LENDING CLUB LOAN DATA

Lending Club is a peer-to-peer Lending company based in the US. They match people looking to invest money with people looking to borrow money. When investors invest their money through Lending Club, this money is passed onto borrowers, and when borrowers pay their loans back, the capital plus the interest passes on back to the investors. It is a win for everybody as they can get typically lower loan rates and higher investor returns.

The Lending Club dataset contains complete loan data for all loans issued through the 2007-2015, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information. Features (variables) include credit scores, number of finance inquiries, address including zip codes and state, and collections among others. Collections indicates whether the customer has missed one or more payments and the team is trying to recover their money.

TABLE OF CONTENT

```
# 1. Importing Libraries


import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.figure_factory as ff
import warnings
from pandas.errors import SettingWithCopyWarning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



import plotly.io as pio
pio.renderers.default = 'colab'



# 2. Loading Dataset from Drive


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

```
Mounted at /content/drive
```

```
# Replace the path with the actual path of your file in Google Drive
file_path = '/content/drive/MyDrive/Colab Notebooks/Lending Club Loan Data.csv'

Lending_Club_Loan_Data = pd.read_csv(file_path,low_memory = False)
Lending_Club_Loan_Data.head()
```

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	install
0	NaN	NaN	2500	2500	2500.0	36 months	13.56	
1	NaN	NaN	30000	30000	30000.0	60 months	18.94	7
2	NaN	NaN	5000	5000	5000.0	36 months	17.97	1
3	NaN	NaN	4000	4000	4000.0	36 months	18.94	1
4	NaN	NaN	30000	30000	30000.0	60 months	16.14	7

5 rows × 145 columns

```
Lending_Club_Loan_Data.shape
```

```
(2260668, 145)
```

```
# Changing the data target variable 'Loan status' to 'Rating' by filtering the loan statu

Lending_Club_Loan_Data = Lending_Club_Loan_Data[Lending_Club_Loan_Data.loan_status != 'Fu
Lending_Club_Loan_Data = Lending_Club_Loan_Data[Lending_Club_Loan_Data.loan_status != 'Do


Lending_Club_Loan_Data['rating'] = np.where((Lending_Club_Loan_Data.loan_status != 'Curre
pd.value_counts(Lending_Club_Loan_Data.rating).to_frame()
```

rating	count
1	1340973
0	919695

There are 9 unique loan statuses.

- Loans that are in "Default" are loans for which borrowers have failed to make payments for an extended period of time. In general, a Note enters Default status when it is 121+ days past due.
- A loan becomes “Charged Off” when there is no longer a reasonable expectation of further payments. Charge Off typically occurs when a loan is 150 days past due (i.e. 30 days after the Default status is reached) and there is no reasonable expectation of sufficient payment to prevent the charge off. In certain circumstances, loans may be charged off at an earlier or later date. Please note, loans for which borrowers have filed for bankruptcy may be charged off earlier based on the date of bankruptcy notification.

- A loan that is in “Default” will still appear in your Notes, in the status of “Default,” while a loan that has been “Charged Off” will appear as charged off, and the remaining principal balance of the Note will be deducted from your account balance

> We will drop ones that are fully paid as these are historical entries. Next step will assign as 0 (good) to Current loans and 1 (bad) to rest including: default and late loans, ones that were charged off or are in grace period.

```
Lending_Club_Loan_Data.shape
```

```
(1216728, 146)
```

```
# Data Information
Lending_Club_Loan_Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1216728 entries, 0 to 2260667
Columns: 146 entries, id to rating
dtypes: float64(105), int64(5), object(36)
memory usage: 1.3+ GB
```

DATA INFORMATION

- loan_amnt: The amount of money requested by the borrower.
- funded_amnt: The amount of money actually funded by investors.
- funded_amnt_inv: The portion of the loan funded by investors.
- term: The loan repayment period in months.
- int_rate: The interest rate on the loan.
- installment: Monthly payment owed by the borrower.
- grade: Lending Club-assigned loan grade.
- sub_grade: Lending Club-assigned sub-grade of the loan.
- emp_title: The job title of the borrower.
- emp_length: The number of years the borrower has been employed.
- home_ownership: The borrower's homeownership status.
- annual_inc: The annual income of the borrower.
- verification_status: Indicates whether the borrower's income was verified.
- issue_d: The month the loan was issued.
- loan_status: The current status of the loan (e.g., Fully Paid, Charged Off).

- pymnt_plan: Whether the borrower is on a payment plan.
- purpose: The stated purpose of the loan.
- title: The loan title provided by the borrower.
- zip_code: The first three digits of the borrower's postal code.
- addr_state: The state where the borrower resides.
- dti: Debt-to-income ratio of the borrower.
- delinq_2yrs: Number of delinquencies in the past two years.
- earliest_cr_line: The date of the borrower's earliest credit line.
- inq_last_6mths: Number of credit inquiries in the last six months.
- mths_since_last_delinq: Months since the last delinquency.
- open_acc: Number of open credit accounts.
- pub_rec: Number of derogatory public records.
- revol_bal: Total credit revolving balance.
- revol_util: Revolving credit utilization rate.
- total_acc: Total number of credit accounts.
- initial_list_status: The initial listing status of the loan.
- out_prncp: Remaining outstanding principal amount for the loan.
- out_prncp_inv: Remaining outstanding principal funded by investors.
- total_pymnt: Total payments received to date.
- total_pymnt_inv: Total payments received to date from investors.
- total_rec_prncp: Total principal received to date.
- total_rec_int: Total interest received to date.
- total_rec_late_fee: Total late fees received to date.
- recoveries: Amount recovered after charge-off.
- collection_recovery_fee: Recovery fee from collections.
- last_pymnt_d: The date of the last payment received.
- last_pymnt_amnt: The amount of the last payment received.
- next_pymnt_d: The due date for the next payment.
- last_credit_pull_d: The date of the last credit report pull.
- collections_12_mths_ex_med: Number of collections in the last 12 months excluding medical.

- policy_code: Code indicating the policy applied to the loan.
- application_type: Indicates whether the application is individual or joint.
- acc_now_delinq: Number of accounts currently delinquent.
- tot_coll_amt: Total collection amount ever owed.
- tot_cur_bal: Total current balance of all accounts.
- open_acc_6m: Number of open trades in the last 6 months.
- open_act_il: Number of open installment accounts.
- open_il_12m: Number of installment accounts opened in the last 12 months.
- open_il_24m: Number of installment accounts opened in the last 24 months.
- mths_since_rcnt_il: Months since the most recent installment account opened.
- total_bal_il: Total current balance of all installment accounts.
- il_util: Ratio of total current installment balance to high credit limit.
- open_rv_12m: Number of revolving trades opened in the last 12 months.
- open_rv_24m: Number of revolving trades opened in the last 24 months.
- max_bal_bc: Maximum current balance on a bankcard.
- all_util: Balance to credit limit across all accounts.
- total_rev_hi_lim: Total revolving credit high limit.
- inq_fi: Number of personal finance inquiries.
- total_cu_tl: Number of finance trades.
- inq_last_12m: Number of inquiries in the last 12 months.
- acc_open_past_24mths: Number of accounts opened in the past 24 months.
- avg_cur_bal: Average current balance of all accounts.
- bc_open_to_buy: Total credit limit available on revolving bankcards.
- bc_util: Utilization of total bankcard credit.
- chargeoff_within_12_mths: Number of charge-offs within the last 12 months.
- delinq_amnt: Amount delinquent.
- mo_sin_old_il_acct: Months since the oldest installment account opened.
- mo_sin_old_rev_tl_op: Months since the oldest revolving account opened.
- mo_sin_rcnt_rev_tl_op: Months since the most recent revolving account opened.
- mo_sin_rcnt_tl: Months since the most recent trade line opened.
- mort_acc: Number of mortgage accounts.

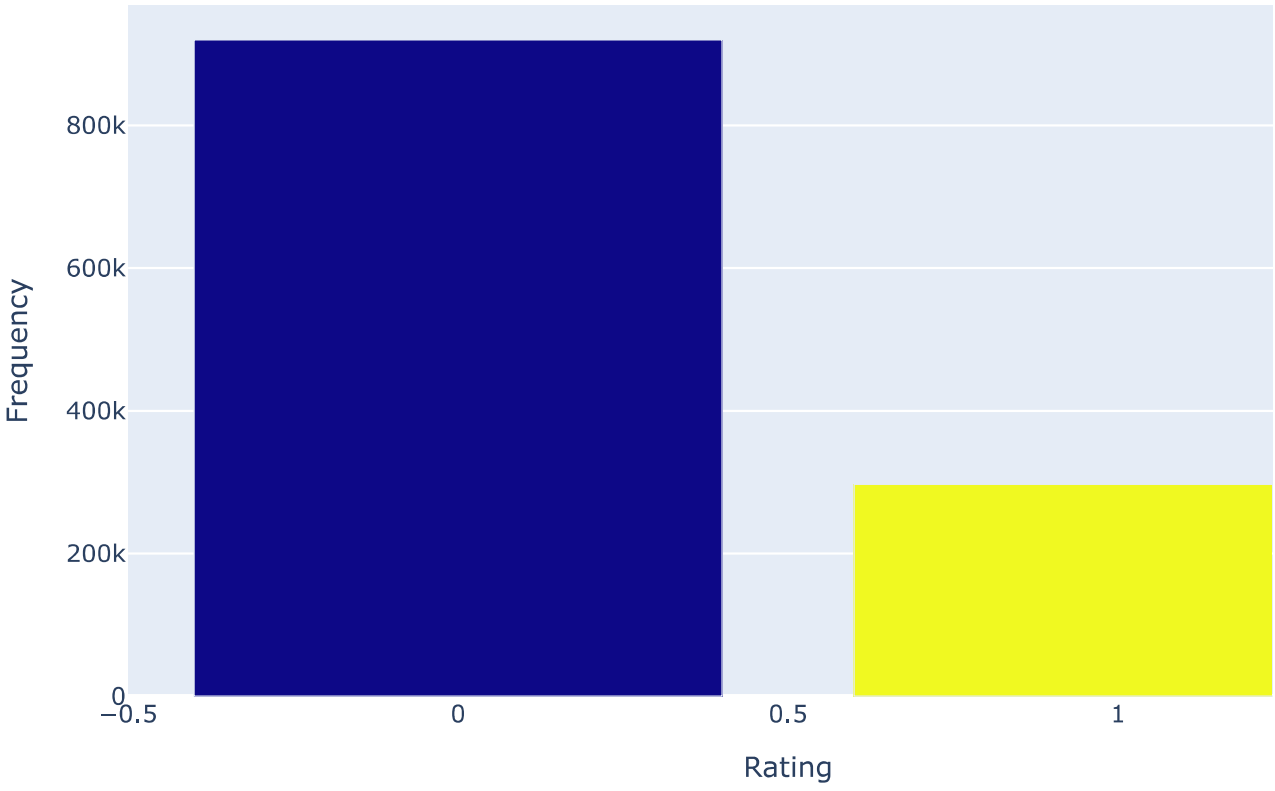
- mths_since_recent_bc: Months since the most recent bankcard opened.
- mths_since_recent_inq: Months since the most recent inquiry.
- mths_since_recent_revol_delinq: Months since the most recent revolving delinquency.
- num_accts_ever_120_pd: Number of accounts ever 120+ days past due.
- num_actv_bc_tl: Number of currently active bankcard trade lines.
- num_actv_rev_tl: Number of currently active revolving trade lines.
- num_bc_sats: Number of satisfactory bankcard accounts.
- num_bc_tl: Number of bankcard trade lines.
- num_il_tl: Number of installment trade lines.
- num_op_rev_tl: Number of open revolving trade lines.
- num_rev_accts: Number of revolving accounts.
- num_rev_tl_bal_gt_0: Number of revolving trade lines with a balance greater than zero.
- num_sats: Number of satisfactory accounts.
- num_tl_120dpd_2m: Number of accounts currently 120 days past due (in the past 2 months).
- num_tl_30dpd: Number of accounts currently 30 days past due.
- num_tl_90g_dpd_24m: Number of accounts 90+ days past due in the past 24 months.
- num_tl_op_past_12m: Number of trade lines opened in the past 12 months.
- pct_tl_nvr_dlq: Percentage of trade lines that were never delinquent.
- percent_bc_gt_75: Percentage of bankcards with utilization greater than 75%.
- pub_rec_bankruptcies: Number of public record bankruptcies.
- tax_liens: Number of tax liens.
- tot_hi_cred_lim: Total high credit/credit limit.
- total_bal_ex_mort: Total balance excluding mortgage accounts.
- total_bc_limit: Total credit limit on all bankcards.
- total_il_high_credit_limit: Total high credit limit on all installment accounts.
- hardship_flag: Indicates if the borrower is in a hardship program.
- disbursement_method: The method of loan disbursement.
- debt_settlement_flag: Indicates if the borrower is in debt settlement.
- rating: Custom rating assigned to the loan.

```
#Describing
Lending_Club_Loan_Data.describe()
```

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	ins
count	0.0	0.0	1.216728e+06	1.216728e+06	1.216728e+06	1.216728e+06	1.21
mean	NaN	NaN	1.584013e+04	1.583817e+04	1.582691e+04	1.347867e+01	4.58
std	NaN	NaN	9.558234e+03	9.557599e+03	9.559130e+03	5.054136e+00	2.72
min	NaN	NaN	5.000000e+02	5.000000e+02	0.000000e+00	5.310000e+00	7.61
25%	NaN	NaN	8.500000e+03	8.500000e+03	8.475000e+03	9.930000e+00	2.61
50%	NaN	NaN	1.400000e+04	1.400000e+04	1.400000e+04	1.279000e+01	3.88
75%	NaN	NaN	2.100000e+04	2.100000e+04	2.100000e+04	1.620000e+01	6.09
max	NaN	NaN	4.000000e+04	4.000000e+04	4.000000e+04	3.099000e+01	1.71

8 rows × 110 columns

```
# Distribution of a Target variablle "rating"


# Group 'rating' into bins and count occurrences
rating_counts = Lending_Club_Loan_Data['rating'].value_counts().reset_index()
rating_counts.columns = ['rating', 'count']

# Create a bar chart
fig = px.bar(
    rating_counts,
    x='rating',
    y='count',
    title="Distribution of Rating",
    labels={'rating': 'Rating', 'count': 'Frequency'},
    color='rating'
)

fig.show()
```



3. Data Analysis

```
Lending_Club_Loan_Data.columns
```

```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       ...
       'hardship_last_payment_amount', 'disbursement_method',
       'debt_settlement_flag', 'debt_settlement_flag_date',
       'settlement_status', 'settlement_date', 'settlement_amount',
       'settlement_percentage', 'settlement_term', 'rating'],
      dtype='object', length=146)
```

3.1 Removing Columns which are more than 50% Data is null values for better Analysis

```
# we remove columns with more that 50% missing values
check_null = Lending_Club_Loan_Data.isnull().sum(axis=0).sort_values(ascending=False)/flc
print(check_null[check_null>0.5].head())
```

```
member_id                                   1.000000
id                                          1.000000
url                                         1.000000
orig_projected_additional_accrued_interest  0.994249
hardship_start_date                         0.992525
```

```
dtype: float64
```

```
# Removing the column 'Loan Status' so, the 'Rating' Column be the Target variable in thi
Lending_Club_Loan_Data = Lending_Club_Loan_Data.drop('loan_status',axis = 1)
```

```
# Printing total numbers and percentage of missing data
Total = Lending_Club_Loan_Data.isnull().sum().sort_values(ascending=False)
Percent = (Lending_Club_Loan_Data.isnull().sum()/Lending_Club_Loan_Data.isnull().count())
Missing_data = pd.concat([Total, Percent], axis=1, keys=['Total', 'Percent'])
Missing_data.head()
```

	Total	Percent
member_id	1216728	1.000000
id	1216728	1.000000
url	1216728	1.000000
orig_projected_additional_accrued_interest	1209731	0.994249
hardship_dpd	1207633	0.992525

```
# Remove columns with more than 50% missing values

# Define the threshold (50%)
threshold = 0.5
Lending_Club_Loan = Lending_Club_Loan_Data.loc[:, Lending_Club_Loan_Data.isnull().mean()
Lending_Club_Loan.columns
```

```
Index(['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
       'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       ...
       'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
       'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_credit_limit',
       'hardship_flag', 'disbursement_method', 'debt_settlement_flag',
       'rating'],
      dtype='object', length=102)
```

```
Lending_Club_Loan.shape
```

```
(1216728, 102)
```

```
Lending_Club_Loan.dtypes.head()
```

	0
loan_amnt	int64
funded_amnt	int64
funded_amnt_inv	float64
term	int64
int_rate	float64

dtype: object

```
Lending_Club_Loan.columns
```

```
Index(['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
       'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       ...
       'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
       'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_credit_limit',
       'hardship_flag', 'disbursement_method', 'debt_settlement_flag',
       'rating'],
      dtype='object', length=102)
```

```
# Checking the Null Values for New DataFrame
Lending_Club_Loan.isnull().sum().head()
```

	0
loan_amnt	0
funded_amnt	0
funded_amnt_inv	0
term	0
int_rate	0

dtype: int64

```
# 3.2 Converting Some Categorical features to Numerical features


# Categorical Column - Term

# Use .loc to ensure you're modifying the original DataFrame
Lending_Club_Loan.loc[:, 'term'] = Lending_Club_Loan['term'].replace({' months': ''}, reg
Lending_Club_Loan.loc[:, 'term'] = pd.to_numeric(Lending_Club_Loan['term'], errors='coerc
Lending_Club_Loan['term'] = Lending_Club_Loan['term'].astype(int)
# Check the result
print("Term after replacement:")
print(Lending_Club_Loan['term'].head())
```

```
Term after replacement:
0    36
1    60
2    36
3    36
4    60
Name: term, dtype: int64
```

```
# Categorical Column - Payment Plan

int_status={'y':1,'n':0}
Lending_Club_Loan.pymnt_plan.replace(int_status,inplace=True)
Lending_Club_Loan['pymnt_plan'] = Lending_Club_Loan['pymnt_plan'].astype(int)
print(" payment plan after:")
print(Lending_Club_Loan.pymnt_plan.head())
```

```
 payment plan after:
0    0
1    0
2    0
3    0
4    0
Name: pymnt_plan, dtype: int64
```

```
# Categorical Column - initial_list_status

int_status={'w':1,'f':0}
Lending_Club_Loan.initial_list_status.replace(int_status,inplace=True)
Lending_Club_Loan['initial_list_status'] = Lending_Club_Loan['initial_list_status'].astyp
print("Status of the initial_list_status after:")
print(Lending_Club_Loan.initial_list_status.head())
```

```
Status of the initial_list_status after:
0    1
1    1
2    1
3    1
4    1
Name: initial_list_status, dtype: int64
```

```
# Categorical Column - hardship_flag

int_status={'N':1,'Y':0}
Lending_Club_Loan.hardship_flag.replace(int_status,inplace=True)
Lending_Club_Loan['hardship_flag'] = Lending_Club_Loan['hardship_flag'].astype(int)
print("hardship_flag after:")
print(Lending_Club_Loan.hardship_flag.head())
```

```
hardship_flag after:
0    1
1    1
2    1
3    1
4    1
Name: hardship_flag, dtype: int64
```

```
# Categorical Column - debt_settlement_flag

int_status={'N':1,'Y':0}
Lending_Club_Loan.debt_settlement_flag.replace(int_status,inplace=True)
Lending_Club_Loan['debt_settlement_flag'] = Lending_Club_Loan['debt_settlement_flag'].ast
print("debt_settlement_flag after:")
print(Lending_Club_Loan.debt_settlement_flag.head())
```

```
debt_settlement_flag after:
0    1
1    1
2    1
3    1
4    1
Name: debt_settlement_flag, dtype: int64
```

```
# 3.3 Taking the specific columns to  perform operations on them from the original datase


# List of columns to group
selected_columns = [
    'loan_amnt', 'funded_amnt','funded_amnt_inv','term', 'int_rate', 'installment',
    'pymnt_plan','purpose','addr_state','out_prncp','out_prncp_inv','total_pymnt','total_
    'total_rec_prncp','total_rec_int','total_rec_late_fee','grade','sub_grade','home_owne
    'verification_status','revol_bal','initial_list_status','recoveries','collection_recc
    'last_pymnt_amnt','application_type','hardship_flag','disbursement_method','debt_sett

# Create a new DataFrame with the selected columns
Lending_Club_Loan_imp = Lending_Club_Loan_Data[selected_columns]



Lending_Club_Loan_imp.shape
```

```
(1216728, 29)
```

```
# 3.4 Checking the Value counts for each columns in Lending club loan Dataset for Rating


# Iterate through each column
for column in Lending_Club_Loan_imp.columns:
    # Get unique values and their counts for the column
    unique_values = Lending_Club_Loan_imp[column].value_counts()

    print(f"\nUnique values for column '{column}':")
    print("-" * 100)
    print(unique_values.head())
```

```

Unique values for column 'loan_amnt':
----------------------------------------------------------------------------------------------------
loan_amnt
10000    108929
20000     77214
```

```
15000     68475
12000     64102
35000     48840
Name: count, dtype: int64

Unique values for column 'funded_amnt':
-----------------------------------------------------------------------------------
funded_amnt
10000    108918
20000     77187
15000     68457
12000     64093
35000     48812
Name: count, dtype: int64

Unique values for column 'funded_amnt_inv':
-----------------------------------------------------------------------------------
funded_amnt_inv
10000.0    104815
20000.0     72872
15000.0     65131
12000.0     61279
5000.0      44359
Name: count, dtype: int64

Unique values for column 'term':
-----------------------------------------------------------------------------------
term
36 months    778360
60 months    438368
Name: count, dtype: int64

Unique values for column 'int_rate':
-----------------------------------------------------------------------------------
int_rate
11.99    27771
13.99    21678
5.32     21404
12.62    18210
16.02    18118
Name: count, dtype: int64

Unique values for column 'installment':
-----------------------------------------------------------------------------------
installment
332.10    2047
301.15    2029
361.38    1944
309.74    1587
320.05    1547
Name: count, dtype: int64
```

```
# 3.5 Checking the Minimum and Maximum for each columns in Lending club loan Dataset for


# Iterate through each column
for column in Lending_Club_Loan_imp.columns:
    # Check if the column is numeric
    if pd.api.types.is_numeric_dtype(Lending_Club_Loan_imp[column]):
```

```
    min_value = Lending_Club_Loan_imp[column].min()
    max_value = Lending_Club_Loan_imp[column].max()
    print(f"\nMinimum and Maximum values for column '{column}':")
    print("-" * 100)
    print(f"Min: {min_value}")
    print(f"Max: {max_value}")
```

```
Minimum and Maximum values for column 'loan_amnt':
----------------------------------------------------------------------------------------------------
Min: 500
Max: 40000

Minimum and Maximum values for column 'funded_amnt':
----------------------------------------------------------------------------------------------------
Min: 500
Max: 40000

Minimum and Maximum values for column 'funded_amnt_inv':
----------------------------------------------------------------------------------------------------
Min: 0.0
Max: 40000.0

Minimum and Maximum values for column 'int_rate':
----------------------------------------------------------------------------------------------------
Min: 5.31
Max: 30.99

Minimum and Maximum values for column 'installment':
----------------------------------------------------------------------------------------------------
Min: 7.61
Max: 1719.83

Minimum and Maximum values for column 'out_prncp':
----------------------------------------------------------------------------------------------------
Min: 0.0
Max: 40000.0

Minimum and Maximum values for column 'out_prncp_inv':
----------------------------------------------------------------------------------------------------
Min: 0.0
Max: 40000.0

Minimum and Maximum values for column 'total_pymnt':
----------------------------------------------------------------------------------------------------
Min: 0.0
Max: 61947.5297968037

Minimum and Maximum values for column 'total_pymnt_inv':
----------------------------------------------------------------------------------------------------
Min: 0.0
Max: 61947.53

Minimum and Maximum values for column 'total_rec_prncp':
----------------------------------------------------------------------------------------------------
Min: 0.0
Max: 40000.0

Minimum and Maximum values for column 'total_rec_int':
```

```
--------------------------------------------------------------------------------
Min: 0.0
Max: 27814.3

Minimum and Maximum values for column 'total_rec_late_fee':
```

```
# 3.6 Splitting the Data into Numerical features and Categorical features
```

```
#  Numerical features¶
```

```
Numerical_features = Lending_Club_Loan.select_dtypes(include = ['integer','float']).colum
print("No of Numerical Features:",len(Numerical_features))
Lending_Club_Loan[Numerical_features].head()
```

```
No of Numerical Features: 85
```

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	annual_inc
0	2500	2500	2500.0	36	13.56	84.92	55000.0
1	30000	30000	30000.0	60	18.94	777.23	90000.0
2	5000	5000	5000.0	36	17.97	180.69	59280.0
3	4000	4000	4000.0	36	18.94	146.51	92000.0
4	30000	30000	30000.0	60	16.14	731.78	57250.0

5 rows × 85 columns

```
Lending_Club_Loan[Numerical_features].columns.tolist()
```

```
['loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'installment',
 'annual_inc',
 'pymnt_plan',
 'dti',
 'delinq_2yrs',
 'inq_last_6mths',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'initial_list_status',
 'out_prncp',
 'out_prncp_inv',
 'total_pymnt',
 'total_pymnt_inv',
 'total_rec_prncp',
 'total_rec_int',
 'total_rec_late_fee',
```

```
        'recoveries',
        'collection_recovery_fee',
        'last_pymnt_amnt',
        'collections_12_mths_ex_med',
        'policy_code',
        'acc_now_delinq',
        'tot_coll_amt',
        'tot_cur_bal',
        'open_acc_6m',
        'open_act_il',
        'open_il_12m',
        'open_il_24m',
        'mths_since_rcnt_il',
        'total_bal_il',
        'il_util',
        'open_rv_12m',
        'open_rv_24m',
        'max_bal_bc',
        'all_util',
        'total_rev_hi_lim',
        'inq_fi',
        'total_cu_tl',
        'inq_last_12m',
        'acc_open_past_24mths',
        'avg_cur_bal',
        'bc_open_to_buy',
        'bc_util',
        'chargeoff_within_12_mths',
        'delinq_amnt',
        'mo_sin_old_il_acct',
        'mo_sin_old_rev_tl_op',
        'mo_sin_rcnt_rev_tl_op',
        'mo_sin_rcnt_tl',
        'mort_acc'
```

```
# Categorical features


Categorical_features = Lending_Club_Loan.select_dtypes(include = ['object']).columns.toli
print("No of Categorical Features:",len(Categorical_features))
Lending_Club_Loan[Categorical_features].head()
```

```
No of Categorical Features: 17
```

	grade	sub_grade	emp_title	emp_length	home_ownership	verification_status	is
0	C	C1	Chef	10+ years	RENT	Not Verified	
1	D	D2	Postmaster	10+ years	MORTGAGE	Source Verified	
2	D	D1	Administrative	6 years	MORTGAGE	Source Verified	
3	D	D2	IT Supervisor	10+ years	MORTGAGE	Source Verified	
4	C	C4	Mechanic	10+ years	MORTGAGE	Not Verified	

```
Lending_Club_Loan[Categorical_features].columns.tolist()
```

```
['grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'purpose',
 'title',
 'zip_code',
 'addr_state',
 'earliest_cr_line',
 'last_pymnt_d',
 'next_pymnt_d',
 'last_credit_pull_d',
 'application_type',
 'disbursement_method']
```

```
# 3.7 Handling Missing values


# Separate numerical and categorical columns
Numerical_features = Lending_Club_Loan.select_dtypes(include=['float64', 'int64']).column
Categorical_features = Lending_Club_Loan.select_dtypes(include=['object']).columns

# Handling missing values for numerical features (e.g., filling with median)
for col in Numerical_features:
    median_value = Lending_Club_Loan[col].median()
    Lending_Club_Loan[col].fillna(median_value, inplace=True)

# Handling missing values for categorical features (e.g., filling with the mode)
for col in Categorical_features:
    mode_value = Lending_Club_Loan[col].mode()[0]  # mode() returns
    Lending_Club_Loan[col].fillna(mode_value, inplace=True)
```

```
print(Lending_Club_Loan.isnull().sum())
```

```
loan_amnt                     0
funded_amnt                   0
funded_amnt_inv               0
term                          0
int_rate                      0
                             ..
total_il_high_credit_limit    0
hardship_flag                 0
disbursement_method           0
debt_settlement_flag          0
rating                        0
Length: 102, dtype: int64
```

```
 # 3.8 Group Columns by Important features for Target variable 'Rating':
```

```
## Loan Information
```

```
Loan_Information = {
    "Loan Information Columns ": [
        'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment'
    ]}
print(f"{Loan_Information}:")
```

```
{'Loan Information Columns ': ['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term',
```

```
## Loan Rating and Payment Details:
```

```
Loan_Rating_and_Payment_Details = { "Loan Rating and Payment Columns": [
         'pymnt_plan', 'purpose', 'addr_state', 'out_prncp', 'out_prncp_inv',
        'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_
    ]}
print(f"{Loan_Rating_and_Payment_Details}:")
```

```
{'Loan Rating and Payment Columns': ['pymnt_plan', 'purpose', 'addr_state', 'out_prnc
```

```
## Borrower Information
```

```
Borrower_Information = { "Borrower Information Columns": [
        'grade', 'sub_grade', 'home_ownership', 'verification_status'
    ]}
print(f"{Borrower_Information}:")
```

```
{'Borrower Information Columns': ['grade', 'sub_grade', 'home_ownership', 'verificati
```

```
##  Loan and Credit Information:


Loan_and_Credit_Information = { "Loan and Credit Information Columns ": [
        'revol_bal', 'initial_list_status', 'recoveries', 'collection_recovery_fee', 'las
        'policy_code', 'application_type', 'hardship_flag', 'disbursement_method', 'debt_
    ]}
print(f"{Loan_and_Credit_Information}:")
```

```
{'Loan and Credit Information Columns ': ['revol_bal', 'initial_list_status', 'recove
```

```
# 4. Data Visualization


## 4.1 Loan Information Columns charts


# Create a box plot to show the distribution of loan amounts by Customer Loan Status Rati
fig = px.box(Lending_Club_Loan,
             x="rating",
             y="loan_amnt",
             title="Loan Amount Distribution by Customer Loan Status Rating",
             labels={"loan_amnt": "Loan Amount", "rating": "Customer Loan Status Rating"}
             color="rating")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

Description:

Visualizing the distribution of "Loan Amount" against "Customer Loan Status Rating," where 0 and 1 represent different customer ratings. The plot highlights that loan amounts for both ratings are centered around similar medians but show differences in their spread and outliers.

```
# Create a box plot to show the distribution of Funded Amount by Loan Amount
fig = px.box(Lending_Club_Loan,
             x="loan_amnt",
             y="funded_amnt",
             title="Funded Amount Investors Distribution by Loan Amount",
             labels={"funded_amnt": "Funded Amount by Investors", "loan_amnt": "Loan Amou
             color="loan_amnt")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

Description:

The distribution of "Funded Amount by Investors" against "Loan Amount," showing how funded amounts vary across different loan sizes. It likely highlights the central tendency, spread, and any potential outliers, indicating whether funded amounts are closely aligned or vary significantly with loan amounts.

```
# Create a box plot to show the distribution of Loan Funded Investors by Funded Amount by
fig = px.box(Lending_Club_Loan,
             x="funded_amnt",
             y="funded_amnt_inv",
             title="Loan Funded Investors Distribution by Funded Amount by Investors",
             labels={"funded_amnt_inv": "Loan Funded Investors", "funded_amnt": "Funded A
             color="funded_amnt")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

Description: The distribution of "Loan Funded Investors" based on the "Funded Amount by Investors," showing how investor contributions vary across different funded amounts. It likely reveals patterns such as the median contribution, the spread of values, and any outliers, indicating the relationship between funding levels and investor participation.

```
# Create a pie chart to show the distribution of Loan Interest Rate by Customer Loan Stat
fig = px.pie(
    Lending_Club_Loan,
    names="rating",
    values="int_rate",
    title="Loan Interest Rate Distribution by Customer Loan Status Rating",
    labels={"int_rate": "Loan Interest Rate", "rating": "Customer Loan Status Rating"},
    color_discrete_sequence=px.colors.sequential.RdBu  # Optional: Use a color palette
)

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of  loan repayment period  by Loan Inter
fig = px.scatter(Lending_Club_Loan,
             x="int_rate",
             y="term",
             title=" loan repayment period Distribution by Loan Interest Rate ",
             labels={"term": " loan repayment period ", "int_rate": "Loan Interest Rate "
             color="int_rate")  # Optional: Add color for better visualization

# Show the chart
```

```
fig.show()
```

```
# Create a scatter plot to show the distribution of borrower Monthly payment by  Borrower
fig = px.scatter(Lending_Club_Loan,
             x="pymnt_plan",
             y="installment",
             title="Borrower Monthly payment Distribution by Borrower payment plan",
             labels={"installment": "borrower Monthly payment", "pymnt_plan": "Borrower p
             color="pymnt_plan")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# 4.2 Loan Rating and Payment charts


# Create a scatter plot to show the distribution of borrower payment plan by Loan Issued
fig = px.scatter(Lending_Club_Loan,
             x="issue_d",
             y="pymnt_plan",
             title="borrower payment plan Distribution by Loan Issued Month",
             labels={"pymnt_plan": "borrower payment plan", "issue_d": "Loan Issued Month
             color="issue_d")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of  Amount delinquent Amount by  Number
fig = px.scatter(Lending_Club_Loan,
             x="delinq_2yrs",
             y="delinq_amnt",
             title=" Amount delinquent Distribution by  Number of delinquencies in the pa
             labels={"delinq_amnt": " Amount delinquent", "delinq_2yrs": " Number of deli
             color="delinq_2yrs")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a box plot to show the distribution of purpose of the loan by loan amounts
fig = px.scatter(Lending_Club_Loan,
             x="loan_amnt",
```

```
            y="purpose",
            title="purpose of the loan Distribution by loan amounts",
            labels={"purpose": "purpose of the loan", "loan_amnt": "loan amounts"},
            color="loan_amnt")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a box plot to show the distribution of borrower resides by  Borrower's homeowner
fig = px.box(Lending_Club_Loan,
            x="home_ownership",
            y="addr_state",
            title="borrower resides Distribution by  Borrower's homeownership status",
            labels={"addr_state": "borrower resides", "home_ownership": " Borrower's hom
            color="home_ownership")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of outstanding principal amount by Outst
fig = px.scatter(Lending_Club_Loan,
            x="out_prncp_inv",
            y="out_prncp",
            title="outstanding principal amountDistribution by outstanding principal fun
            labels={"out_prncp": "outstanding principal amount", "out_prncp_inv": "outst
            color="out_prncp_inv")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of Total payments received to date by To
fig = px.scatter(Lending_Club_Loan,
            x="total_rec_prncp",
            y="total_pymnt",
            title="Total payments received to date Distribution by  Total principal rece
            labels={"total_pymnt": "Total payments received to date", "total_rec_prncp":
            color="total_rec_prncp")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of Last credit report received  by Next
fig = px.scatter(Lending_Club_Loan,
             x="next_pymnt_d",
             y="last_pymnt_amnt",
             title="Last credit report  Distribution by Next payment Due",
             labels={"total_rec_late_fee": "last credit report ", "next_pymnt_d": "Next p
             color="next_pymnt_d")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# 4.3 Borrower_Information charts


# Create a scatter plot to show the distribution of loan grade by  Borrower Annual income
fig = px.scatter(Lending_Club_Loan,
             x="annual_inc",
             y="grade",
             title=" loan gradeDistribution by  Borrower Annual income",
             labels={"grade": " loan grade", "annual_inc": " Borrower Annual income"},
             color="annual_inc")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of Loan sub-grade by Loan funded by inve
fig = px.scatter(Lending_Club_Loan,
             x="funded_amnt_inv",
             y="sub_grade",
             title="Loan sub-grade Distribution by Loan funded by investors",
             labels={"sub_grade": "Loan sub-grade", "funded_amnt_inv": "Loan funded by in
             color="funded_amnt_inv")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of Total current balance of all installm
fig = px.scatter(Lending_Club_Loan,
             x="open_il_12m",
             y="total_bal_il",
             title="Total current balance of all installment accounts Distribution by ins
             labels={"total_bal_il": "Total current balance of all installment accounts",
             color="open_il_12m")  # Optional: Add color for better visualization

# Show the chart
```

```
fig.show()
```

```
# Create a scatter plot to show the distribution of  borrower's income verification  by
fig = px.scatter(Lending_Club_Loan,
             x="dti",
             y="verification_status",
             title="borrower's income verificationDistribution by Borrower's Debt-to-inco
             labels={"verification_status": "borrower's income verification", "dti": "Bor
             color="dti")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# 4.4 borrower's income verification charts


# Create a histogram plot to show the distribution of Revolving balance by Revolving util
fig = px.histogram(Lending_Club_Loan,
             x="revol_util",
             y="revol_bal",
             title="Revolving balanceDistribution by Revolving utilization rate",
             labels={"revol_bal": "Revolving balance", "revol_util": "Revolving utilizati
             color="revol_util")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a histogram plot to show the distribution of Initial listing status  by Current
fig = px.histogram(Lending_Club_Loan,
             x="rating",
             y="initial_list_status",
             title="Initial listing statusDistribution by Current loan rating",
             labels={"initial_list_status": "CInitial listing status", "rating": "Current
             color="rating")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```


```
# Create a box plot to show the distribution of Amount recovered after charge-off by Reco
fig = px.scatter(Lending_Club_Loan,
             x="collection_recovery_fee",
             y="recoveries",
             title="Amount recovered after charge-offDistribution by Recovery fee collect
             labels={"recoveries": "Amount recovered after charge-off", "collection_recov
             color="collection_recovery_fee")  # Optional: Add color for better visualiza

# Show the chart
fig.show()
```

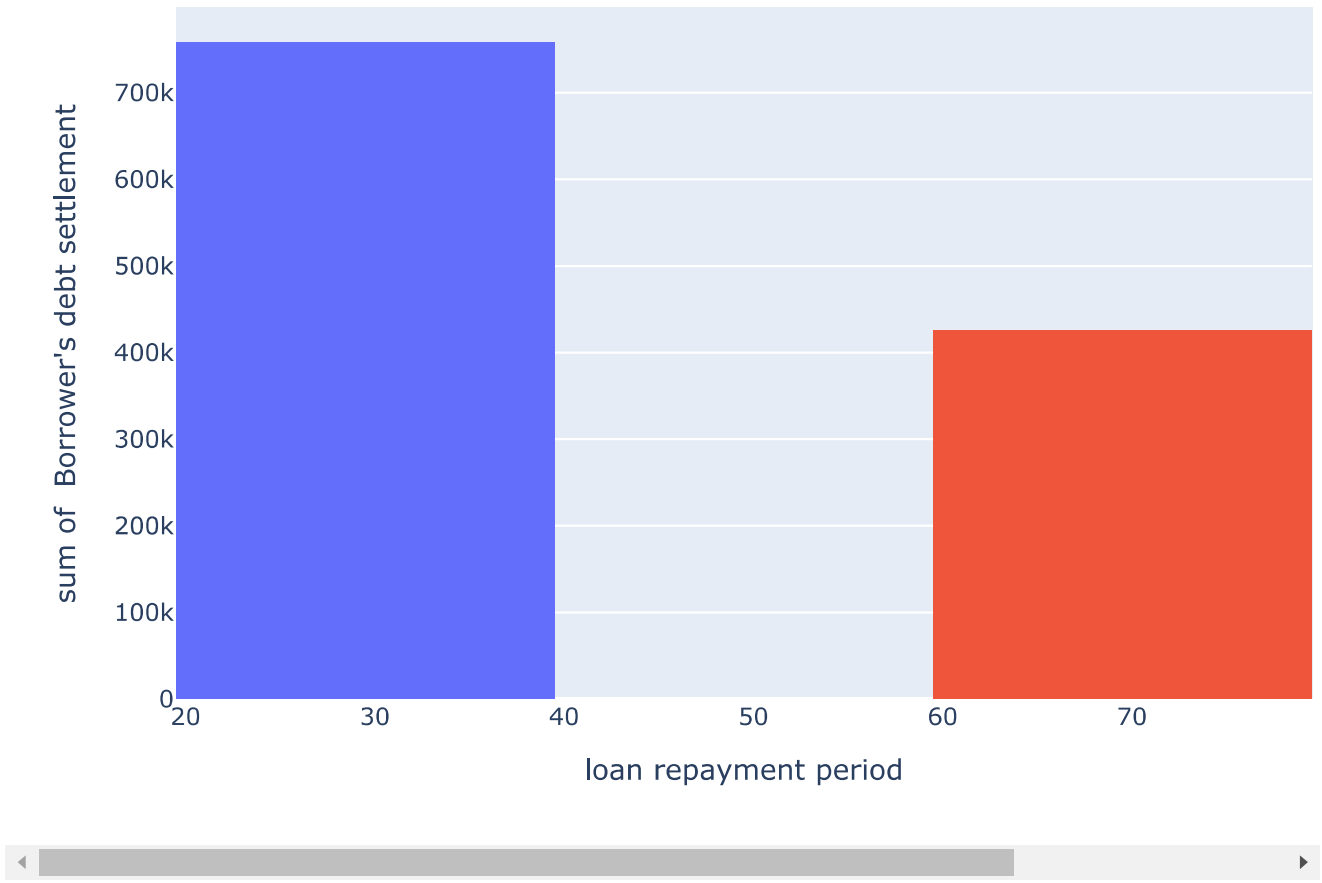
```
# Create a scatter plot to show the distribution of  Recovery fee Status by  Total collec
fig = px.scatter(Lending_Club_Loan,
             x="tot_coll_amt",
             y="collection_recovery_fee",
             title="Recovery fee Status Distribution by  Total collection amount ",
             labels={"collection_recovery_fee": "Recovery fee Status", "tot_coll_amt": "
             color="tot_coll_amt")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a box plot to show the distribution of Application is individual or joint by Cus
fig = px.box(Lending_Club_Loan,
             x="rating",
             y="application_type",
             title=" Application is individual or joint Distribution by Custom rating",
             labels={"application_type": " Application is individual or joint", "rating":
             color="rating")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of Borrower is in a hardship program by
fig = px.scatter(Lending_Club_Loan,
             x="term",
             y="hardship_flag",
             title=" Borrower is in a hardship program Distribution by loan repayment per
             labels={"hardship_flag": " Borrower is in a hardship program ", "term": "loa
             color="term")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a scatter plot to show the distribution of Method of loan disbursement by amount
fig = px.scatter(Lending_Club_Loan,
             x="loan_amnt",
             y="disbursement_method",
             title=" Method of loan disbursement Distribution by amount of money",
             labels={" disbursement_method": "Method of loan disbursement", "loan_amnt":
             color="loan_amnt")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```

```
# Create a histogram plot to show the distribution of Borrower's debt settlement by loan
fig = px.histogram(Lending_Club_Loan,
             x="term",
             y="debt_settlement_flag",
             title=" Borrower's debt settlement Distribution by loan repayment period",
             labels={"debt_settlement_flag": " Borrower's debt settlement", "term": "loan
             color="term")  # Optional: Add color for better visualization

# Show the chart
fig.show()
```




```
# 5. Correlation matrics
```

```
Lending_Club_Loan_imp.columns
```

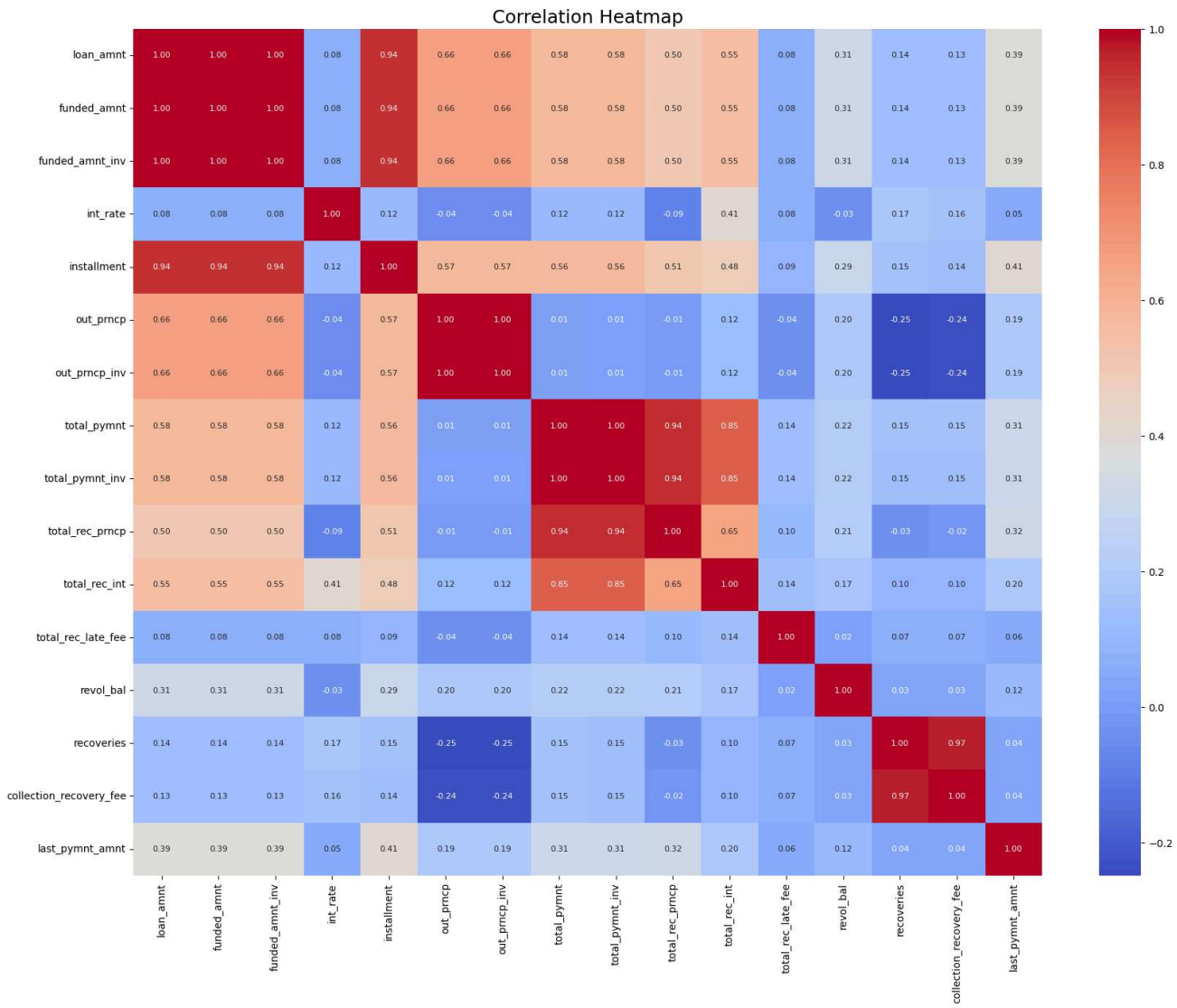
```
Index(['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
       'installment', 'pymnt_plan', 'purpose', 'addr_state', 'out_prncp',
       'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp',
       'total_rec_int', 'total_rec_late_fee', 'grade', 'sub_grade',
       'home_ownership', 'verification_status', 'revol_bal',
       'initial_list_status', 'recoveries', 'collection_recovery_fee',
       'last_pymnt_amnt', 'application_type', 'hardship_flag',
```

```
       'disbursement_method', 'debt_settlement_flag'],
      dtype='object')
```

```
Lending_Club_Loan.columns
```

```
Index(['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
       'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length',
       ...
       'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
       'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_credit_limit',
       'hardship_flag', 'disbursement_method', 'debt_settlement_flag',
       'rating'],
      dtype='object', length=102)
```

```
# Select numerical columns dynamically
Numerical_features = Lending_Club_Loan_imp.select_dtypes(include=['float64', 'int64']).cc

# Calculate correlation matrix
correlation = Lending_Club_Loan_imp[Numerical_features].corr()

# Create the heatmap
plt.figure(figsize=(20, 15))  # Adjust size as needed
sns.heatmap(
    correlation,
    annot=True,         # Add correlation values
    cmap='coolwarm',    # Color map
    fmt=".2f",          # Format for annotations
    annot_kws={"size": 8}  # Font size for annotations
)
plt.title('Correlation Heatmap', fontsize=18)
plt.xticks(fontsize=10, rotation=90)  # Rotate x-axis labels
plt.yticks(fontsize=10)  # Adjust y-axis font size
plt.show()
```


Correlation Heatmap
loan_amnt 1.00 1.00 1.00 0.08 0.94 0.66 0.66 0.58 0.58 0.50 0.55 0.08 0.31 0.14 0.13 0.39
funded_amnt 1.00 1.00 1.00 0.08 0.94 0.66 0.66 0.58 0.58 0.50 0.55 0.08 0.31 0.14 0.13 0.39
funded_amnt_inv 1.00 1.00 1.00 0.08 0.94 0.66 0.66 0.58 0.58 0.50 0.55 0.08 0.31 0.14 0.13 0.39
int_rate 0.08 0.08 0.08 1.00 0.12 -0.04 -0.04 0.12 0.12 -0.09 0.41 0.08 -0.03 0.17 0.16 0.05
installment 0.94 0.94 0.94 0.12 1.00 0.57 0.57 0.56 0.56 0.51 0.48 0.09 0.29 0.15 0.14 0.41
out_prncp 0.66 0.66 0.66 -0.04 0.57 1.00 1.00 0.01 0.01 -0.01 0.12 -0.04 0.20 -0.25 -0.24 0.19
out_prncp_inv 0.66 0.66 0.66 -0.04 0.57 1.00 1.00 0.01 0.01 -0.01 0.12 -0.04 0.20 -0.25 -0.24 0.19
total_pymnt 0.58 0.58 0.58 0.12 0.56 0.01 0.01 1.00 1.00 0.94 0.85 0.14 0.22 0.15 0.15 0.31
total_pymnt_inv 0.58 0.58 0.58 0.12 0.56 0.01 0.01 1.00 1.00 0.94 0.85 0.14 0.22 0.15 0.15 0.31
total_rec_prncp 0.50 0.50 0.50 -0.09 0.51 -0.01 -0.01 0.94 0.94 1.00 0.65 0.10 0.21 -0.03 -0.02 0.32
total_rec_int 0.55 0.55 0.55 0.41 0.48 0.12 0.12 0.85 0.85 0.65 1.00 0.14 0.17 0.10 0.10 0.20
total_rec_late_fee 0.08 0.08 0.08 0.08 0.09 -0.04 -0.04 0.14 0.14 0.10 0.14 1.00 0.02 0.07 0.07 0.06
revol_bal 0.31 0.31 0.31 -0.03 0.29 0.20 0.20 0.22 0.22 0.21 0.17 0.02 1.00 0.03 0.03 0.12
recoveries 0.14 0.14 0.14 0.17 0.15 -0.25 -0.25 0.15 0.15 -0.03 0.10 0.07 0.03 1.00 0.97 0.04
collection_recovery_fee 0.13 0.13 0.13 0.16 0.14 -0.24 -0.24 0.15 0.15 -0.02 0.10 0.07 0.03 0.97 1.00 0.04
last_pymnt_amnt 0.39 0.39 0.39 0.05 0.41 0.19 0.19 0.31 0.31 0.32 0.20 0.06 0.12 0.04 0.04 1.00
loan_amnt
funded_amnt
funded_amnt_inv
int_rate
installment
out_prncp
out_prncp_inv
total_pymnt
total_pymnt_inv
total_rec_prncp
total_rec_int
total_rec_late_fee
revol_bal
recoveries
collection_recovery_fee
last_pymnt_amnt
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

```
# 6. Implementing ML model ( Random Forest Classifier)


# Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report



# Filter columns that exist in the dataset
Numerical_features = [col for col in Numerical_features if col in Lending_Club_Loan.colum


# Splitting the Dataset
X = Lending_Club_Loan[Numerical_features]
y = Lending_Club_Loan['rating']
X_train , X_test , y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4


#Initalize and train the random forest classifier
rf_classifier = RandomForestClassifier(n_estimators=6, random_state=42)
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=6, random_state=42)
```

```
#predict and evaluate
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy:{accuracy:.2f}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report :\n", classification_report(y_test, y_pred))
```

```
Accuracy:0.97
Confusion Matrix:
 [[274708   1371]
 [  9500  79440]]
Classification Report :
               precision    recall  f1-score   support

           0       0.97      1.00      0.98    276079
           1       0.98      0.89      0.94     88940

    accuracy                           0.97    365019
   macro avg       0.97      0.94      0.96    365019
weighted avg       0.97      0.97      0.97    365019
```

```
# Using Gini-index for RandomForestClassifier


# Initialize and train the Decision Tree Classifier
classifier = RandomForestClassifier(criterion='gini', max_depth=4, random_state=42)
```

```
classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.97
Confusion Matrix:
[[274939   1140]
 [ 10299  78641]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98    276079
           1       0.99      0.88      0.93     88940

    accuracy                           0.97    365019
   macro avg       0.97      0.94      0.96    365019
weighted avg       0.97      0.97      0.97    365019
```

```
# Using Entropy for RandomForestClassifier


# Initialize and train the Decision Tree Classifier
classifier = RandomForestClassifier(criterion='entropy', max_depth=4, random_state=42)
classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.97
Confusion Matrix:
[[274608   1471]
 [ 10287  78653]]
Classification Report:
```

```
              precision    recall  f1-score   support
```