Context

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

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

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

➡️ Personal Loan ➡️ EMI Free Loan ➡️ Personal Overdraft ➡️ Advance Salary Loan

This project will focus on the underwriting process behind Personal Loan only.

➔ 1. Define Problem Statement and perform Exploratory Data Analysis

Problem Statement :

Objective: Determine the creditworthiness of applicants to decide if a personal loan should be approved and define optimal repayment terms.

Extended Problem Definition :

LoanTap aims to minimize the risk of non-performing assets (NPAs) by accurately identifying potential loan defaulters while maximizing the loan issuance to creditworthy customers. This approach should balance maximizing profit (interest earned on loans) with minimizing losses (defaults) through an effective underwriting process.

b) Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required), missing value detection, statistical summary.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('C:/SCALER\Case_Studies/7_LoanTrap/logistic_regressi
```

In [3]:
```
1 df.sample(5)
```

Out[3]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_lengt
184742	4000.0	36 months	8.90	127.02	A	A5	Lead Pastor	7 year
180934	7200.0	36 months	7.62	224.37	A	A3	Mountain Park Health Center	4 year
20219	12175.0	36 months	16.29	429.79	C	C4	MPP Group Of Companies	9 year
1938	17500.0	60 months	8.18	356.35	B	B1	Operations Coordinator	9 year
8903	20000.0	60 months	11.99	444.79	C	C1	NaN	Nal

5 rows × 27 columns

In [4]:
```
1 df.shape
```

Out[4]: (396030, 27)

Feature discription :-

loan_amnt : The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

term : The number of payments on the loan. Values are in months and can be either 36 or 60.

int_rate : Interest Rate on the loan

installment : The monthly payment owed by the borrower if the loan originates.

grade : LoanTap assigned loan grade

sub_grade : LoanTap assigned loan subgrade

emp_title :The job title supplied by the Borrower when applying for the loan.

emp_length : Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.

home_ownership : The home ownership status provided by the borrower during registration or obtained from the credit report.

annual_inc : The self-reported annual income provided by the borrower during registration.

verification_status : Indicates if income was verified by LoanTap, not verified, or if the income source was verified

issue_d : The month which the loan was funded

loan_status : Current status of the loan - Target Variable

purpose : A category provided by the borrower for the loan request.

title : The loan title provided by the borrower

dti : A ratio calculated using the borrower’s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LoanTap loan, divided by the borrower’s self-reported monthly income.

earliest_cr_line :The month the borrower's earliest reported credit line was opened open_acc : The number of open credit lines in the borrower's credit file.

pub_rec : Number of derogatory public records

revol_bal : Total credit revolving balance

revol_util : Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.

total_acc : The total number of credit lines currently in the borrower's credit file

initial_list_status : The initial listing status of the loan. Possible values are – W, F

application_type : Indicates whether the loan is an individual application or a joint application with two co-borrowers mort_acc : Number of mortgage accounts.

pub_rec_bankruptcies : Number of public record bankruptcies Address: Address of the individual

In [5]:
```
1 df.info()
```

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

In [6]:
```
1 df.isnull().sum().sort_values(ascending=False).head(10)
```

Out[6]:
```
mort_acc                37795
emp_title               22927
emp_length              18301
title                    1756
pub_rec_bankruptcies      535
revol_util                276
loan_amnt                   0
dti                         0
application_type            0
initial_list_status         0
dtype: int64
```

In [7]:
```
1 df.describe()
```

Out[7]:

	loan_amnt	int_rate	installment	annual_inc	dti	op
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.

In [8]:
```
1 df.describe(include='object')
```

Out[8]:

	term	grade	sub_grade	emp_title	emp_length	home_ownership	verification_s
count	396030	396030	396030	373103	377729	396030	39
unique	2	7	35	173105	11	6	
top	36 months	B	B3	Teacher	10+ years	MORTGAGE	Ve
freq	302005	116018	26655	4389	126041	198348	13

c) Univariate Analysis

In [9]:
```
1 object_features = []
2 numerical_features = []
3
4 for col in df.columns:
5     if df[col].dtype==object:
6         object_features.append(col)
7     else:
8         numerical_features.append(col)
```

In [10]:
```
1 print(object_features)
```

```
['term', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'verification_status', 'issue_d', 'loan_status', 'purpose', 'title', 'earliest_cr_line', 'initial_list_status', 'application_type', 'address']
```

In [11]:
```
print(numerical_features)
```

```
['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_ac
c', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'pub_
rec_bankruptcies']
```

In [12]:
```
# List of numerical features (columns)
numerical_features = ['loan_amnt', 'int_rate', 'installment', 'annual
                      'pub_rec', 'revol_bal', 'revol_util', 'total_ac

# Check skewness for each numerical feature
skewness_dict = {}
for feature in numerical_features:
    skewness_dict[feature] = df[feature].skew()

# Display skewness for each numerical column
for feature, skew in skewness_dict.items():
    print(f"Skewness of {feature}: {skew}")
```

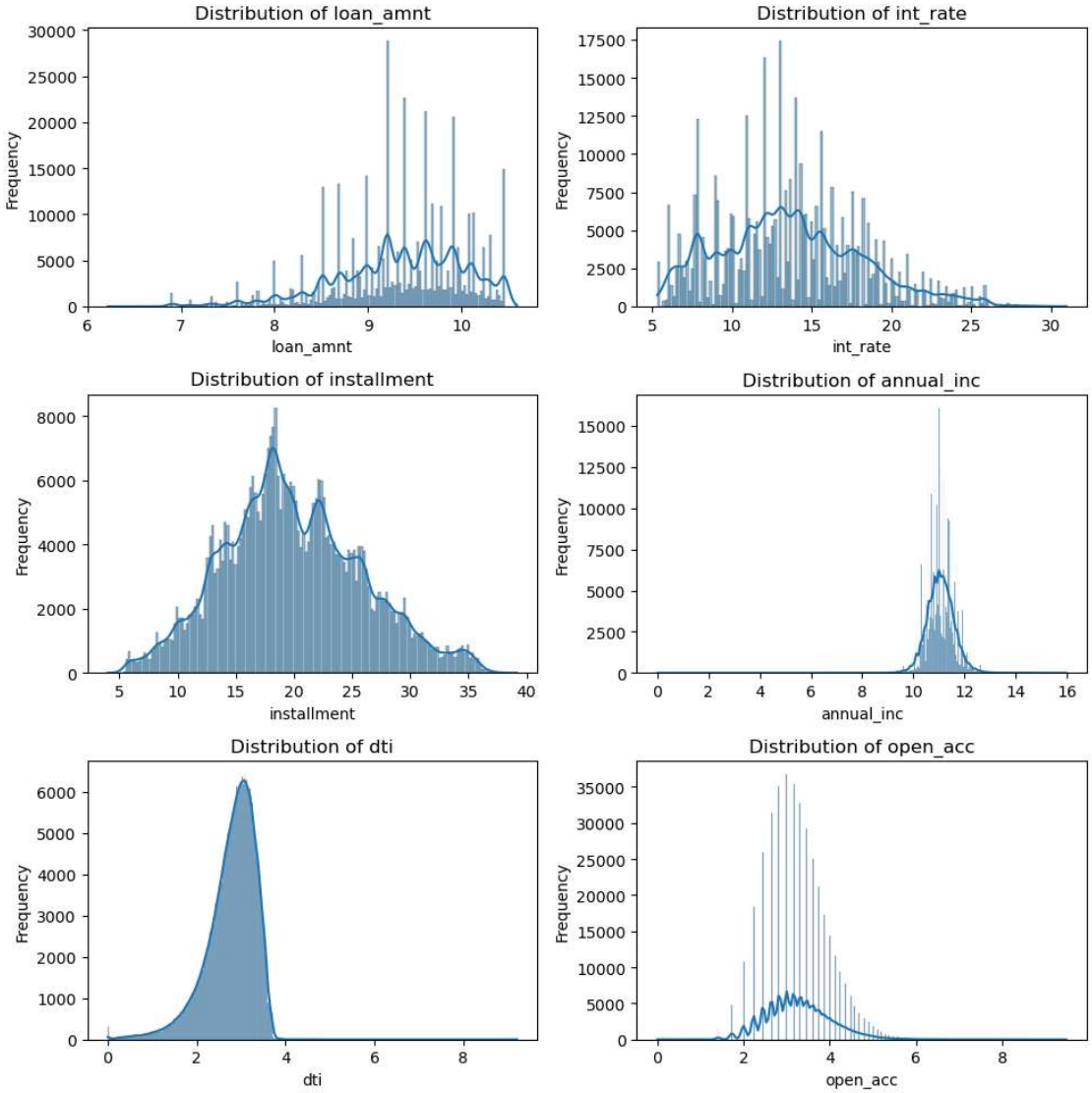
```
Skewness of loan_amnt: 0.777285467097746
Skewness of int_rate: 0.4206694719582165
Skewness of installment: 0.983598160890438
Skewness of annual_inc: 41.042724746560665
Skewness of dti: 431.05122535490415
Skewness of open_acc: 1.2130188444585455
Skewness of pub_rec: 16.576564199464457
Skewness of revol_bal: 11.727515124126919
Skewness of revol_util: -0.07177802032853098
Skewness of total_acc: 0.8643276369429
Skewness of mort_acc: 1.6001324380874855
Skewness of pub_rec_bankruptcies: 3.4234403681961583
```

In [13]:

```
from scipy import stats

def apply_log_transformation(df, features):
    for feature in features:
        df[feature] = np.log1p(df[feature])  # log1p to handle zero v
    return df

def apply_sqrt_transformation(df, features):
    for feature in features:
        df[feature] = np.sqrt(df[feature])
    return df

log_transformed_features = ['loan_amnt', 'annual_inc', 'dti', 'pub_re
sqrt_transformed_features = ['installment', 'open_acc', 'total_acc']

df = apply_log_transformation(df, log_transformed_features)

df = apply_sqrt_transformation(df, sqrt_transformed_features)

print("\nSkewness after transformation:")
for feature in df.columns:
    if feature in numerical_features:
        print(f"Skewness of {feature}: {df[feature].skew()}")
```

```
Skewness after transformation:
Skewness of loan_amnt: -0.6680611188267545
Skewness of int_rate: 0.4206694719582165
Skewness of installment: 0.26130316409228166
Skewness of annual_inc: 0.16622501307433268
Skewness of dti: -1.230713217981101
Skewness of open_acc: 0.4269654089750745
Skewness of pub_rec: 2.5354727777923403
Skewness of revol_bal: -2.8518972748189184
Skewness of revol_util: -0.07177802032853098
Skewness of total_acc: 0.17769891689572875
Skewness of mort_acc: 0.29647893480908233
Skewness of pub_rec_bankruptcies: 2.6217121173045226
```

In [14]:
```
numerical_features1 = ['loan_amnt', 'int_rate', 'installment', 'annua

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

for i,var in enumerate(numerical_features1,1):
    plt.subplot(3,2,i)
    sns.histplot(df[var],kde=True)
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Based on these distribution plots, we can make the following observations:

➤ Loan Amount (loan_amnt):

The distribution appears multimodal, with peaks at certain loan amounts. This suggests that specific loan amounts are more common, possibly due to LoanTap's product design or

➤ Interest Rate (int_rate):

The interest rate distribution appears right-skewed, with a higher concentration of loans in the 10-20% range. Few loans have interest rates above 25%, indicating that higher rates are less common.

➤ Installment (installment):

This distribution is also right-skewed, with most installments clustered between 200 and 500. This likely reflects typical monthly payments for the majority of borrowers and suggests that smaller installments are more popular.

➤ Annual Income (annual_inc):

The annual income distribution is highly right-skewed, with a long tail reaching into the higher income levels. Most borrowers have an income below a certain range (likely under 200,000), but there are a few individuals with very high incomes, possibly skewing the mean.

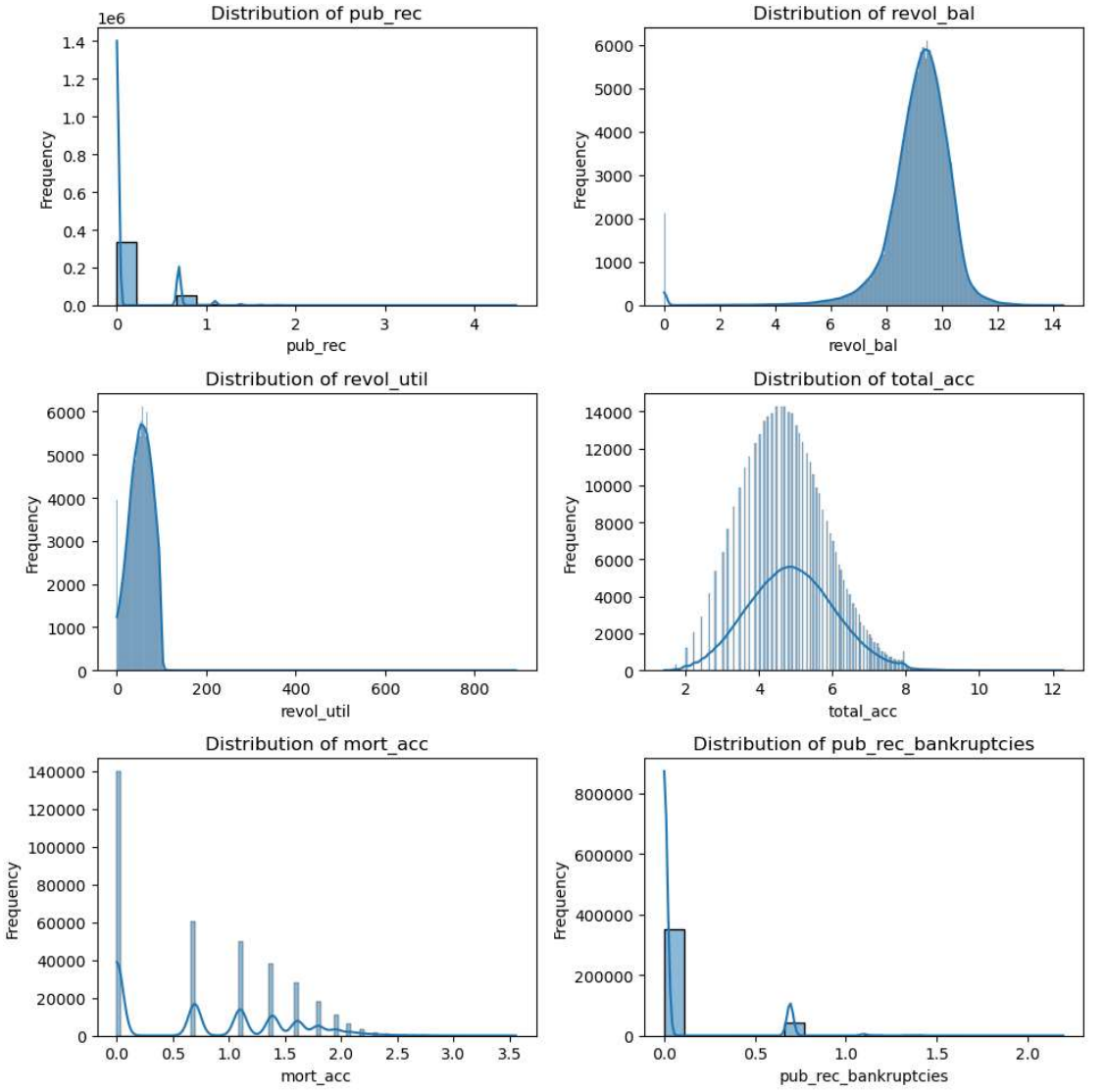
➤ Debt-to-Income Ratio (dti):

The dti distribution is also right-skewed, with a large concentration at the lower end (close to zero). This suggests that many borrowers have manageable debt relative to their income. A few outliers with very high dti ratios indicate borrowers who may be over-leveraged, although these cases seem rare.

➤ Open Accounts (open_acc):

The distribution of open accounts is concentrated below 20, indicating that most borrowers have a relatively small number of open credit lines. There are fewer borrowers with a higher number of open accounts, suggesting that such cases are less common and possibly reflect higher credit activity.

In [15]:
```
numerical_features2 = ['pub_rec', 'revol_bal', 'revol_util', 'total_a

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

for i,var in enumerate(numerical_features2,1):
    plt.subplot(3,2,i)
    sns.histplot(df[var],kde=True)
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Based on these distribution plots, we can make the following observations:

➤ pub_rec (Public Records) :

The high positive skew in the pub_rec distribution indicates that most individuals have a clean credit history with few or no public records. However, a small segment of the population has a significantly higher number of public records, which could be due to factors like bankruptcies, foreclosures, or tax liens.

➤ revol_bal (Revolving Balance) :

The right-skewed distribution of revol_bal suggests that most individuals have low revolving balances (credit card debt), while a few individuals carry substantial credit card debt. This could be attributed to factors like income levels, spending habits, and access to credit.

➤ revol_util (Revolving Utilization Rate) :

The negative skew in the revol_util distribution indicates that most individuals maintain low credit utilization rates, which is generally considered a positive factor for creditworthiness. However, a smaller group of individuals have high utilization rates, potentially putting them at risk of higher interest rates and reduced credit scores.

➤ total_acc (Total Number of Accounts) :

The approximately normal distribution of total_acc suggests that the number of total credit accounts is relatively evenly distributed across the population. This indicates a diverse range of credit histories, with some individuals having many accounts and others having fewer.

➤ mort_acc (Number of Mortgage Accounts) :

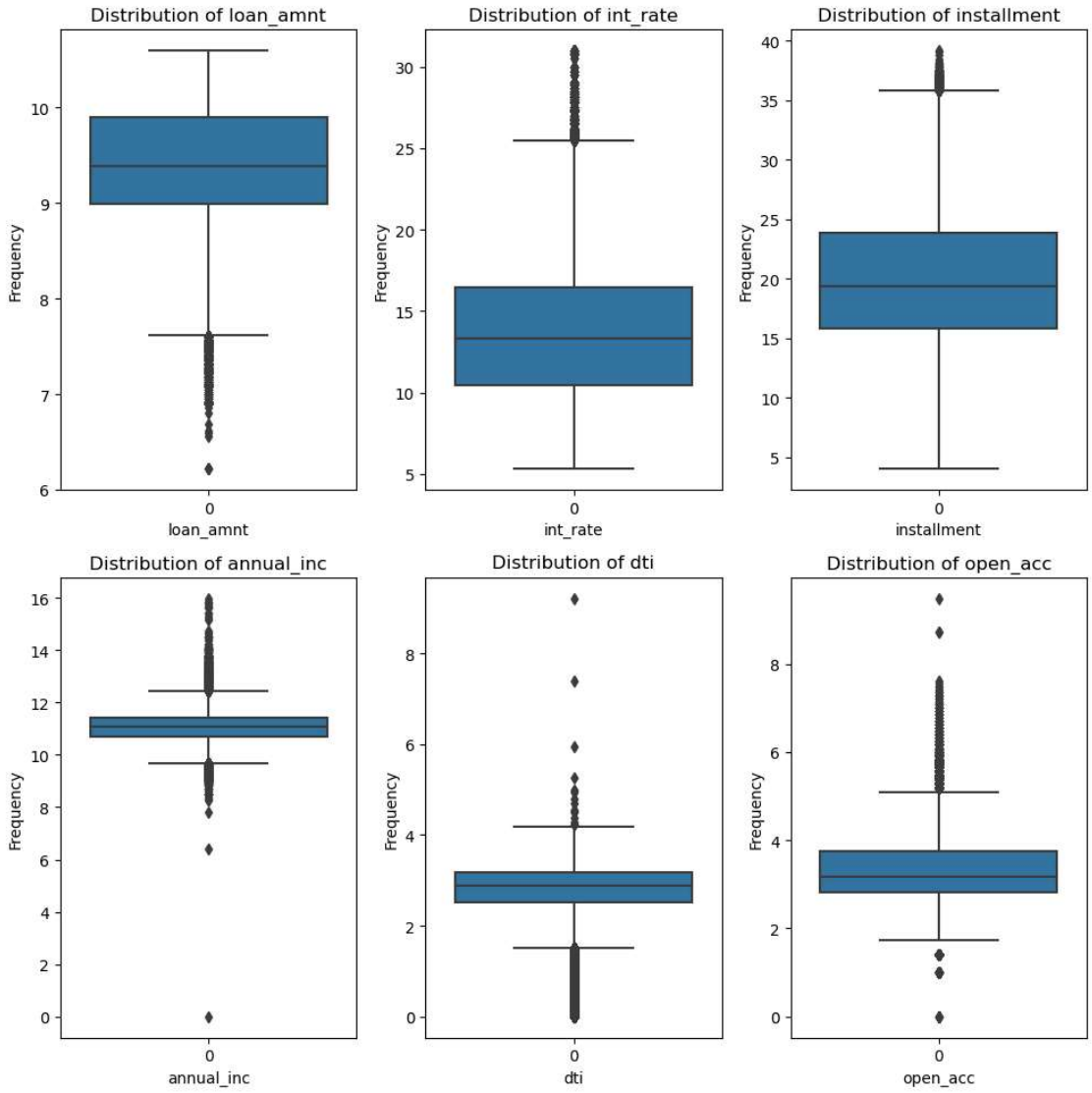
The high positive skew in the mort_acc distribution indicates that most individuals have few or no mortgage accounts. However, a smaller segment of the population has multiple mortgage accounts, possibly due to factors like homeownership, real estate investments, or refinancing.

➤ pub_rec_bankruptcies (Public Record Bankruptcies) :

The high positive skew in the pub_rec_bankruptcies distribution indicates that most individuals

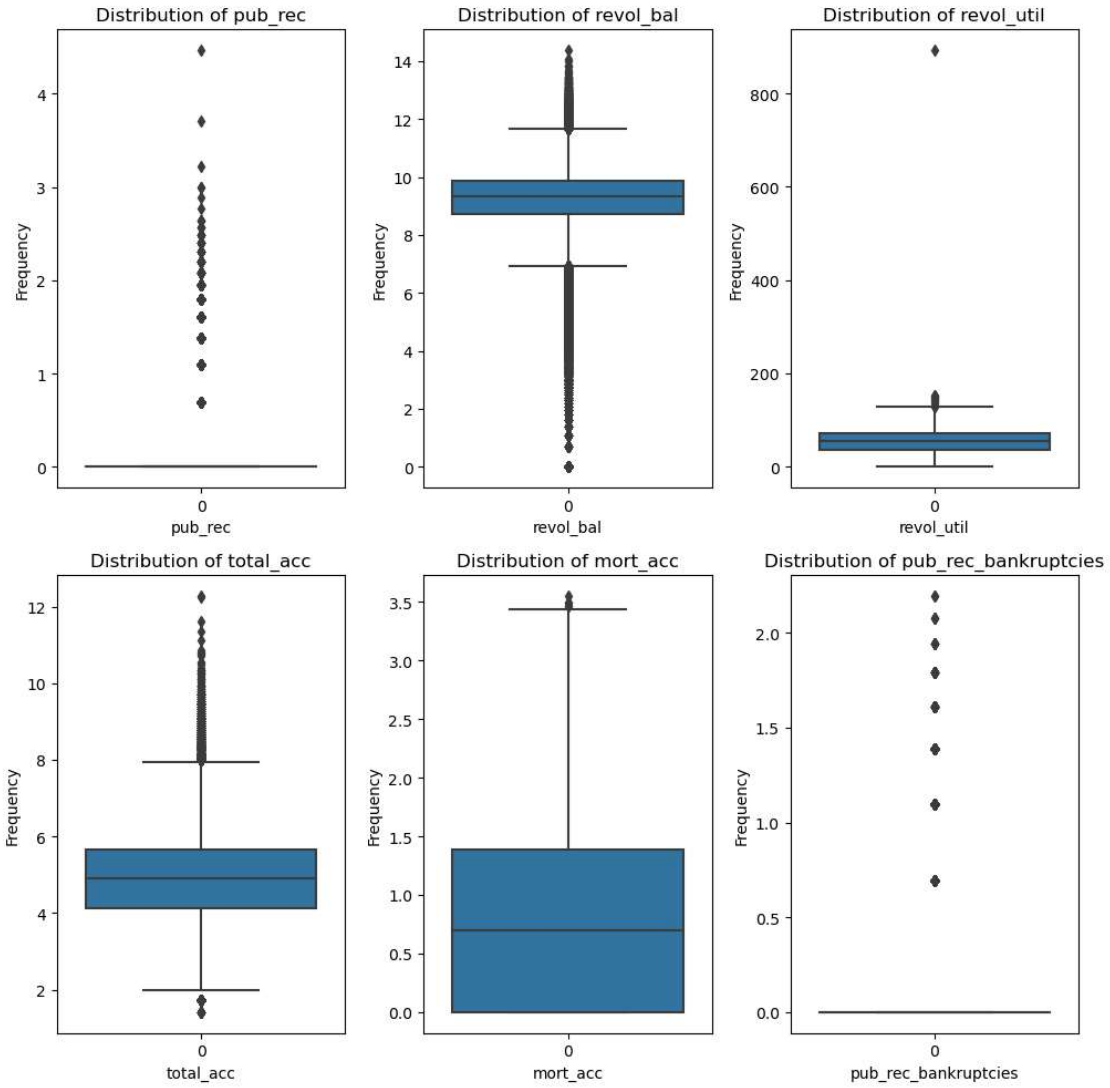
In [16]:

```
numerical_features1 = ['loan_amnt', 'int_rate', 'installment', 'annua

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

for i,var in enumerate(numerical_features1,1):
    plt.subplot(2,3,i)
    sns.boxplot(df[var])
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



In [17]:

```
numerical_features2 = ['pub_rec', 'revol_bal', 'revol_util', 'total_a

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

for i,var in enumerate(numerical_features2,1):
    plt.subplot(2,3,i)
    sns.boxplot(df[var])
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

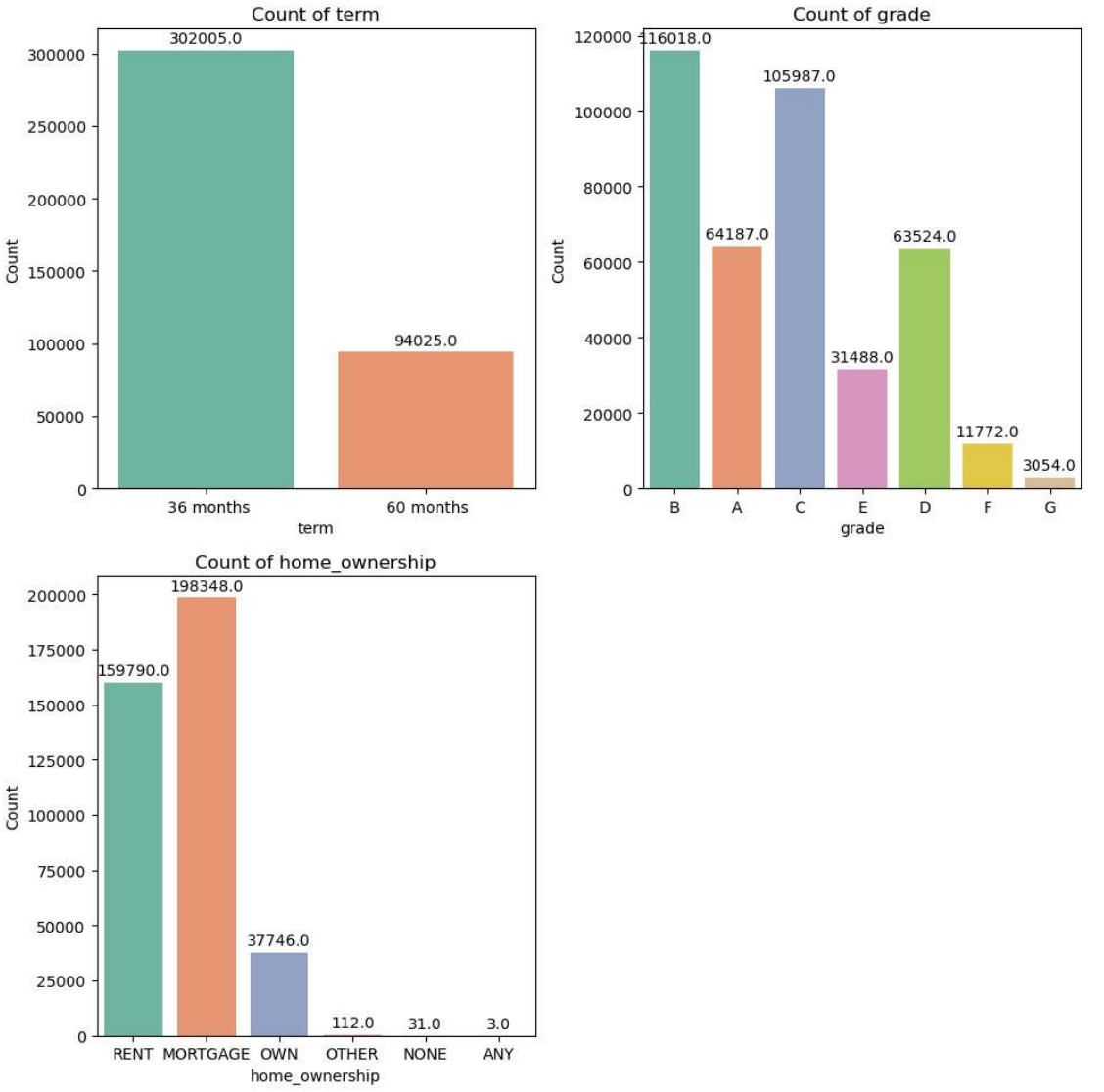


In [18]:

```
categorical_vars = ['term', 'grade','emp_length', 'home_ownership', '
                    'loan_status', 'initial_list_status', 'applicati
```

In [19]:

```
categorical_vars1 = ['term', 'grade', 'home_ownership']

plt.figure(figsize=(10, 10))
for i, var in enumerate(categorical_vars1, 1):
    plt.subplot(2, 2, i)
    ax = sns.countplot(x=var, data=df, palette='Set2')
    plt.title(f'Count of {var}')
    plt.xlabel(var)
    plt.ylabel('Count')

    for p in ax.patches:
        ax.annotate(f'{p.get_height()}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='baseline', fontsize=10, color='b
                    textcoords='offset points')
plt.tight_layout()
plt.show()
```



Observations :-

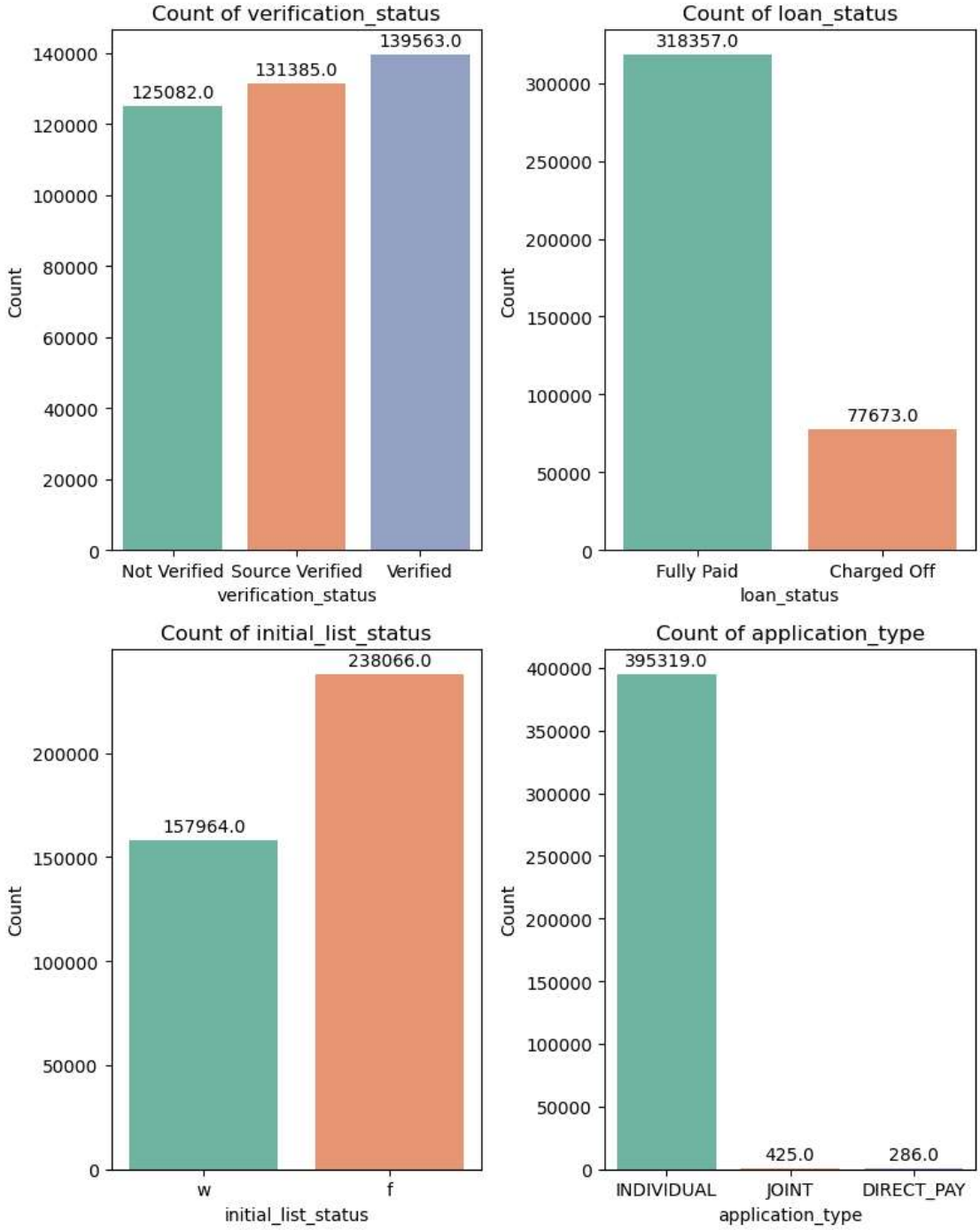
➤ term : The data shows a highly imbalanced distribution between 36 months and 60 months , with 36 months having significantly more observations.The countplot shows majority of people took loan for 36 months rest took for 60 months.

➤ grade : Grade B has the highest count, while Category G is the least frequent in the dataset.

➤ home_ownership : The 90 % of the data is distributed among rent, mortage, and own.

In [20]:

```
categorical_vars2 = ['verification_status', 'loan_status', 'initial_l

plt.figure(figsize=(8, 10))
for i, var in enumerate(categorical_vars2, 1):
    plt.subplot(2, 2, i)
    ax = sns.countplot(x=var, data=df, palette='Set2')
    plt.title(f'Count of {var}')
    plt.xlabel(var)
    plt.ylabel('Count')

    for p in ax.patches:
        ax.annotate(f'{p.get_height()}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='baseline', fontsize=10, color='b
                    textcoords='offset points')

plt.tight_layout()
plt.show()
```



Observations :-

➤ verification_status : The countplot displays an almost uniform distribution across the categories, indicating balanced data.Number of verified people > source verified people > not verified people.

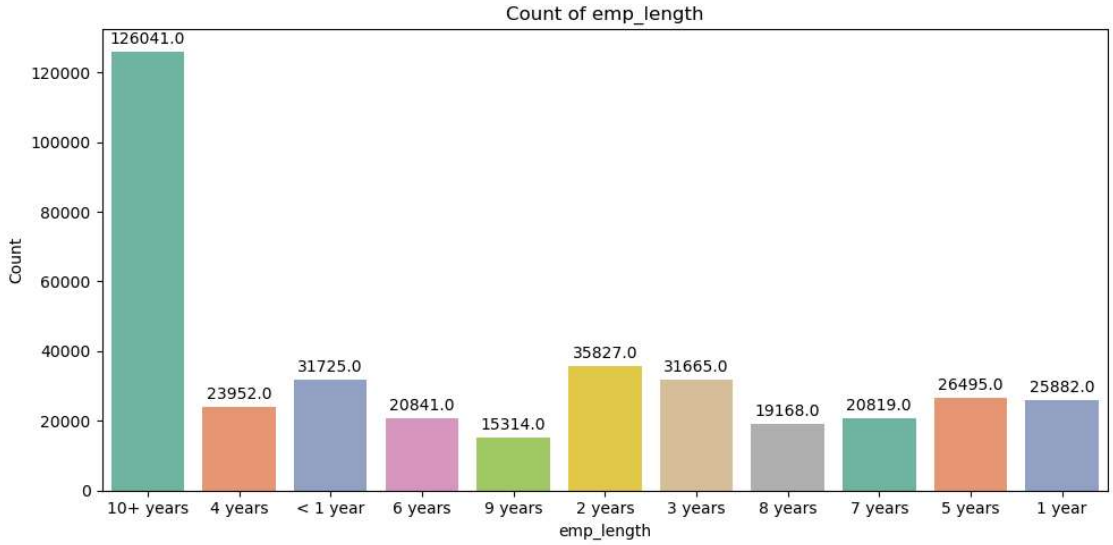
➤ loan_status : The data shows a highly imbalanced distribution between Fully paid and charged off , with most of the people has fully paid their loan.

➤ initial_list_status : Majority of people having only part (F) of the loan is available for funding or investment at first where other people is listed as a whole loan (W), and it is available for full funding or investment.

➤ application_type : Almost 99 % of people has application type as individual. Where the number of people with Joint and direct pay has very few data points, which might indicate an outlier or an uncommon event

In [21]:

```
plt.figure(figsize=(10, 5))

ax = sns.countplot(x=df['emp_length'], data=df, palette='Set2')
plt.title('Count of emp_length')
plt.xlabel('emp_length')
plt.ylabel('Count')

for p in ax.patches:
    ax.annotate(f'{p.get_height()}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='baseline', fontsize=10, color='black
                textcoords='offset points')

plt.tight_layout()
plt.show()
```

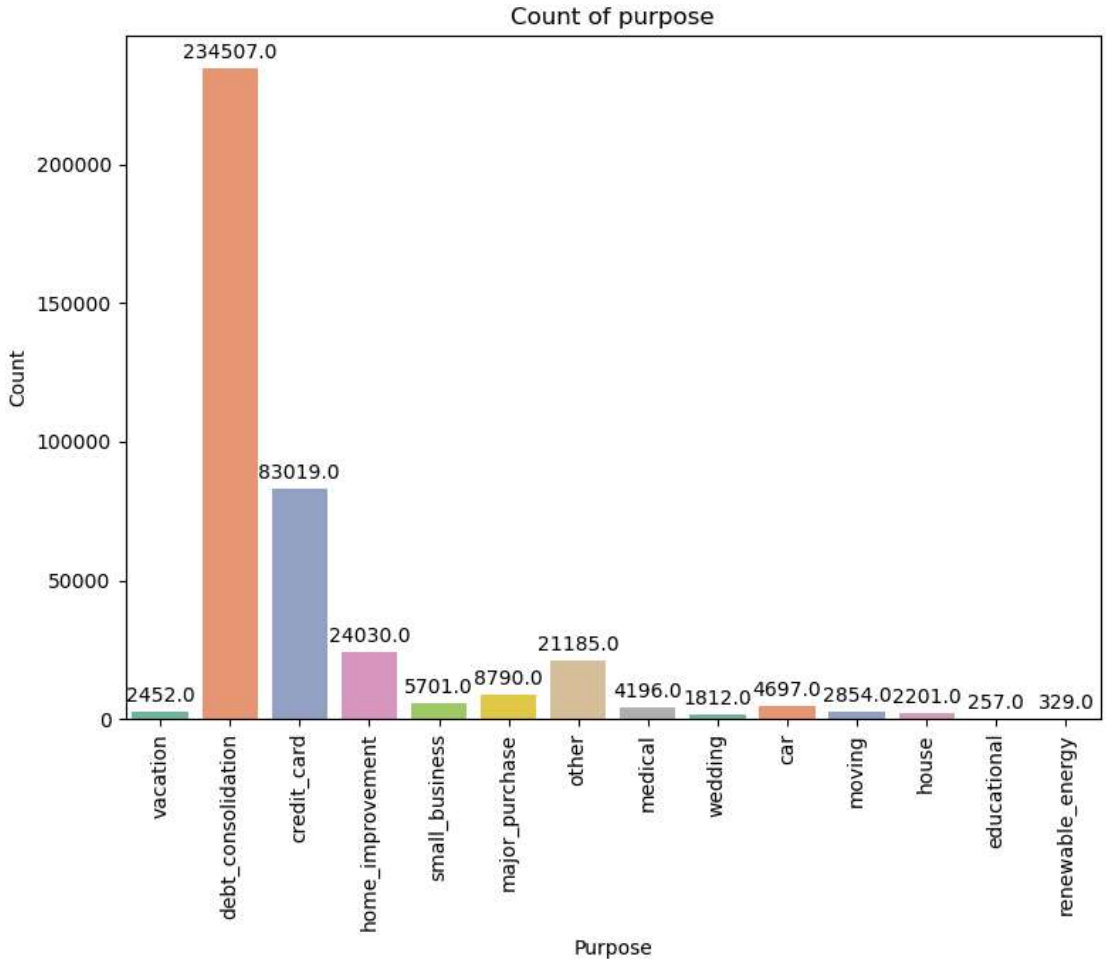


Observation :-

People having 10+ years experience has the highest count, while People having 9 years is the least frequent in the dataset. The countplot shows a right-skewed distribution with a large number of instances in 10+ years, moderate between <1 years, 2 years, 3 years, 5 years, 1 year,4 years,6 years and fewer in 8 years ,9 years suggesting that the model may have a bias towards People having 10+ years experience.

In [22]:

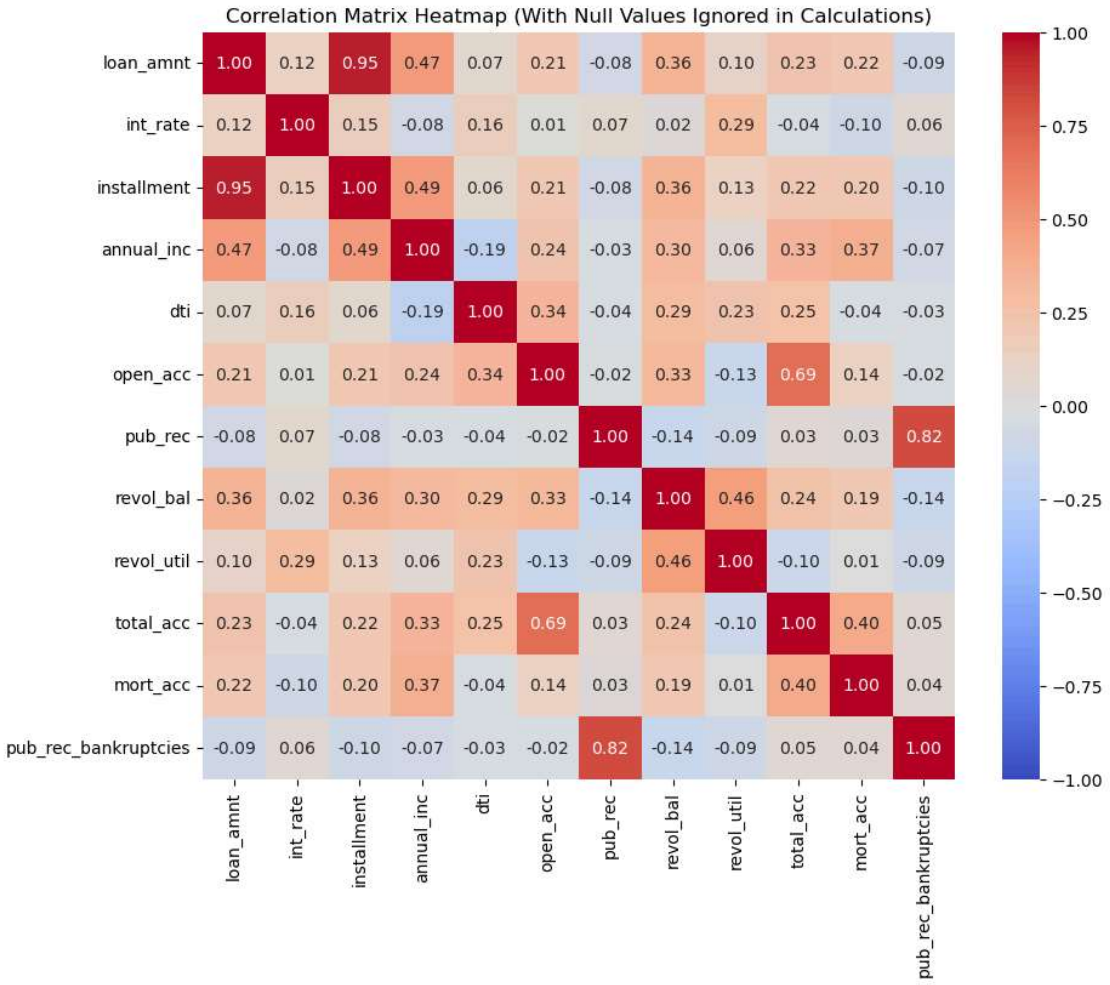
```
plt.figure(figsize=(8, 7))

ax = sns.countplot(x=df['purpose'], data=df, palette='Set2')
plt.title('Count of purpose')
plt.xlabel('Purpose')
plt.ylabel('Count')
plt.xticks(rotation=90)

for p in ax.patches:
    ax.annotate(f'{p.get_height()}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='baseline', fontsize=10, color='black
                textcoords='offset points')
plt.tight_layout()
plt.show()
```



Observation :-

debt consolidation has the highest count, while educational is the least frequent in the dataset. small business, mdeical , car , moving ,house has very few data points, vacation ,wedding, educational, renewable energy might indicate an outlier.

In [30]:
```
correlation_matrix = df[numerical_features].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f
plt.title("Correlation Matrix Heatmap (With Null Values Ignored in Ca
plt.show()
```



==========

➔ 2. Data Preprocessing

a) Duplicate value check

In [31]:
```
df.duplicated().sum()
```

Out[31]: 0

b) Missing value treatment

In [32]:
```
null_percentage = df.isnull().sum()
null_percentage.sort_values(ascending = False).head(8)
```

Out[32]:
```
mort_acc                37795
emp_title               22927
emp_length              18301
title                    1756
pub_rec_bankruptcies      535
revol_util                276
loan_amnt                   0
dti                         0
dtype: int64
```

In [33]:
```
df.shape
```

Out[33]: (396030, 27)

In [34]:
```
df['revol_util'] = df.groupby('grade')['revol_util'].transform(lambda
```

In [35]:
```
df['pub_rec_bankruptcies'] = df.groupby('grade')['pub_rec_bankruptcie
```

In [36]:
```
df['title'] = df.groupby('grade')['title'].transform(lambda x: x.fill
```

In [37]:
```
df['emp_length'] = df.groupby('grade')['emp_length'].transform(lambda
```

In [38]:
```
df['emp_title'] = df.groupby(['grade', 'emp_length'])['emp_title'].tr
```

In [39]:
```
df['mort_acc'] = df.groupby('grade')['mort_acc'].transform(lambda x:
```

In [40]:
```
null_percentage = df.isnull().sum()
null_percentage.sort_values(ascending = False).head(8)
```

Out[40]:
```
loan_amnt               0
title                   0
pub_rec_bankruptcies    0
mort_acc                0
application_type        0
initial_list_status     0
total_acc               0
revol_util              0
dtype: int64
```

c) Outlier treatment

In [41]:
```
numerical_columns =[]
for i in df.columns:
    if (df[i].dtypes==float):
        numerical_columns.append(i)

print(numerical_columns)
```

```
['loan_amnt', 'int_rate', 'installment', 'annual_inc', 'dti', 'open_ac
c', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc', 'pub_
rec_bankruptcies']
```

In [42]:
```
from scipy import stats

z_scores = stats.zscore(df[numerical_columns])
z_scores_df = pd.DataFrame(z_scores, columns=numerical_columns)
threshold = 3  # Typically values greater than 3 are considered outli
outliers = (z_scores_df.abs() > threshold)
outlier_rows = df[outliers.any(axis=1)]

outlier_rows.shape
```

Out[42]:
```
(25473, 27)
```

In [43]:
```
df_cleaned = df[~outliers.any(axis=1)]
df_cleaned.shape
```

Out[43]:
```
(370557, 27)
```

In [44]:
```
df_cleaned.columns
```

Out[44]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_gra
de',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'titl
e',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'application_ty
pe',
       'mort_acc', 'pub_rec_bankruptcies', 'address'],
      dtype='object')
```

In [45]:
```
df_cleaned
```

Out[45]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_leng
0	9.210440	36 months	11.44	18.151584	B	B4	Marketing	10+ yea
1	8.987322	36 months	11.99	16.299693	B	B5	Credit analyst	4 yea
2	9.655090	36 months	10.49	22.515994	B	B3	Statistician	< 1 ye
3	8.881975	36 months	6.49	14.854292	A	A2	Client Advocate	6 yea
4	10.101354	60 months	17.27	24.684611	C	C5	Destiny Management Inc.	9 yea
...	...	...	...	...	...	...	...	
396025	9.210440	60 months	10.99	14.743812	B	B4	licensed bankere	2 yea
396026	9.952325	36 months	12.29	26.465449	C	C1	Agent	5 yea
396027	8.517393	36 months	9.99	12.701181	B	B1	City Carrier	10+ yea
396028	9.952325	60 months	15.31	22.428107	C	C2	Gracon Services, Inc	10+ yea
396029	7.601402	36 months	13.61	8.244998	C	C2	Internal Revenue Service	10+ yea

370557 rows × 27 columns

d) Feature engineering

In [46]:
```
# Convert to datetime format
df['issue_d'] = pd.to_datetime(df['issue_d'], format='%b-%Y')
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'], forma
```

In [47]:
```
# Extract month and year
df['loan_issue_month'] = df['issue_d'].dt.month
df['loan_issue_year'] = df['issue_d'].dt.year
df['earliest_cr_line_month'] = df['earliest_cr_line'].dt.month
df['earliest_cr_line_year'] = df['earliest_cr_line'].dt.year
```

In [48]:
```
df['credit_history_length_years'] = (df['issue_d'] - df['earliest_cr_
```

In [49]:
```
df['credit_history_length_years'] = df['credit_history_length_years']
```

In [50]:
```
df_cleaned.drop(['issue_d', 'earliest_cr_line'], axis=1, inplace=True
```

```
C:\Users\Prajwal\AppData\Local\Temp\ipykernel_2948\3795973535.py:1: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df_cleaned.drop(['issue_d', 'earliest_cr_line'], axis=1, inplace=True)
```

e) Data Preparation for Modeling.

In [51]:
```
categorical_columns = df.select_dtypes(include=['object', 'category']

for col in categorical_columns:
    unique_count = df[col].nunique()
    if unique_count < 8:
        print(f"{col}: {unique_count}")
```

```
term: 2
grade: 7
home_ownership: 6
verification_status: 3
loan_status: 2
initial_list_status: 2
application_type: 3
```

In [52]:
```
# Columns to one-hot encode
columns_to_encode = ['grade', 'home_ownership', 'verification_status'
df_cleaned = pd.get_dummies(df_cleaned, columns=columns_to_encode, dr
```

In [53]:
```
# One-hot encoding for the 'term' column
df_cleaned = pd.get_dummies(df_cleaned, columns=['term'], prefix='ter
```

In [54]:
```
# Apply One-Hot Encoding to 'purpose' and 'emp_length'
df_cleaned = pd.get_dummies(df_cleaned, columns=['purpose', 'emp_leng
df_cleaned.head(5)
```

Out[54]:

	loan_amnt	int_rate	installment	sub_grade	emp_title	annual_inc	title	
0	9.210440	11.44	18.151584	B4	Marketing	11.669938	Vacation	3.304
1	8.987322	11.99	16.299693	B5	Credit analyst	11.082158	Debt consolidation	3.137
2	9.655090	10.49	22.515994	B3	Statistician	10.670303	Credit card refinancing	2.623
3	8.881975	6.49	14.854292	A2	Client Advocate	10.896758	Credit card refinancing	1.280
4	10.101354	17.27	24.684611	C5	Destiny Management Inc.	10.915107	Credit Card Refinance	3.553

5 rows × 57 columns

In [55]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df_cleaned['sub_grade_encoded'] = label_encoder.fit_transform(df_clea

df_cleaned.drop(['sub_grade'], axis=1, inplace=True)

df_cleaned.head()
```

Out[55]:

	loan_amnt	int_rate	installment	emp_title	annual_inc	title	dti	open_
0	9.210440	11.44	18.151584	Marketing	11.669938	Vacation	3.304686	4.000
1	8.987322	11.99	16.299693	Credit analyst	11.082158	Debt consolidation	3.137666	4.123
2	9.655090	10.49	22.515994	Statistician	10.670303	Credit card refinancing	2.623944	3.605
3	8.881975	6.49	14.854292	Client Advocate	10.896758	Credit card refinancing	1.280934	2.449
4	10.101354	17.27	24.684611	Destiny Management Inc.	10.915107	Credit Card Refinance	3.553918	3.605

5 rows × 57 columns

In [56]:
```
categorical_columns = df_cleaned.select_dtypes(include=['object', 'ca

for col in categorical_columns:
    unique_count = df_cleaned[col].nunique()
    print(f"{col}: {unique_count}")
```

```
emp_title: 164695
title: 45763
address: 368494
```

In [57]:
```
# Frequency encoding for emp_title
emp_title_counts = df_cleaned['emp_title'].value_counts()
df_cleaned['emp_title_encoded'] = df_cleaned['emp_title'].map(emp_tit

# Frequency encoding for title
title_counts = df_cleaned['title'].value_counts()
df_cleaned['title_encoded'] = df_cleaned['title'].map(title_counts)
```

In [58]:
```
# Drop original 'emp_title' and 'title' columns
df_cleaned.drop(['emp_title', 'title','address'], axis=1, inplace=Tru
```

In [59]:
```
df_cleaned.sample(5)
```

Out[59]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_
54685	9.680406	18.49	20.262774	10.778977	2.520113	3.000000	0.693147	8.464
79362	10.463132	13.98	28.531211	11.407576	3.154444	3.000000	0.000000	11.478
128629	9.974691	21.18	24.148499	10.933125	3.180551	3.464102	0.000000	8.754
2342	10.239996	11.14	30.307590	11.451061	1.803359	3.162278	0.000000	10.278
230457	9.952325	11.14	26.247095	11.849405	1.791759	2.449490	0.000000	8.931

5 rows × 56 columns

In [60]:
```
categorical_columns = df_cleaned.select_dtypes(include=['object', 'ca

for col in categorical_columns:
    unique_count = df_cleaned[col].nunique()
    print(f"{col}: {unique_count}")
```

In [61]:
```
# Select boolean columns and convert them to 0 and 1
bool_columns = df_cleaned.select_dtypes(include=['bool']).columns
df_cleaned[bool_columns] = df_cleaned[bool_columns].astype(int)
```

In [62]:

```
1 df_cleaned.dtypes
```

Out[62]:

loan_amnt	float64
int_rate	float64
installment	float64
annual_inc	float64
dti	float64
open_acc	float64
pub_rec	float64
revol_bal	float64
revol_util	float64
total_acc	float64
mort_acc	float64
pub_rec_bankruptcies	float64
grade_B	int32
grade_C	int32
grade_D	int32
grade_E	int32
grade_F	int32
grade_G	int32
home_ownership_MORTGAGE	int32
home_ownership_NONE	int32
home_ownership_OTHER	int32
home_ownership_OWN	int32
home_ownership_RENT	int32
verification_status_Source Verified	int32
verification_status_Verified	int32
loan_status_Fully Paid	int32
initial_list_status_w	int32
application_type_INDIVIDUAL	int32
application_type_JOINT	int32
term_ 60 months	int32
purpose_credit_card	int32
purpose_debt_consolidation	int32
purpose_educational	int32
purpose_home_improvement	int32
purpose_house	int32
purpose_major_purchase	int32
purpose_medical	int32
purpose_moving	int32
purpose_other	int32
purpose_renewable_energy	int32
purpose_small_business	int32
purpose_vacation	int32
purpose_wedding	int32
emp_length_10+ years	int32
emp_length_2 years	int32
emp_length_3 years	int32
emp_length_4 years	int32
emp_length_5 years	int32
emp_length_6 years	int32
emp_length_7 years	int32
emp_length_8 years	int32
emp_length_9 years	int32
emp_length_< 1 year	int32
sub_grade_encoded	int32
emp_title_encoded	int64

```
title_encoded                 int64
dtype: object
```

In [63]:
```
1 df_cleaned.sample(5)
```

Out[63]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_
84162	7.824446	10.49	9.013878	11.407576	2.836150	4.242641	0.000000	10.715
284167	9.903538	7.49	24.940730	11.925042	3.108614	3.605551	0.000000	11.417
347691	10.257694	24.99	28.919716	11.225257	2.776332	3.162278	0.000000	9.142
56394	8.987322	13.33	16.456913	10.373522	3.648057	3.162278	0.693147	9.841
142721	9.210440	17.77	18.983677	11.722384	2.901422	3.162278	0.000000	9.780

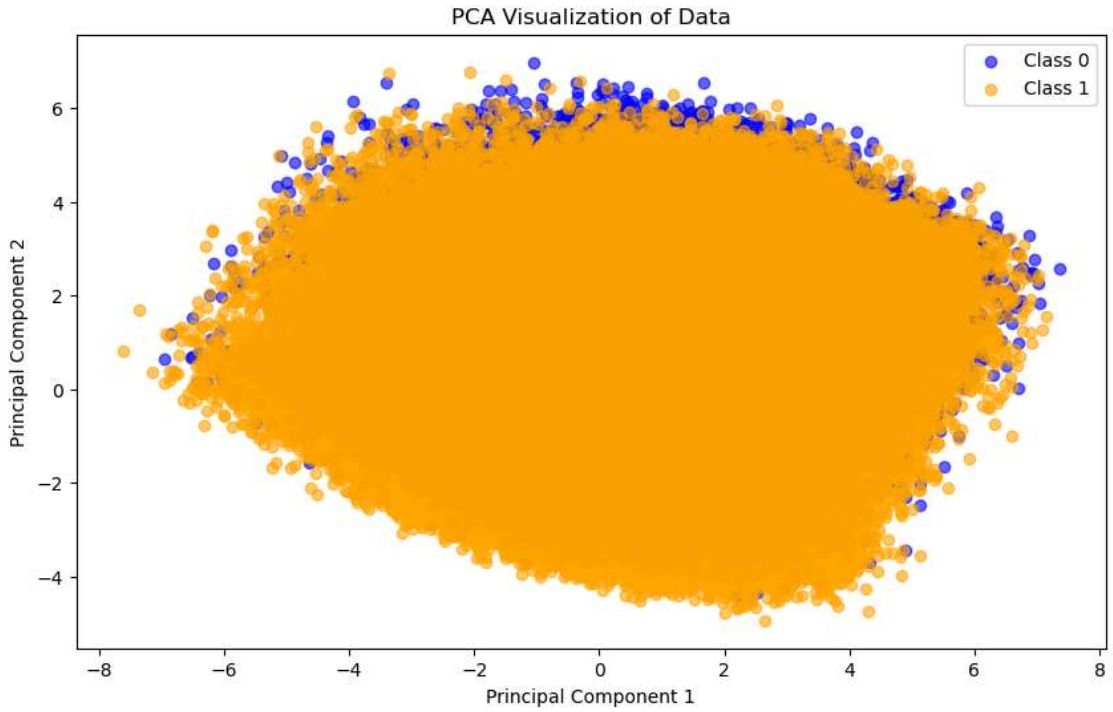
5 rows × 56 columns

==========

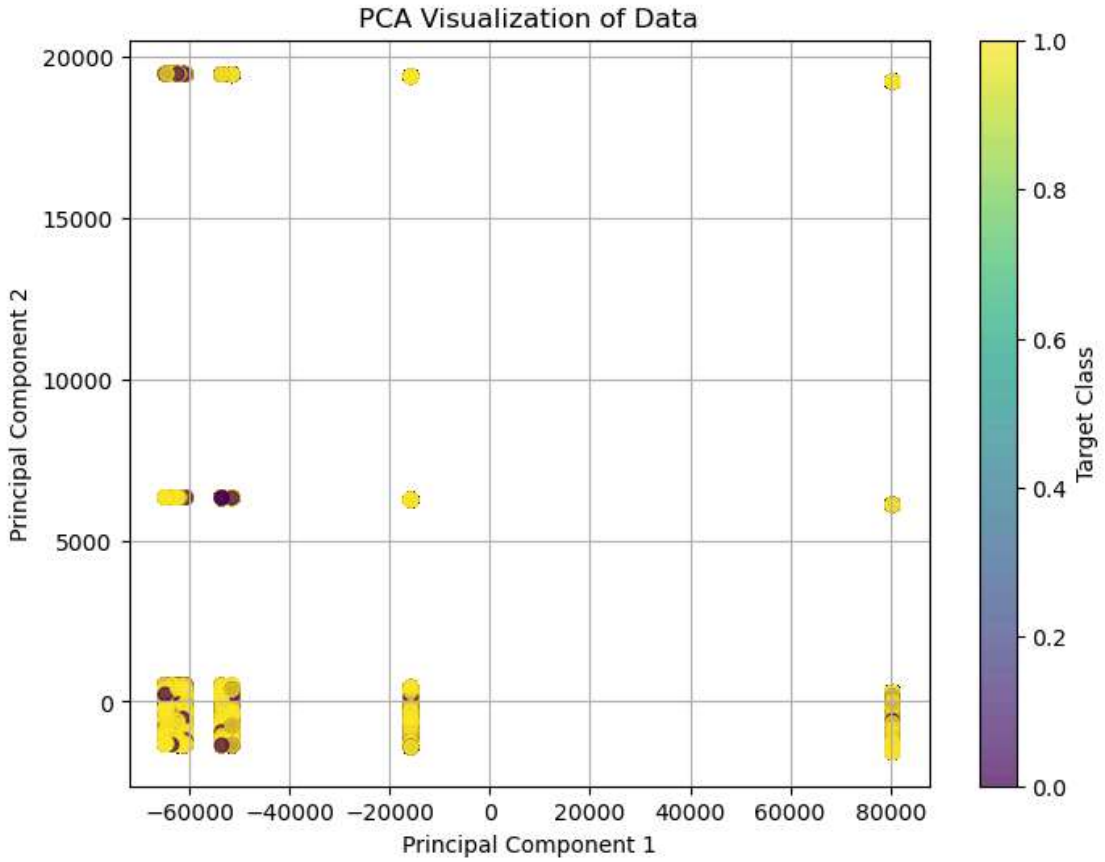
➔ 3. Model building

In [64]:

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Drop the target column and scale the features
X = df_cleaned.drop(columns=['loan_status_Fully Paid'])
y = df_cleaned['loan_status_Fully Paid']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(10, 6))
plt.scatter(X_pca[y == 0, 0], X_pca[y == 0, 1], alpha=0.6, label='Cla
plt.scatter(X_pca[y == 1, 0], X_pca[y == 1, 1], alpha=0.6, label='Cla

plt.title('PCA Visualization of Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



In [65]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
visualize = pca.fit_transform(X)

def scatter_visualize(visualize, target):
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(visualize[:, 0], visualize[:, 1], c=target,
    plt.colorbar(scatter, label='Target Class')
    plt.title('PCA Visualization of Data')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.grid(True)
    plt.show()

scatter_visualize(visualize, y)
```



In [66]:
```
df_cleaned['loan_status_Fully Paid'].value_counts()
```

Out[66]:
```
loan_status_Fully Paid
1    297728
0     72829
Name: count, dtype: int64
```

In [67]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [68]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_sca
from collections import Counter
print("Before SMOTE:", Counter(y_train))
print("After SMOTE:", Counter(y_train_resampled))
```

```
Before SMOTE: Counter({1: 238182, 0: 58263})
After SMOTE: Counter({1: 238182, 0: 238182})
```

In [69]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

rfe = RFE(estimator=model, n_features_to_select=20)  # Select the top
X_train_selected = rfe.fit_transform(X_train_resampled, y_train_resam

selected_features = rfe.support_
print("Selected Features: ", selected_features)

X_test_selected = rfe.transform(X_test_scaled)
```

```
Selected Features:  [ True  True  True  True  True  True False  True  Tr
ue  True False False
  True  True  True False False False False False False False  True False
 False False False False  True False False False  True False False False
 False  True False  True False False False False False False False False
 False False False False  True  True  True]
```

In [70]:
```
model.fit(X_train_selected, y_train_resampled)
```

Out[70]:
```
LogisticRegression(https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.Logis
LogisticRegression()
```

In [71]:
```
y_pred = model.predict(X_test_selected)

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.31      0.66      0.43     14566
           1       0.89      0.65      0.75     59546

    accuracy                           0.65     74112
   macro avg       0.60      0.65      0.59     74112
weighted avg       0.77      0.65      0.68     74112
```

In [72]:
```
feature_names = X.columns
selected_feature_names = feature_names[rfe.support_]

coefficients = model.coef_[0]

coefficients_df = pd.DataFrame({
    'Feature': selected_feature_names,
    'Coefficient': coefficients
})

coefficients_df = coefficients_df.sort_values(by='Coefficient', key=a

print(coefficients_df)
```

```
                    Feature  Coefficient
17         sub_grade_encoded    -0.768521
0                  loan_amnt    -0.501306
2                installment     0.297446
3                 annual_inc     0.274388
1                   int_rate     0.221586
5                   open_acc    -0.210705
4                        dti    -0.203489
8                  total_acc     0.158643
10                   grade_C    -0.149826
7                 revol_util    -0.147910
9                    grade_B    -0.125599
12       home_ownership_RENT    -0.116720
19             title_encoded    -0.101696
11                   grade_D    -0.089126
13           term_ 60 months    -0.081052
16    purpose_small_business    -0.077797
18         emp_title_encoded    -0.076978
6                  revol_bal     0.076745
14  purpose_home_improvement    -0.070124
15             purpose_other    -0.054591
```

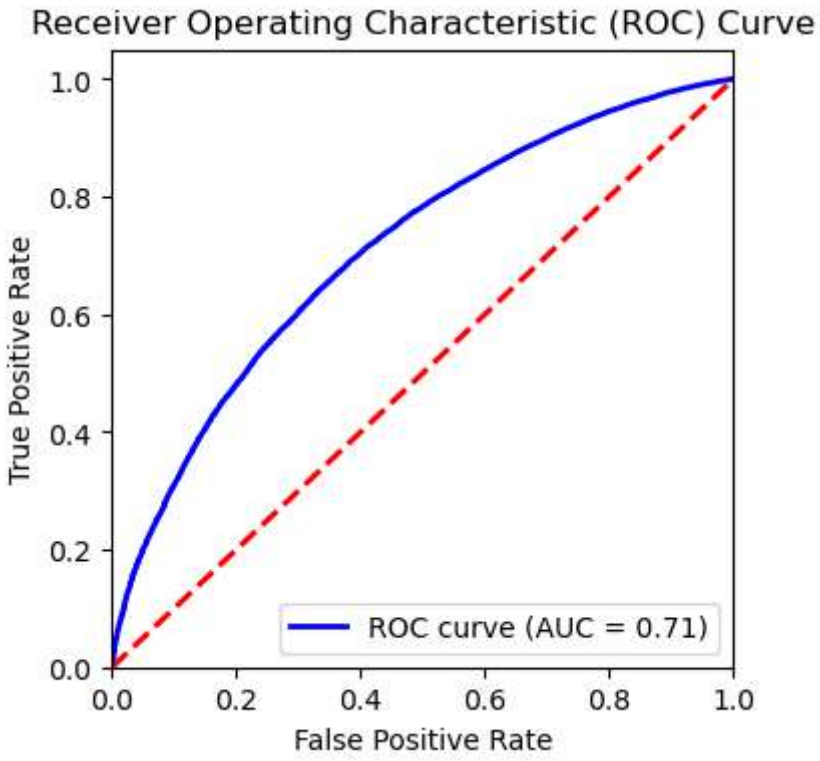
==========

➔ 4. Results Evaluation

1. ROC AUC Curve & comments

In [77]:

```
X_test_selected = rfe.transform(X_test_scaled)

y_prob = model.predict_proba(X_test_selected)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_prob)

roc_auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(4, 4))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_
plt.plot([0, 1], [0, 1], color='red', linestyle='--', lw=2)  # Diagon
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Comment on the ROC Curve :-

➞ A model with an AUC of 0.65-0.7 is generally considered to have moderate performance. If your AUC is closer to 1, it would indicate a better ability of the model to distinguish between the two classes.

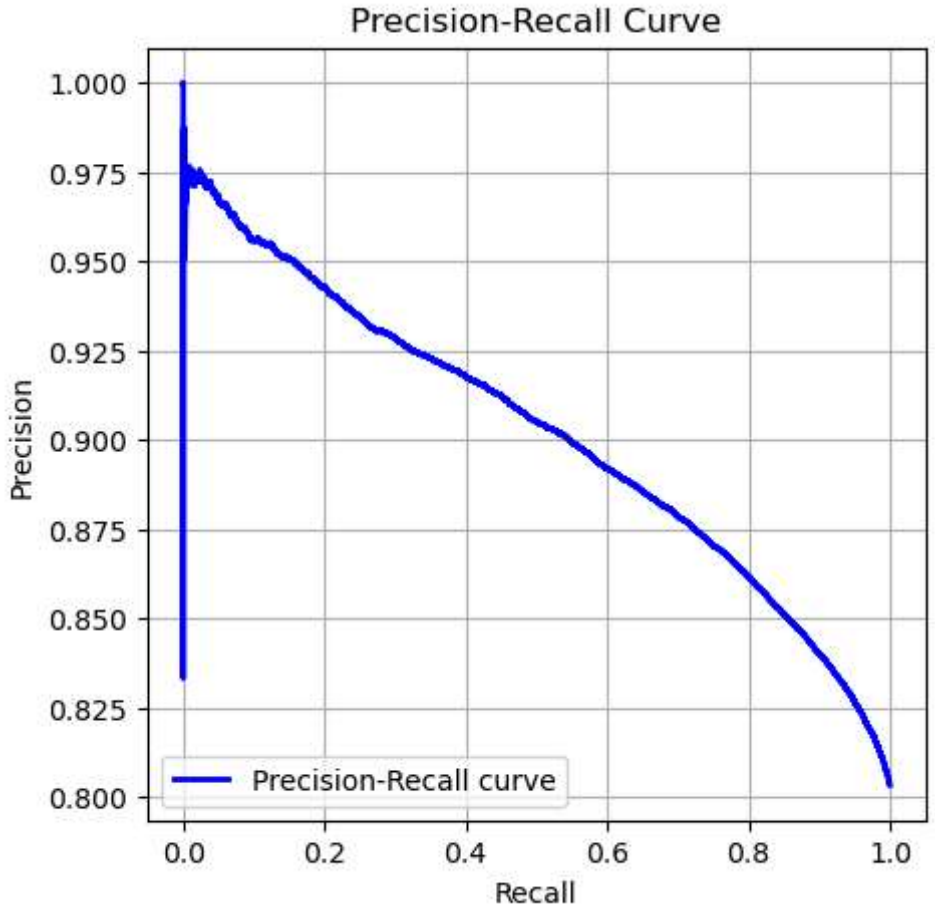
➞ The ROC curve should generally bow upwards, indicating a model that performs better than random chance. If the curve is near the diagonal line (red dashed line), the model has poor discriminative ability.

➞ As we can see our AUC = 0.71 and the ROC curve is also bow upwards which means our model has moderate performance not good .

2. Precision Recall Curve & comments

In [79]:

```
from sklearn.metrics import precision_recall_curve, average_precision
import matplotlib.pyplot as plt

X_test_selected = rfe.transform(X_test_scaled)

y_prob = model.predict_proba(X_test_selected)[:, 1]

precision, recall, thresholds = precision_recall_curve(y_test, y_prob

plt.figure(figsize=(5, 5))
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Reca
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.grid(True)
plt.show()

pr_auc = average_precision_score(y_test, y_prob)
print(f'Average Precision (PR AUC): {pr_auc:.2f}')
```




```
Average Precision (PR AUC): 0.90
```

Comments on the Results :-

➞ Precision-Recall Curve :

Based on your model's results, if the curve is near the top-right corner, it means your model is able to achieve both high precision and high recall. This is good for imbalanced datasets where the positive class (class 1) is more important.

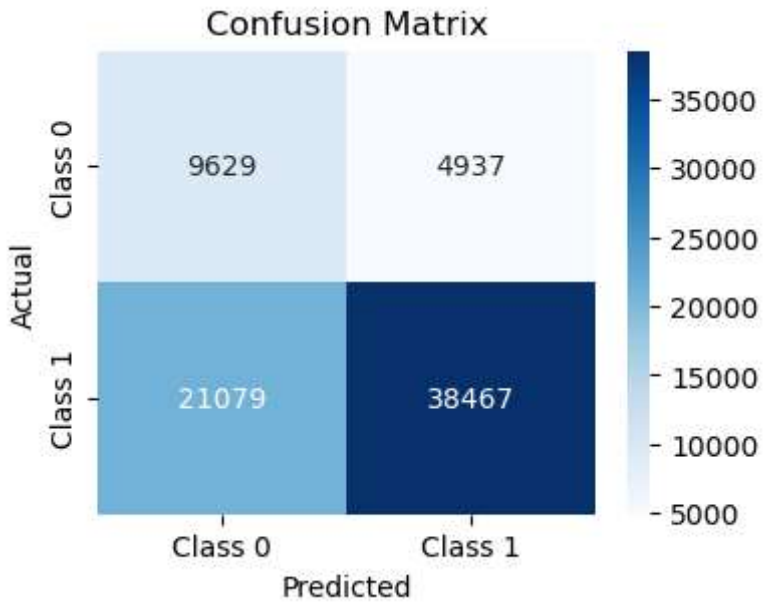
➞ PR AUC :

A PR AUC score of 0.7 or above is considered good. The closer the score is to 1, the better the model is at distinguishing between positive and negative classes with respect to both precision and recall.In our case (PR AUC) is 0.90 which means good

3. Classification Report (Confusion Matrix etc)

In [80]:
```
from sklearn.metrics import classification_report, confusion_matrix

report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Clas
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.31      0.66      0.43     14566
           1       0.89      0.65      0.75     59546

    accuracy                           0.65     74112
   macro avg       0.60      0.65      0.59     74112
weighted avg       0.77      0.65      0.68     74112
```



4. Tradeoff Questions

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

This can be achieved through :-

➞ Adjusting decision threshold : Lowering the threshold for classifying a defaulter may increase recall but may decrease precision. You can use techniques like ROC curves or Precision-Recall curves to find the best threshold.

➞ Optimizing the F1-score : The current F1-score for class 1 is 0.75, which is decent but can be improved by increasing recall.

➞ Additional resampling techniques like undersampling the majority class or applying SMOTE-EN (SMOTE with edited nearest neighbors) could further help reduce false positives by making the model better at distinguishing between the classes.

➞ The current model may benefit from hyperparameter tuning using cross-validation techniques like GridSearchCV or RandomizedSearchCV. Consider using algorithms like Random Forest or XGBoost, which are better suited for handling class imbalances and may offer improved recall.

➞ Incorporating Domain Knowledge.Feature engineering and selection can help refine the model to focus on the most predictive features, further reducing false positives

```
b) Since NPA (non-performing asset) is a real problem in this industry, it’s important we play safe and shouldn’t disburse loans to anyone
```

➞ Increase Recall for Defaulters (Class 1): Ensure more real defaulters are detected to minimize NPAs.

➞ Adjust Decision Threshold: Lower the threshold to classify defaulters more conservatively.

➞ Focus on F1-Score Optimization: Balance precision and recall to ensure safe decisions.

➞ Use Resampling Techniques: Improve model balance with advanced resampling like SMOTE-EN.

➞ Explore Robust Models: Use algorithms like Random Forest or XGBoost for better NPA detection.

➞ Incorporate Domain Knowledge: Prioritize features like debt-to-income ratio and credit history.

➞ Regular Model Monitoring: Continuously update the model with new NPA-related data.

==========

Actionable Insights :-

1. Imbalanced Data Handling: The model's recall for defaulters can be improved by using advanced techniques like SMOTE or SMOTE-ENN for better class balance.
2. Threshold Adjustment: Lowering the classification threshold can reduce the risk of misclassifying defaulters, ensuring fewer NPAs.

3. Key Feature Focus: Features like dti, loan_amount, and credit history should be monitored and weighted more heavily in decision-making.
4. Improved Data Collection: Gather more granular data about borrower behavior, repayment history, and external economic factors.
5. Use Ensemble Models: Algorithms like Random Forest and Gradient Boosting can capture complex patterns and improve prediction.

==========

Recommendations :-

1. Policy Adaptation: Implement stricter loan approval policies for high-risk profiles as flagged by the model.
2. Regular Model Training: Continuously train the model with new data to adapt to changing borrower behavior.
3. Business Rules Integration: Supplement the model with domain-specific rules (e.g., exclude borrowers with a high debt-to-income ratio).
4. Pilot Testing: Run the updated model on historical data to validate its effectiveness in identifying defaulters.
5. Performance Monitoring: Track key metrics like recall and F1-score for defaulters over time and adjust as needed.

==========