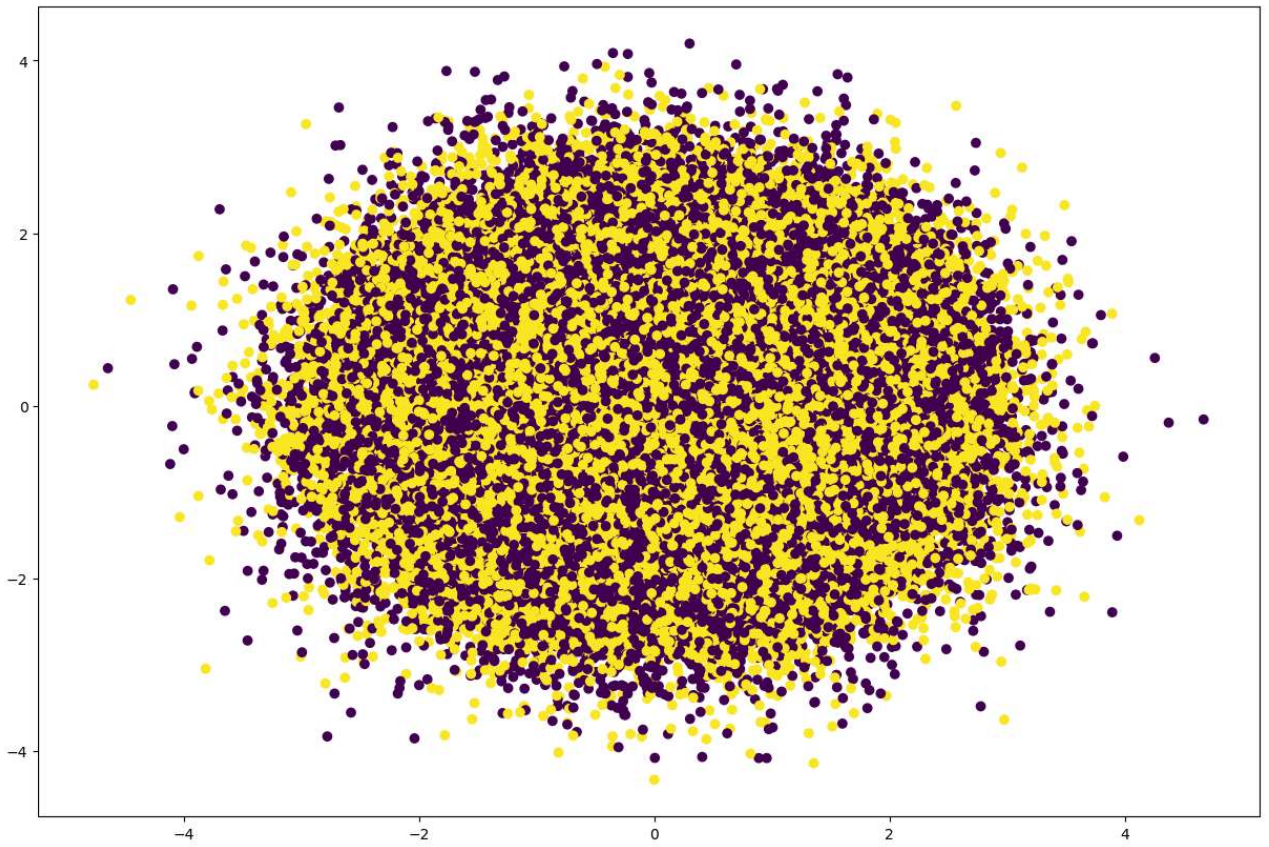
Out[114…
```
0.9997917775258542
```

In [115…
```
from sklearn.inspection import DecisionBoundaryDisplay
```

In [116…
```
df.shape
```

Out[116…
```
(216113, 24)
```

In [117…
```
# dataset creation with 3 classes
from sklearn.datasets import make_classification

X, y = make_classification(n_samples= 216113,
                           n_features= 24,
                           n_classes = 2,
                           n_redundant=0,
                           n_clusters_per_class=1,
                           random_state=5)
y=y.reshape(len(y), 1)
plt.scatter(X[:, 0], X[:, 1], c = y)
plt.show()
```



In [118…
```
X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_state=4
X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,rand
X_train.shape
```

Out[118… (129667, 24)

In [119…
```
model = LogisticRegression(multi_class='ovr')
# fit model
model.fit(X_train, y_train)


print(f'Training Accuracy:{model.score(X_train,y_train)}')
print(f'Validation Accuracy :{model.score(X_val,y_val)}')
print(f'Test Accuracy:{model.score(X_test,y_test)}')
```

```
Training Accuracy:0.9758998048848203
Validation Accuracy :0.9742035490363926
Test Accuracy:0.9758461929990977
```

In [120…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report
```

In [121…
```
''' X = df.drop('loan_status', axis=1)
y = df['loan_status']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

model = RandomForestClassifier()

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accura

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Make predictions using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))'''
```

Out[121…

```
' X = df.drop(\'loan_status\', axis=1)\ny = df[\'loan_status\']\nX_train, X_test, y_trai
n, y_test = train_test_split(X, y, test_size=0.3, random_state=42)\n\nmodel = RandomFore
stClassifier()\n\n# Define the hyperparameter grid\nparam_grid = {\n    \'n_estimators
\': [50, 100, 200],\n    \'max_depth\': [None, 10, 20],\n    \'min_samples_split\': [2,
5, 10]\n}\n\n# Initialize GridSearchCV\ngrid_search = GridSearchCV(estimator=model, para
m_grid=param_grid, cv=5, scoring=\'accuracy\')\n\n# Fit the model\ngrid_search.fit(X_tra
in, y_train)\n\n# Get the best hyperparameters\nprint("Best Hyperparameters:", grid_sear
ch.best_params_)\n\n# Make predictions using the best model\nbest_model = grid_search.be
st_estimator_\ny_pred = best_model.predict(X_test)\n\n# Evaluate the model\nprint("Accur
acy:", accuracy_score(y_test, y_pred))'
```

In [122…

```
X = df.drop('loan_status', axis=1)
y = df['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4



# Define the model
model = RandomForestClassifier()

# Define the hyperparameter space
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'bootstrap': [True, False]
}

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_i

# Fit the model
random_search.fit(X_train, y_train)
```

```
# Get the best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

# Make predictions using the best model
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Best Hyperparameters: {'n_estimators': 100, 'min_samples_split': 2, 'max_depth': 20, 'bo
otstrap': True}
Accuracy: 0.9997532159052349
```

In [123…

```
X = df.drop('loan_status', axis=1)
y = df['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Define Logistic Regression model
model = LogisticRegression()

# Define the grid of hyperparameters
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],     # Regularization strength (inverse of lambda)
    'penalty': ['l1', 'l2'],          # Regularization type (L1: Lasso, L2: Ridge)
    'solver': ['liblinear', 'saga']    # Solvers compatible with both L1 and L2 penalti
}

# Initialize GridSearchCV with Logistic Regression model and hyperparameter grid
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accur

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Make predictions using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
```

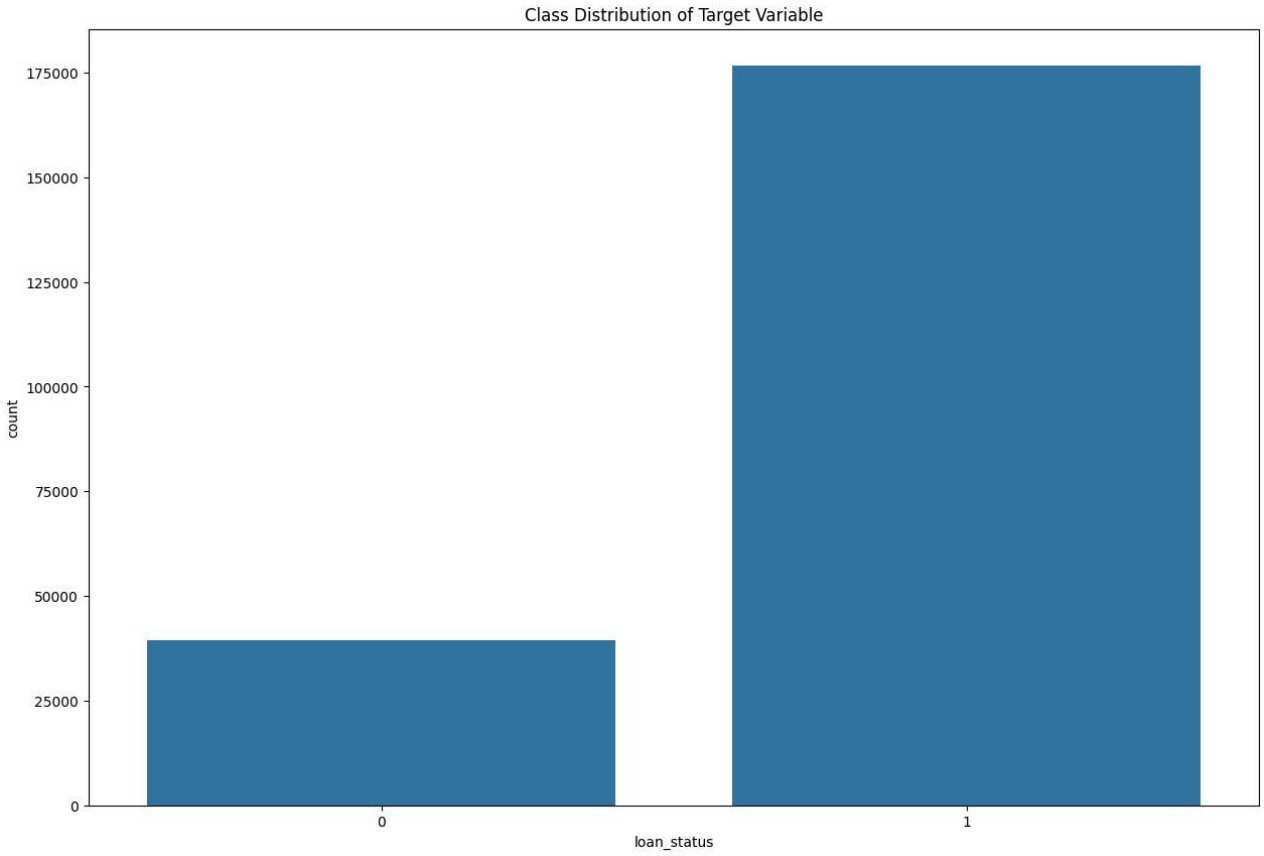
```
Best Hyperparameters: {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
Accuracy: 0.999491007804547
```

In [124…

```
# Check the distribution of the target variable
print(df['loan_status'].value_counts())

# Visualize the target variable distribution
sbn.countplot(x='loan_status', data=df)
plt.title('Class Distribution of Target Variable')
plt.show()
```

```
loan_status
1    176648
```

```
0     39465
Name: count, dtype: int64
```



In [125…

```
# Apply SMOTE for oversampling the minority class
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Train a model on the resampled data
model = RandomForestClassifier(random_state=42)
model.fit(X_resampled, y_resampled)

# Predictions and evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11778
           1       1.00      1.00      1.00     53056

    accuracy                           1.00     64834
   macro avg       1.00      1.00      1.00     64834
weighted avg       1.00      1.00      1.00     64834
```

In [126…

```
# Define Logistic Regression model with class_weight='balanced'
model = LogisticRegression(class_weight='balanced', random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)
```

```
# Evaluate the model
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.24      0.61      0.34     11778
           1       0.87      0.56      0.68     53056

    accuracy                           0.57     64834
   macro avg       0.55      0.59      0.51     64834
weighted avg       0.75      0.57      0.62     64834
```

In [127…

```
# Splitting the dataset
X = df.drop('loan_status', axis=1)
y = df['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Train Logistic Regression model with balanced class weights
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

# Extract the coefficients
coefficients = model.coef_[0]  # Get the coefficients for the features
feature_names = X.columns  # Get the feature names

# Create a dataframe to display coefficients with their corresponding feature names
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

# Sort by the absolute value of the coefficient to see the most important features
coef_df['abs_Coefficient'] = coef_df['Coefficient'].abs()
coef_df = coef_df.sort_values(by='abs_Coefficient', ascending=False)

# Display the coefficients
print(coef_df[['Feature', 'Coefficient']])
```

```
                 Feature  Coefficient
8               open_acc    -0.036623
18        issue_d_month     0.031694
7                   dti    -0.028576
4    verification_status    -0.021148
17              address     0.019812
20            ecr_month     0.019078
15             mort_acc     0.018009
12            total_acc     0.013421
0             emp_title     0.012124
1            emp_length     0.011610
11           revol_util    -0.009772
6                 title     0.004404
19         issue_d_year     0.003348
9               pub_rec    -0.003090
14     application_type     0.002148
2        home_ownership     0.002033
5               purpose     0.002015
21            ecr_year     0.001912
22               state     0.001742
13   initial_list_status    -0.001485
16  pub_rec_bankruptcies    -0.000651
10            revol_bal     0.000019
3            annual_inc     0.000008
```

In [128…

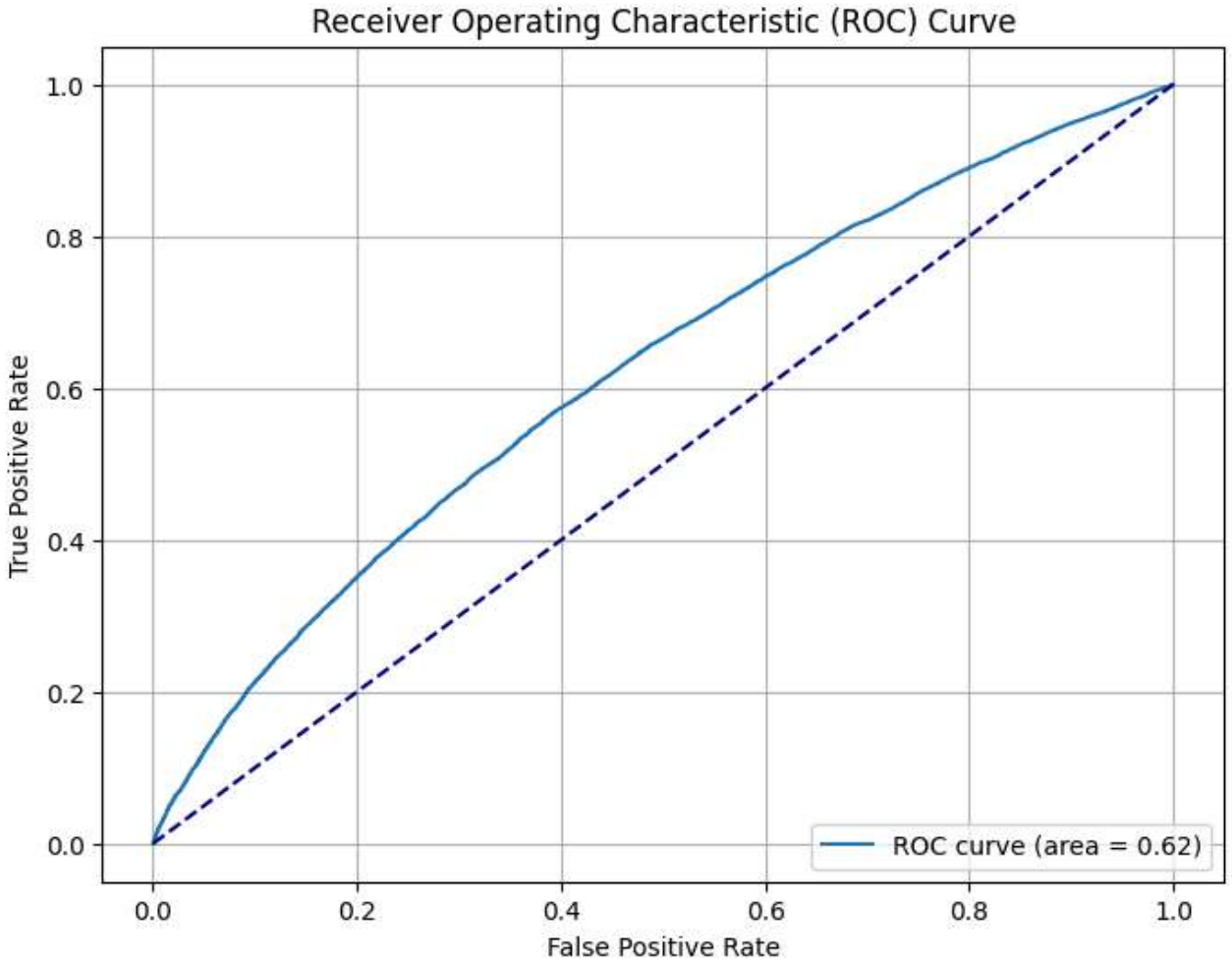
```
# Splitting the dataset
X = df.drop('loan_status', axis=1)
y = df['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Train Logistic Regression model with balanced class weights
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

# Extract the coefficients
coefficients = model.coef_[0]  # Get the coefficients for the features
feature_names = X.columns  # Get the feature names

# Create a dataframe to display coefficients with their corresponding feature names
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

# Calculate the odds ratio
coef_df['Odds Ratio'] = np.exp(coef_df['Coefficient'])

# Sort by the absolute value of the coefficient to see the most important features
coef_df['abs_Coefficient'] = coef_df['Coefficient'].abs()
coef_df = coef_df.sort_values(by='abs_Coefficient', ascending=False)

# Display the coefficients and odds ratio
print(coef_df[['Feature', 'Coefficient', 'Odds Ratio']])
```

```
                Feature  Coefficient  Odds Ratio
8             open_acc    -0.036623    0.964039
18       issue_d_month     0.031694    1.032202
7                  dti    -0.028576    0.971829
4   verification_status    -0.021148    0.979074
17             address     0.019812    1.020009
20           ecr_month     0.019078    1.019261
15            mort_acc     0.018009    1.018172
12           total_acc     0.013421    1.013511
0            emp_title     0.012124    1.012198
1           emp_length     0.011610    1.011678
11           revol_util    -0.009772    0.990275
6                 title     0.004404    1.004414
19         issue_d_year     0.003348    1.003353
9               pub_rec    -0.003090    0.996914
14     application_type     0.002148    1.002150
2        home_ownership     0.002033    1.002035
5                purpose     0.002015    1.002017
21             ecr_year     0.001912    1.001914
22                state     0.001742    1.001744
13  initial_list_status    -0.001485    0.998516
16 pub_rec_bankruptcies    -0.000651    0.999349
10           revol_bal     0.000019    1.000019
3            annual_inc     0.000008    1.000008
```

In [129…

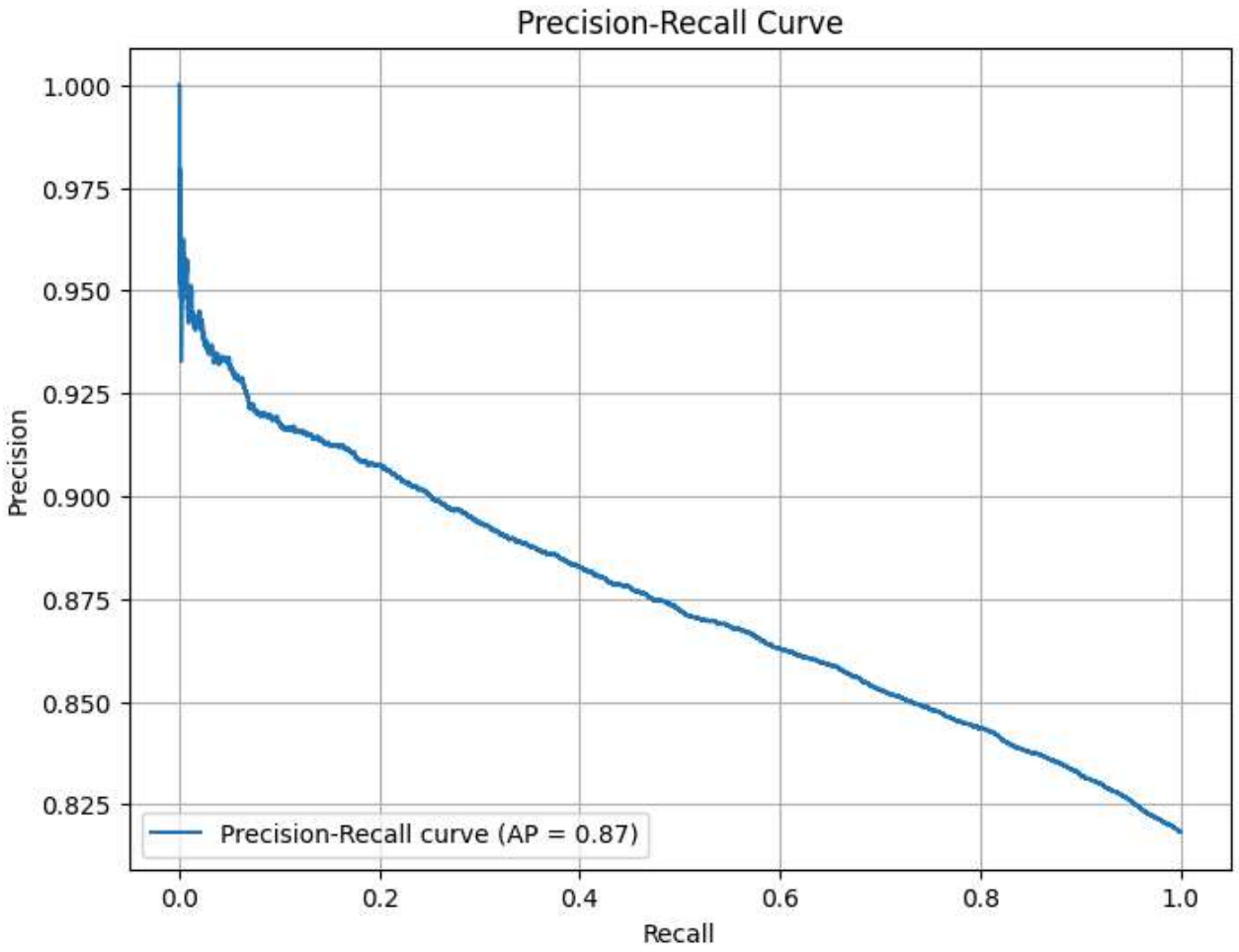
```
from sklearn.metrics import roc_curve, roc_auc_score

# Assuming model is already trained and X_test, y_test are available
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Get predicted probabilities

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Plot the ROC curve
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc_score(y_test, y_pred_proba):.2f})
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')  # Line for random model
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



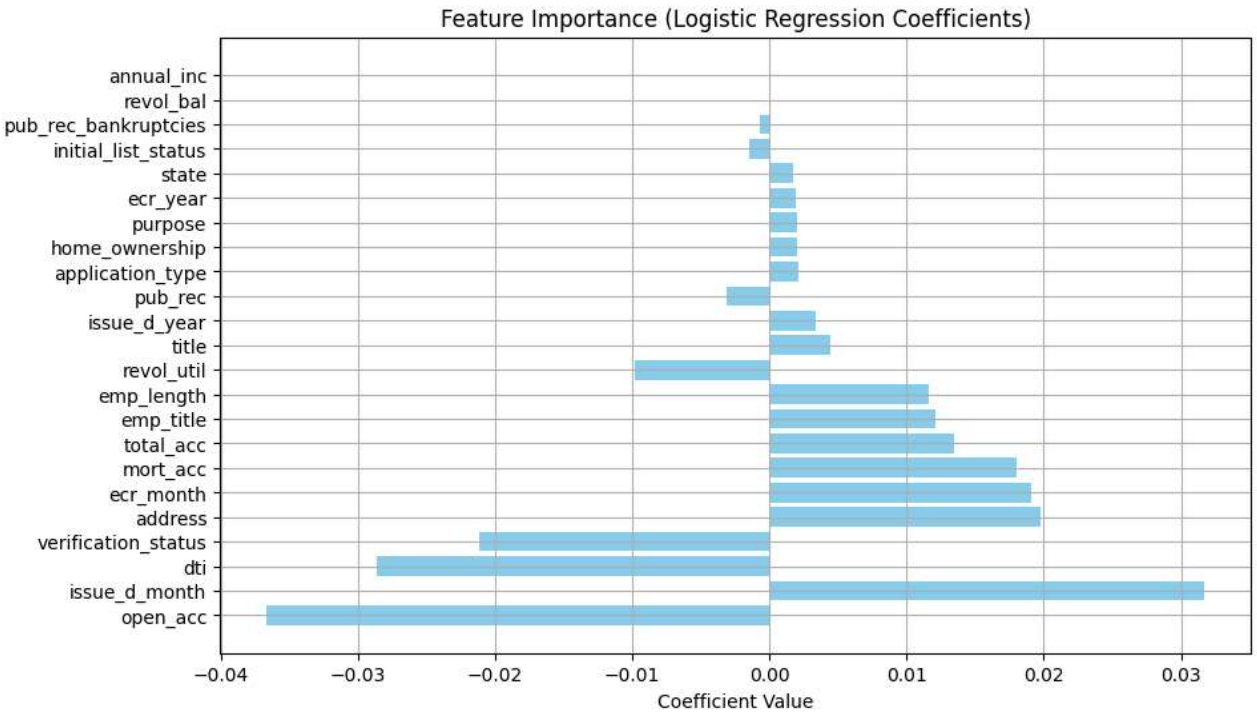
In [130…

```
# Calculate precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

# Plot the precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label=f'Precision-Recall curve (AP = {average_precision_sco
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.grid(True)
plt.show()
```



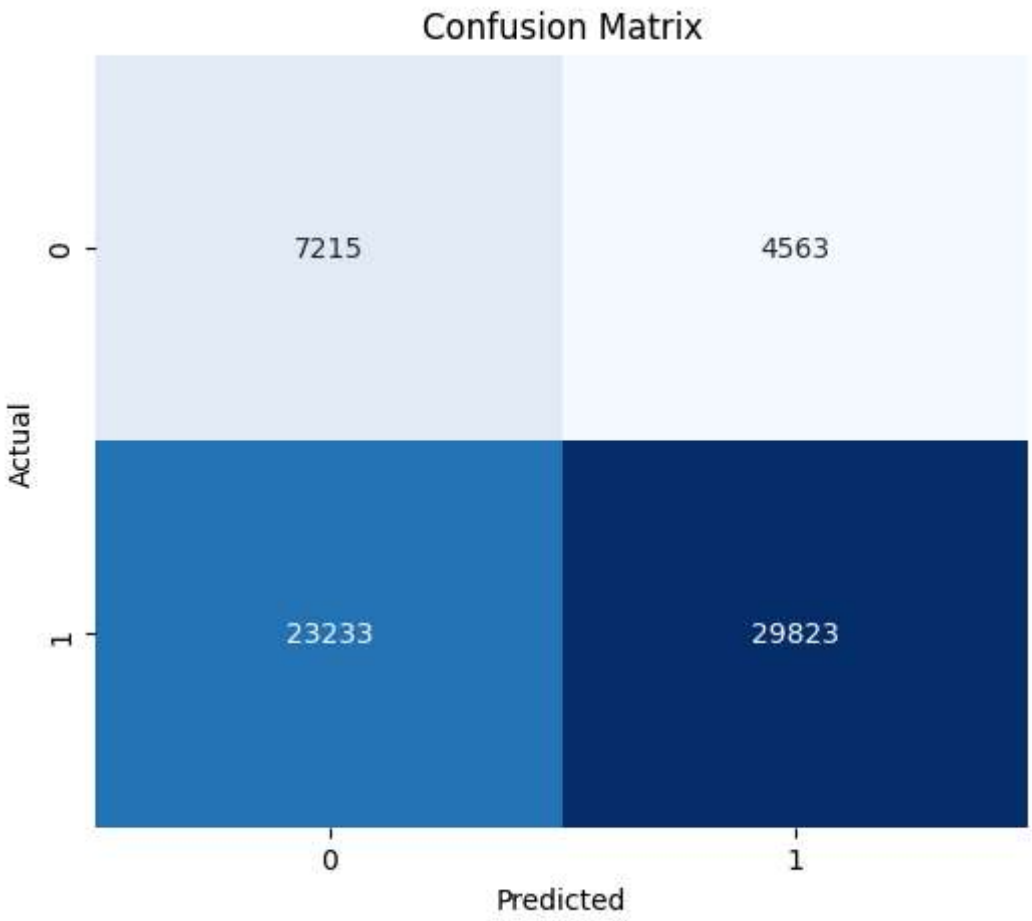
In [131…

```
# Sort the coefficients by their absolute values
coef_df = coef_df.sort_values(by='abs_Coefficient', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(coef_df['Feature'], coef_df['Coefficient'], color='skyblue')
plt.xlabel('Coefficient Value')
plt.title('Feature Importance (Logistic Regression Coefficients)')
plt.grid(True)
plt.show()
```



In [132…

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Create confusion matrix
cm = confusion_matrix(y_test, model.predict(X_test))

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



In [133…

```
from sklearn.metrics import confusion_matrix

# Predicted probabilities
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Adjust threshold (e.g., 0.3 instead of 0.5)
threshold = 0.3
y_pred_adjusted = (y_pred_proba >= threshold).astype(int)

# Confusion matrix for adjusted threshold
cm_adjusted = confusion_matrix(y_test, y_pred_adjusted)

print(cm_adjusted)
```

```
[[  360 11418]
 [  844 52212]]
```

In [134…

```
model = LogisticRegression(class_weight={0: 1, 1: 2})  # 2x more importance on false po
model.fit(X_train, y_train)
```

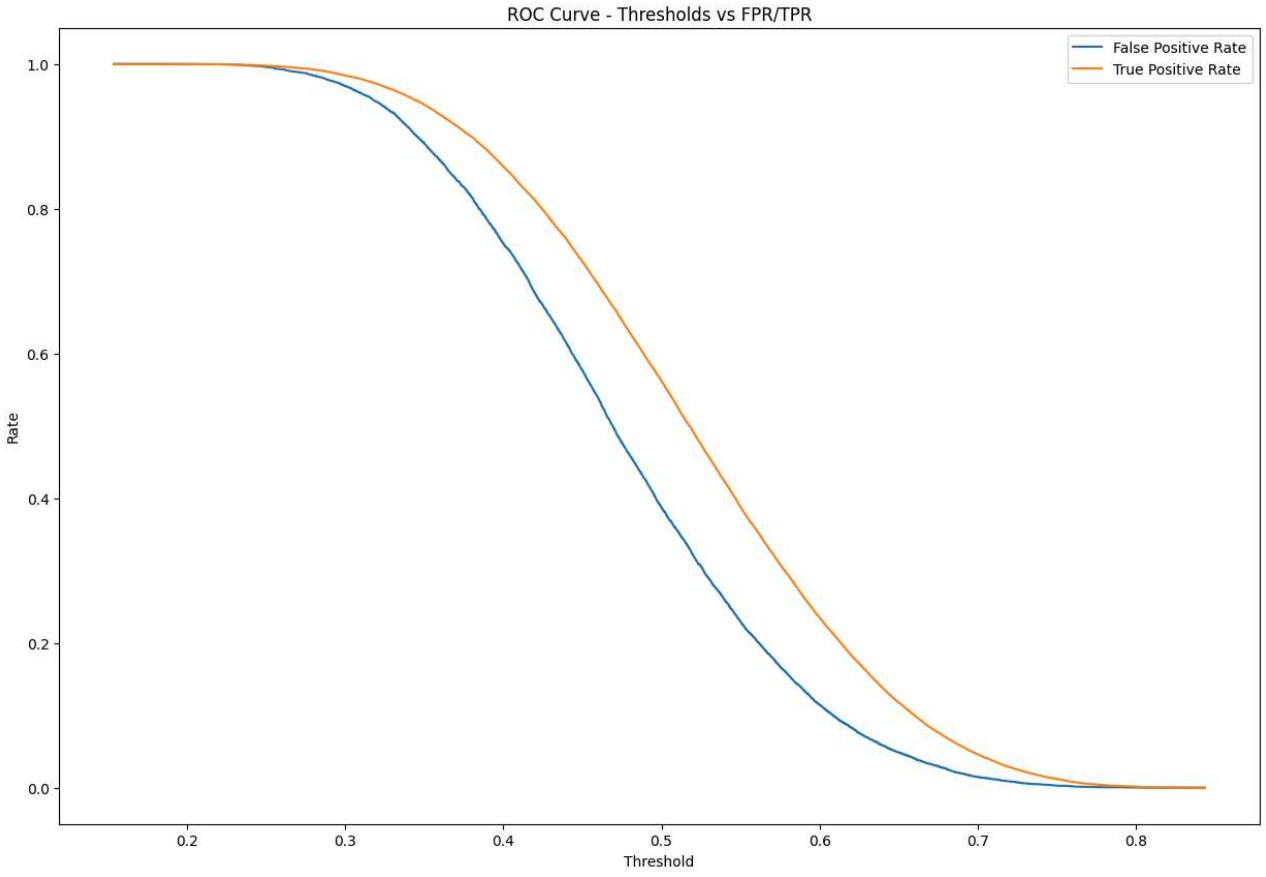
Out[134…

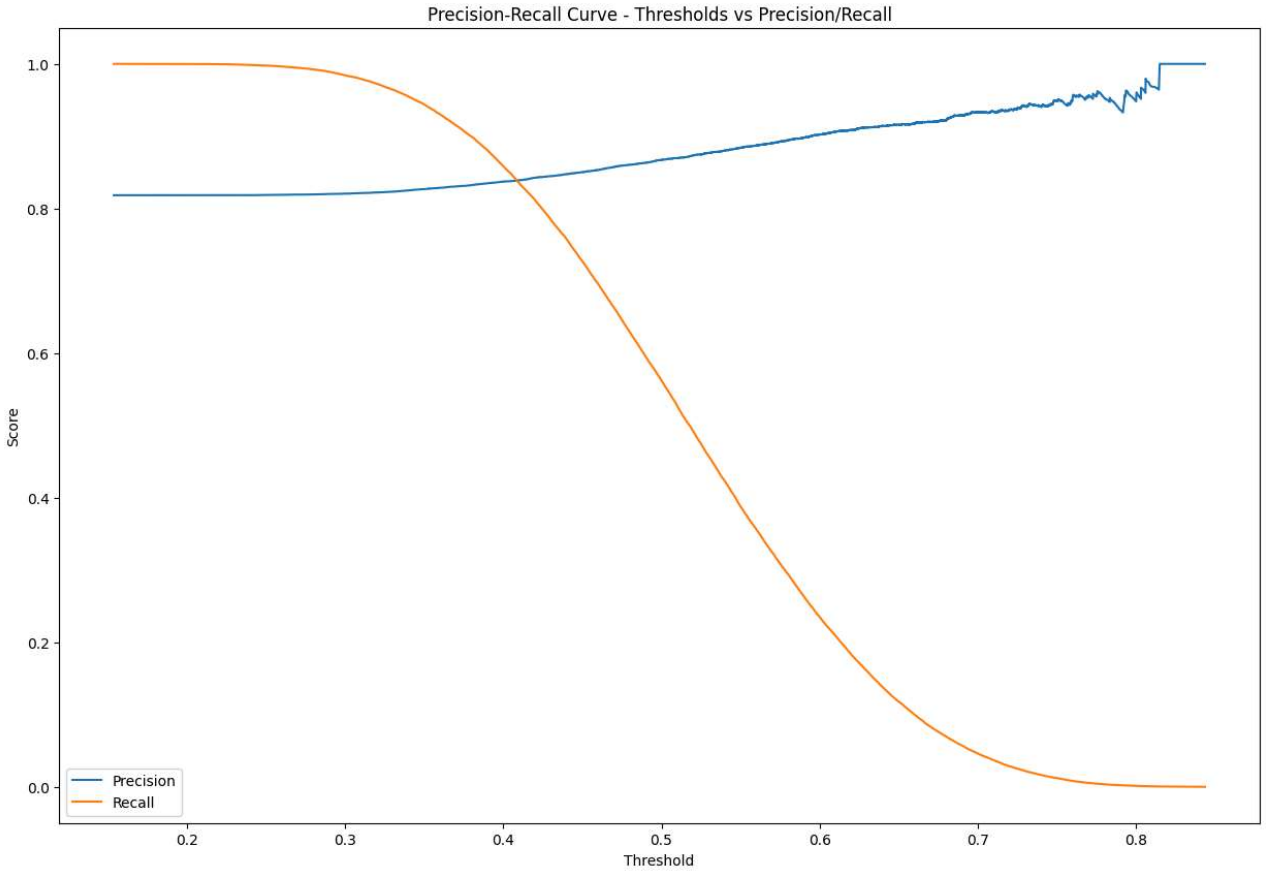
```
LogisticRegression(class_weight={0: 1, 1: 2})
```

In [135…

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import roc_curve, precision_recall_curve

# ROC curve
fpr, tpr, roc_thresholds = roc_curve(y_test, y_pred_proba)
plt.plot(roc_thresholds, fpr, label='False Positive Rate')
plt.plot(roc_thresholds, tpr, label='True Positive Rate')
plt.xlabel('Threshold')
plt.ylabel('Rate')
plt.legend()
plt.title('ROC Curve - Thresholds vs FPR/TPR')
plt.show()

# Precision-Recall Curve
precision, recall, pr_thresholds = precision_recall_curve(y_test, y_pred_proba)
plt.plot(pr_thresholds, precision[:-1], label='Precision')
plt.plot(pr_thresholds, recall[:-1], label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision-Recall Curve - Thresholds vs Precision/Recall')
plt.show()
```





Questionnaire (Answers should present in the text editor along with insights):

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

   =>80.36 %

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

   => Both are highly correlated, with correlation value of 0.96

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

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.. => Recall and F1 Score

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

   => The gap between precision and recall highlights the bank's approach to managing risk versus seizing growth opportunities. A bank should decide based on its strategic objectives—whether it prioritizes safety or expansion.

8. Which were the features that heavily affected the outcome?

   => Open_Account, dti, verification status, issue_d_month

9. Will the results be affected by geographical location?

   => No

In [135…