In [4]:
```
import numpy as np,pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusi
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [5]:
```
df = pd.read_csv("logistic_regression.csv")
```

In [6]:
```
df.head()
```

Out[6]:

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

5 rows × 27 columns

In [7]:
```
df.isna().sum()
```

Out[7]:
```
loan_amnt                 0
term                      0
int_rate                  0
installment               0
grade                     0
sub_grade                 0
emp_title             22927
emp_length            18301
home_ownership            0
annual_inc                0
verification_status       0
issue_d                   0
loan_status               0
purpose                   0
title                  1756
dti                       0
earliest_cr_line          0
open_acc                  0
pub_rec                   0
revol_bal                 0
revol_util              276
total_acc                 0
initial_list_status       0
application_type          0
mort_acc              37795
pub_rec_bankruptcies    535
address                   0
dtype: int64
```

In [8]:
```
df.head()
```

Out[8]:

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

5 rows × 27 columns

Insight:

There are lot of null values found in some columns.

In [9]:
```
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 [10]:
```
df.describe()
```

Out[10]:

	loan_amnt	int_rate	installment	annual_inc	dti	ope
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.0(
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.3
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.1
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.0(
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.0(
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.0(
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.0(
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.0(

In [11]:
```
df.shape
```

Out[11]:
```
(396030, 27)
```

In [12]:
```
df.isnull().sum()/len(df)*100
```

Out[12]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443401
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

In [13]:
```
df.nunique()
```

Out[13]:
```
loan_amnt               1397
term                       2
int_rate                 566
installment            55706
grade                      7
sub_grade                 35
emp_title             173105
emp_length                11
home_ownership             6
annual_inc             27197
verification_status        3
issue_d                  115
loan_status                2
purpose                   14
title                  48816
dti                     4262
earliest_cr_line         684
open_acc                  61
pub_rec                   20
revol_bal              55622
revol_util              1226
total_acc                118
initial_list_status        2
application_type           3
mort_acc                  33
pub_rec_bankruptcies       9
address               393700
dtype: int64
```

In [14]:
```
df['term'].value_counts()
```

Out[14]:
```
term
 36 months    302005
 60 months     94025
Name: count, dtype: int64
```

In [15]:
```
df['term'] = df['term'].str[:3]
```

In [16]:
```
emp_length_map = {
    '10+ years': 10,
    '9 years': 9,
    '8 years': 8,
    '7 years': 7,
    '6 years': 6,
    '5 years': 5,
    '4 years': 4,
    '3 years': 3,
    '2 years': 2,
    '1 year': 1,
    '< 1 year': 0.11
}

# Replace the values in the emp_length column
df['emp_length'] = df['emp_length'].replace(emp_length_map)
```

In [17]:
```
df['emp_length'].value_counts()
```

Out[17]:
```
emp_length
10.00    126041
2.00      35827
0.11      31725
3.00      31665
5.00      26495
1.00      25882
4.00      23952
6.00      20841
7.00      20819
8.00      19168
9.00      15314
Name: count, dtype: int64
```

Treating the Missing Values

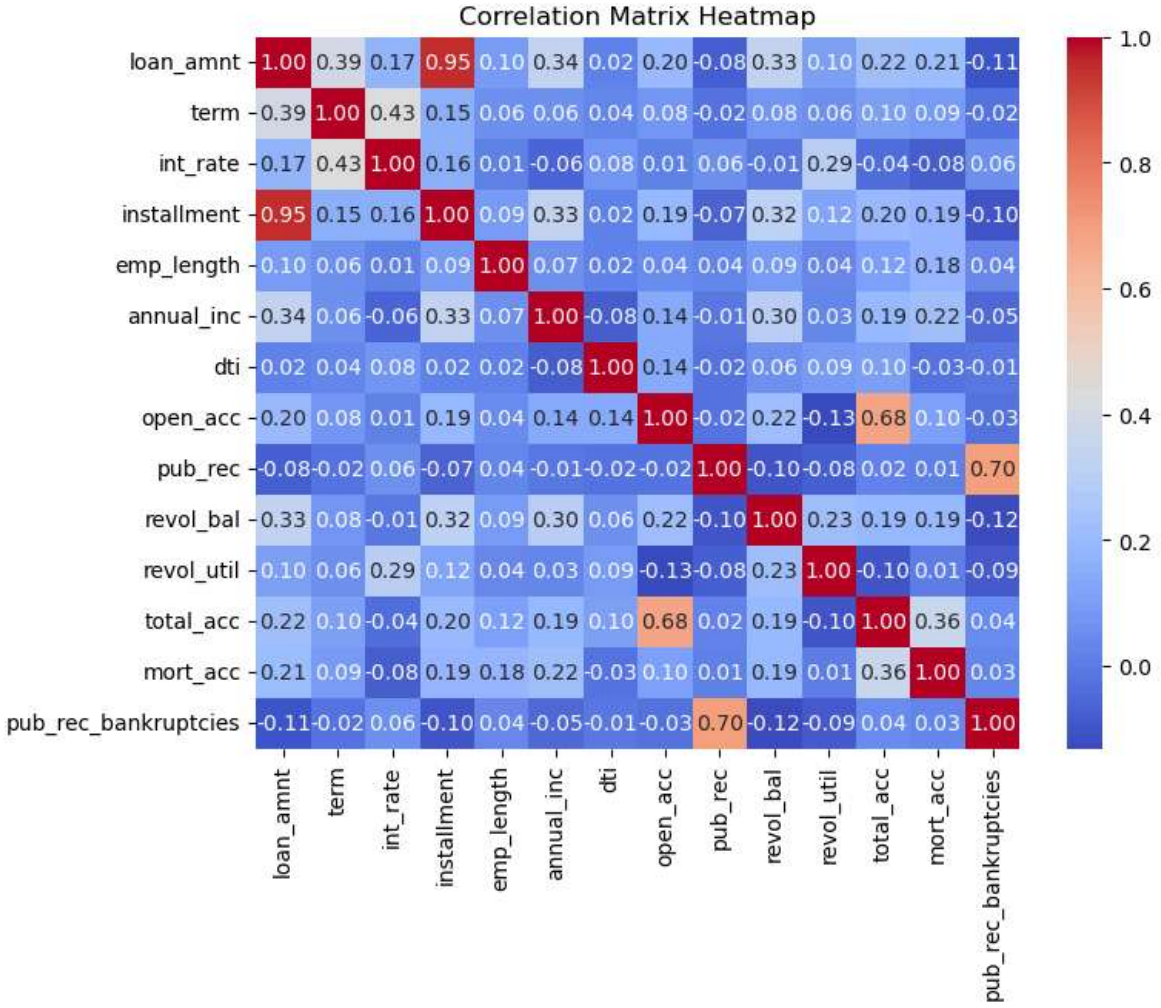
In [18]:
```
df['emp_length'].fillna(df['emp_length'].mean(), inplace=True)
df['revol_util'].fillna(df['revol_util'].mean(), inplace=True)
df['mort_acc'].fillna(df['mort_acc'].mean(), inplace=True)
df['pub_rec_bankruptcies'].fillna(df['pub_rec_bankruptcies'].mean(), inpla
```

In [19]:
```
df['emp_title'].fillna(df['emp_title'].mode()[0], inplace=True)
df['title'].fillna(df['title'].mode()[0], inplace=True)
```

In [20]:
```
df.isna().sum()
```

Out[20]:
```
loan_amnt               0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
emp_title               0
emp_length              0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
title                   0
dti                     0
earliest_cr_line        0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
initial_list_status     0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
address                 0
dtype: int64
```

In [21]:
```
# Creating a heatmap

arr = df.copy()

arr.drop(columns=['grade','sub_grade','emp_title','verification_status','l
                  'initial_list_status','home_ownership','issue_d','applic
                  'address','title','earliest_cr_line','purpose'], inplace

corr_matrix = arr.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



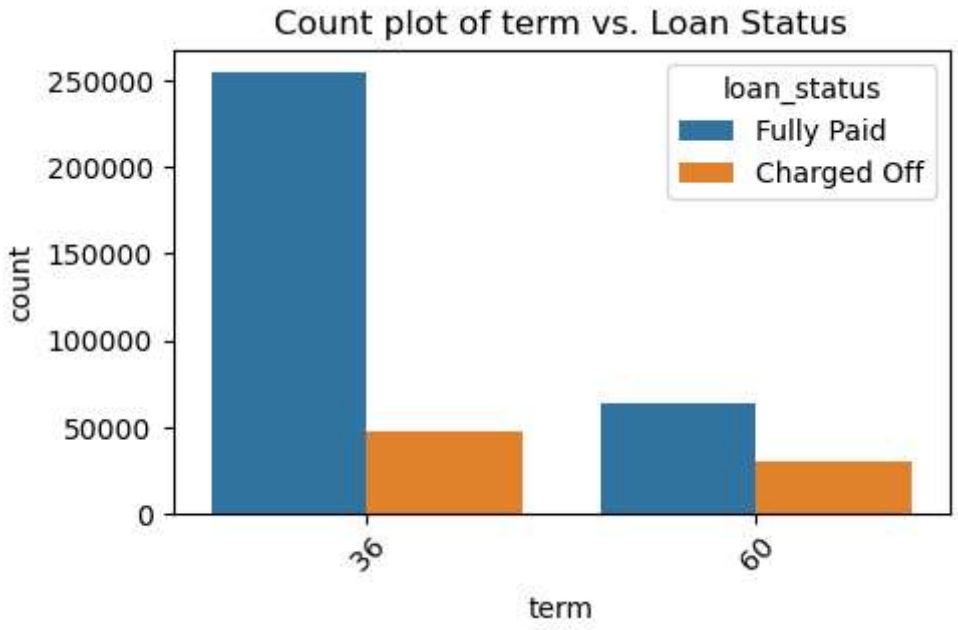
Insight:

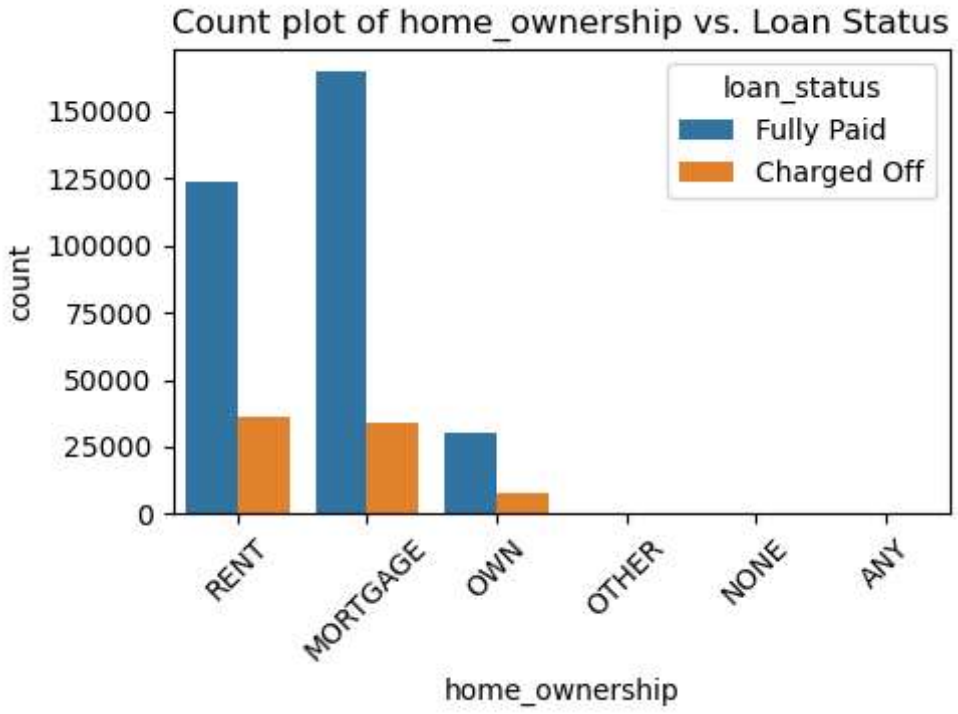
1.pub_rec_bankruptcies have a relatively positive co-relation with pub_rec.

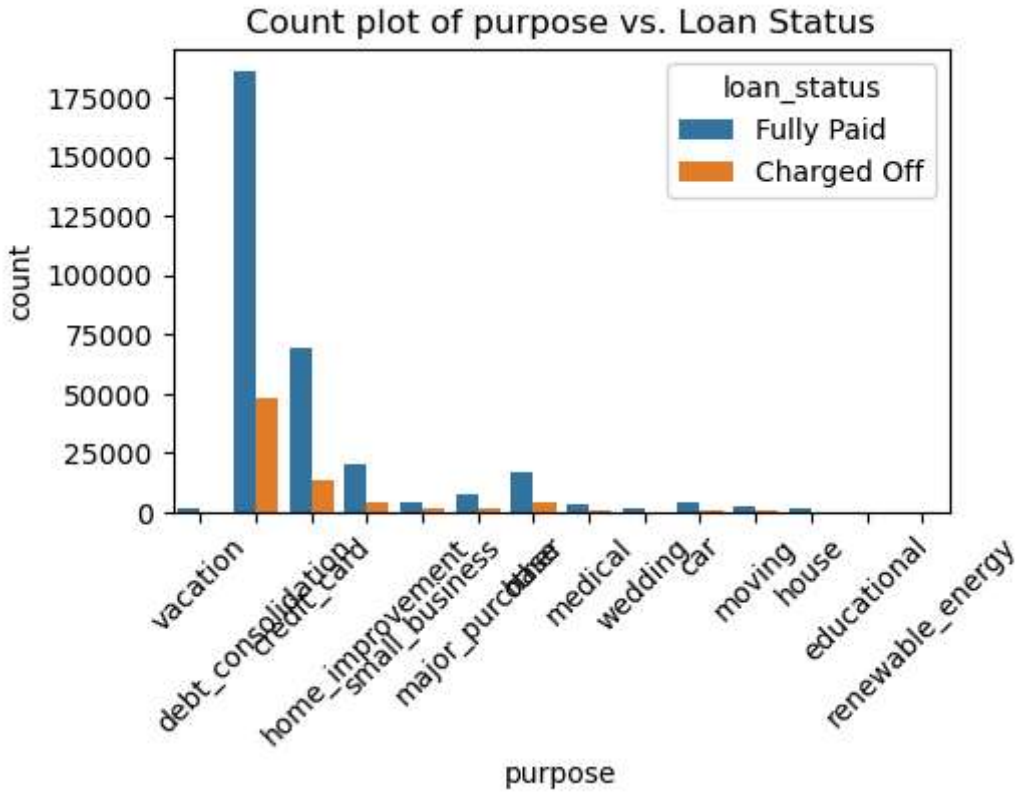
2.Understandably open_acc has a postive co-relation with total_acc.

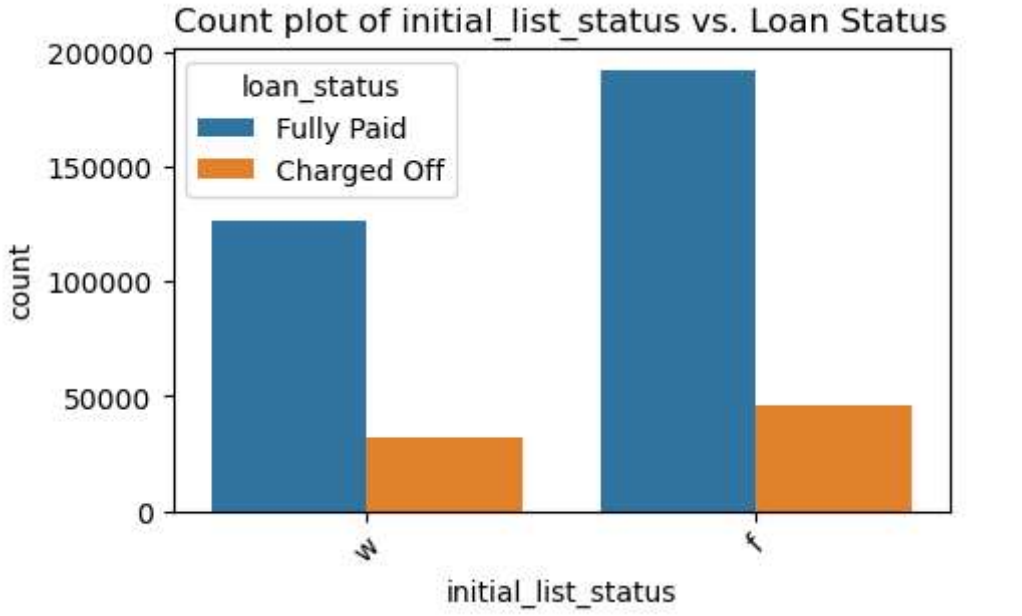
3.annual_income and revol_bal has a positive co-relation.

In [22]:
```
categorical_features = [ 'term', 'home_ownership', 'purpose',
                         'initial_list_status']


for feature in categorical_features:
    plt.figure(figsize=(5, 3))
    sns.countplot(data=df, x=feature, hue='loan_status')
    plt.title(f'Count plot of {feature} vs. Loan Status')
    plt.xticks(rotation=45)
    plt.show()
```





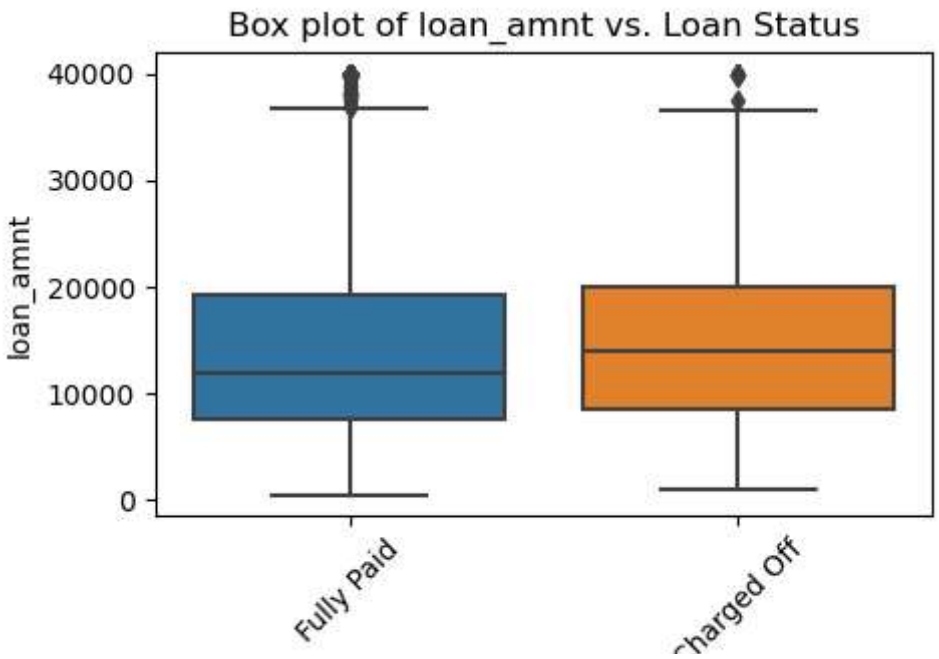




Insight:

1.36 months loans are paid fully more than 60 months showing that maybe shorter loans are paid more easily.
2.people who are on Rent or Mortgage tend to pay their loan full.
3. Vacation and debt consolidation are the most frequent reasons f or loans among users.

In [23]:
```
numerical_features = ['loan_amnt', 'int_rate', 'installment', 'annual_inc'
                      'dti', 'open_acc', 'revol_util', 'revol_bal',
                      'total_acc', 'mort_acc','emp_length','pub_rec_bankru

for feature in numerical_features:
    plt.figure(figsize=(5, 3))
    sns.boxplot(data=df, x='loan_status', y=feature)
    plt.title(f'Box plot of {feature} vs. Loan Status')
    plt.xticks(rotation=45)
    plt.show()
```



Insight:

There are lots of outliers present in all the columns we will use IQR method to solve this issue.

In [24]:
```
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()


Q1 = df[numeric_cols].quantile(0.25)
Q3 = df[numeric_cols].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df = df[~((df[numeric_cols] < lower_bound) | (df[numeric_cols] > upper_bou
```

In [25]:
```
print(df)
```

```
        loan_amnt term  int_rate  installment grade sub_grade  \
0         10000.0   36     11.44       329.48     B        B4
1          8000.0   36     11.99       265.68     B        B5
2         15600.0   36     10.49       506.97     B        B3
3          7200.0   36      6.49       220.65     A        A2
4         24375.0   60     17.27       609.33     C        C5
...           ...  ...       ...          ...   ...       ...
396024     6000.0   36     13.11       202.49     B        B4
396025    10000.0   60     10.99       217.38     B        B4
396027     5000.0   36      9.99       161.32     B        B1
396028    21000.0   60     15.31       503.02     C        C2
396029     2000.0   36     13.61        67.98     C        C2

                     emp_title  emp_length home_ownership  annual_inc
...  \
0                    Marketing       10.00           RENT    117000.0
...
1                Credit analyst         4.00       MORTGAGE     65000.0
...
2                   Statistician        0.11           RENT     43057.0
...
3                Client Advocate        6.00           RENT     54000.0
...
4         Destiny Management Inc.        9.00       MORTGAGE     55000.0
...
...                          ...         ...            ...         ...
...
396024   Michael's Arts & Crafts        5.00           RENT     64000.0
...
396025          licensed bankere        2.00           RENT     40000.0
...
396027              City Carrier       10.00           RENT     56500.0
...
396028      Gracon Services, Inc       10.00       MORTGAGE     64000.0
...
396029  Internal Revenue Service       10.00           RENT     42996.0
...

       open_acc pub_rec revol_bal revol_util total_acc  initial_list_stat
us  \
0          16.0     0.0   36369.0       41.8      25.0
w
1          17.0     0.0   20131.0       53.3      27.0
f
2          13.0     0.0   11987.0       92.2      26.0
f
3           6.0     0.0    5472.0       21.5      13.0
f
4          13.0     0.0   24584.0       69.8      43.0
f
...         ...     ...       ...        ...       ...
...
396024      7.0     0.0   11456.0       97.1       9.0
w
396025      6.0     0.0    1990.0       34.3      23.0
w
396027     15.0     0.0   32704.0       66.9      23.0
```

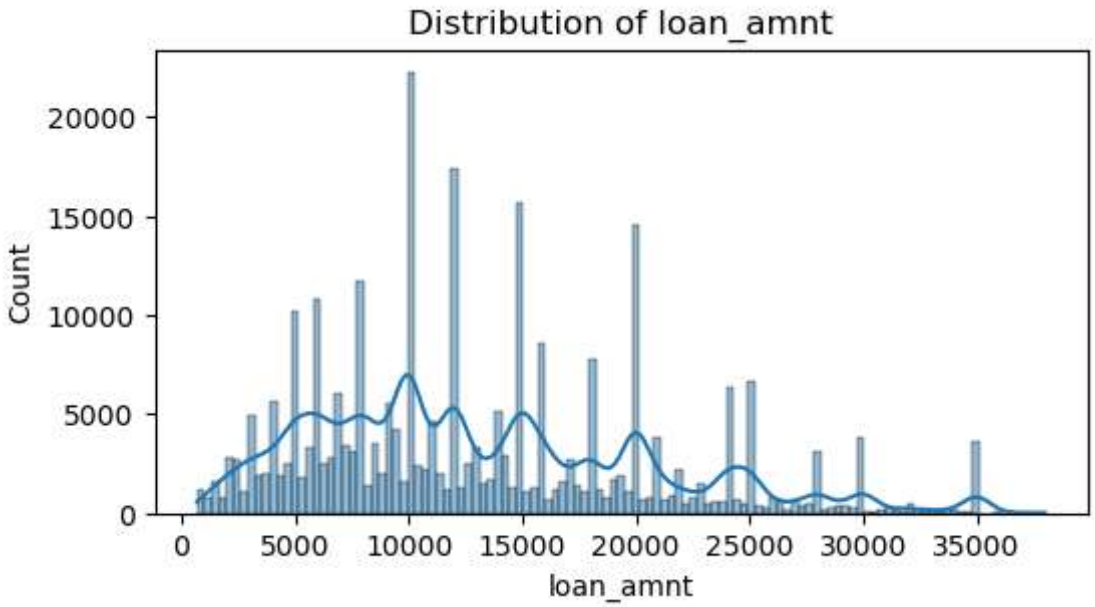
```
f
396028      9.0     0.0   15704.0       53.8      20.0
f
396029      3.0     0.0    4292.0       91.3      19.0
f

       application_type  mort_acc  pub_rec_bankruptcies  \
0            INDIVIDUAL  0.000000                   0.0
1            INDIVIDUAL  3.000000                   0.0
2            INDIVIDUAL  0.000000                   0.0
3            INDIVIDUAL  0.000000                   0.0
4            INDIVIDUAL  1.000000                   0.0
...                 ...       ...                   ...
396024       INDIVIDUAL  0.000000                   0.0
396025       INDIVIDUAL  0.000000                   0.0
396027       INDIVIDUAL  0.000000                   0.0
396028       INDIVIDUAL  5.000000                   0.0
396029       INDIVIDUAL  1.813991                   0.0

                                                address
0          0174 Michelle Gateway\r\nMendozaberg, OK 22690
1       1076 Carney Fort Apt. 347\r\nLoganmouth, SD 05113
2       87025 Mark Dale Apt. 269\r\nNew Sabrina, WV 05113
3                 823 Reid Ford\r\nDelacruzside, MA 00813
4                  679 Luna Roads\r\nGreggshire, VA 11650
...                                                  ...
396024  514 Cynthia Park Apt. 402\r\nWest Williamside,...
396025   12951 Williams Crossing\r\nJohnnyville, DC 30723
396027  953 Matthew Points Suite 414\r\nReedfort, NY 7...
396028  7843 Blake Freeway Apt. 229\r\nNew Michael, FL...
396029      787 Michelle Causeway\r\nBriannaton, AR 48052

[284402 rows x 27 columns]
```

In [26]:

```
num_duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {num_duplicates}")
```

```
Number of duplicate rows: 0
```

In [27]:
```
numerical_cols = ['loan_amnt', 'int_rate', 'installment', 'annual_inc',
                      'dti', 'open_acc', 'revol_util', 'revol_bal',
                      'total_acc', 'mort_acc','emp_length','pub_rec_bankru

for i in numerical_cols:
    plt.figure(figsize=(6, 3))
    sns.histplot(data=df, x=i, kde=True)
    plt.title(f'Distribution of {i}')
    plt.show()
```



Insight:

1.revol_bal is right skewed distribution.
2.dti and revol_util is somewhat normally distributed.
3.installment is rightly skewed.
4. Others are not distributed normally.

In [28]:
```
#Extracting the month and year from earliest_cr_line

df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'], errors = '

df['month'] = df['earliest_cr_line'].dt.month

df['year'] = df['earliest_cr_line'].dt.year
```

```
C:\Users\avesh\AppData\Local\Temp\ipykernel_1548\1529716156.py:3: UserWar
ning: Could not infer format, so each element will be parsed individuall
y, falling back to `dateutil`. To ensure parsing is consistent and as-exp
ected, please specify a format.
  df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'], errors
= 'coerce')
```

In [ ]:

```
```

In [29]:

```
df['region'] = df['address'].str.split().str[-2]
```

In [30]:

```
# Flag for Pub_rec_flag, Mort_acc, Pub_rec_bankruptcies.

df['pub_rec_flag'] = df['pub_rec'].apply(lambda x: 1 if x > 0 else 0)

df['mort_acc_flag'] = df['mort_acc'].apply(lambda x: 1 if x > 0 else 0)

df['pub_rec_bankruptcies_flag'] = df['pub_rec_bankruptcies'].apply(lambda

print(df[['pub_rec', 'pub_rec_flag', 'mort_acc', 'mort_acc_flag', 'pub_rec
```

```
   pub_rec  pub_rec_flag  mort_acc  mort_acc_flag  pub_rec_bankruptcies
\
0      0.0             0       0.0              0                   0.0
1      0.0             0       3.0              1                   0.0
2      0.0             0       0.0              0                   0.0
3      0.0             0       0.0              0                   0.0
4      0.0             0       1.0              1                   0.0

   pub_rec_bankruptcies_flag
0                          0
1                          0
2                          0
3                          0
4                          0
```

In [31]:

```
df.drop(columns=('address'), inplace = True)
df.drop(columns=('pub_rec_bankruptcies'), inplace = True)
df.drop(columns=('mort_acc'), inplace = True)
df.drop(columns=('pub_rec'), inplace = True)
df.drop(columns=['earliest_cr_line'],inplace = True)
```

In [32]:
```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\avesh\anacon
da3\lib\site-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in c:\users\avesh\anaconda3
\lib\site-packages (from category_encoders) (1.24.3)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\avesh\ana
conda3\lib\site-packages (from category_encoders) (1.5.1)
Requirement already satisfied: scipy>=1.0.0 in c:\users\avesh\anaconda3\l
ib\site-packages (from category_encoders) (1.11.1)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\avesh\anaco
nda3\lib\site-packages (from category_encoders) (0.14.0)
Requirement already satisfied: pandas>=1.0.5 in c:\users\avesh\anaconda3
\lib\site-packages (from category_encoders) (2.0.3)
Requirement already satisfied: patsy>=0.5.1 in c:\users\avesh\anaconda3\l
ib\site-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\avesh\a
naconda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2.8.
2)
Requirement already satisfied: pytz>=2020.1 in c:\users\avesh\anaconda3\l
ib\site-packages (from pandas>=1.0.5->category_encoders) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\avesh\anaconda3
\lib\site-packages (from pandas>=1.0.5->category_encoders) (2023.3)
Requirement already satisfied: six in c:\users\avesh\anaconda3\lib\site-p
ackages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in c:\users\avesh\anaconda3
\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\avesh\ana
conda3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders)
(3.5.0)
Requirement already satisfied: packaging>=21.3 in c:\users\avesh\anaconda
3\lib\site-packages (from statsmodels>=0.9.0->category_encoders) (23.1)
```

In [33]:
```
import category_encoders as ce
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from category_encoders import TargetEncoder

categorical_cols = ['term', 'grade', 'sub_grade', 'home_ownership',
                    'verification_status', 'issue_d', 'purpose',
                    'initial_list_status', 'application_type', 'region']

numerical_cols = ['loan_amnt', 'int_rate', 'installment', 'emp_length', 'a
                  'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mon
                  'pub_rec_flag', 'mort_acc_flag', 'pub_rec_bankruptcies_f



df_filtered = df[categorical_cols + numerical_cols + ['loan_status']]

cols = {'Fully Paid': 0, 'Charged Off':1}

df_filtered['loan_status'] = df_filtered['loan_status'].map(cols)

X = df_filtered.drop(columns=['loan_status'])  # Features (excluding the t
y = df_filtered['loan_status']  # Target variable


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Apply Target Encoding to categorical columns
target_encoder = TargetEncoder(cols=categorical_cols)
X_train_encoded = target_encoder.fit_transform(X_train, y_train)
X_test_encoded = target_encoder.transform(X_test)

# Apply scaling to numerical columns
scaler = StandardScaler()

arr_train = scaler.fit_transform(X_train_encoded[numerical_cols])
arr_test = scaler.transform(X_test_encoded[numerical_cols])
X_train_scaled = pd.DataFrame(arr_train,columns=numerical_cols, index=X_tr
X_test_scaled = pd.DataFrame(arr_test, columns=numerical_cols, index=X_tes

# Concatenate scaled numerical data with the remaining encoded categorical
X_train_final = pd.concat([X_train_scaled, X_train_encoded.drop(columns=nu
X_test_final = pd.concat([X_test_scaled, X_test_encoded.drop(columns=numer
```

```
C:\Users\avesh\AppData\Local\Temp\ipykernel_1548\4201719986.py:21: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_filtered['loan_status'] = df_filtered['loan_status'].map(cols)
```

In [34]:
```
X_train_final.head()
```

Out[34]:

	loan_amnt	int_rate	installment	emp_length	annual_inc	dti	open_acc	revo
305080	-0.152217	0.573650	0.103213	0.062516	1.247842	-0.870429	0.325897	-0.28
266843	-1.498665	-1.595457	-1.596246	-0.784196	0.015116	-0.680639	-0.371334	-1.28
26276	-0.559517	1.640878	-0.299776	1.191465	0.543427	0.006099	-0.371334	-0.36
259525	0.924942	-0.431208	0.192840	1.191465	2.020832	-0.162464	0.790718	1.59
29380	1.497183	0.441979	2.101161	0.909228	2.216413	-0.952837	0.558307	2.10

5 rows × 25 columns

In [35]:
```
X_test_final.head()
```

Out[35]:

	loan_amnt	int_rate	installment	emp_length	annual_inc	dti	open_acc	revo
86593	-0.744654	-0.255647	-0.688646	-1.599862	-0.407534	1.533154	2.185179	-0.82
348752	-0.562883	0.289517	-0.427794	-0.219722	-0.643513	-0.304806	1.487948	-0.73
285761	0.251718	-1.334425	0.321613	0.344753	2.445348	0.110982	0.325897	2.52
44259	2.271390	-0.163246	1.315251	-1.066434	0.543427	-0.413436	0.558307	-0.23
258316	2.537314	-1.496126	2.771000	-1.599862	2.832776	-0.547037	0.558307	-0.49

5 rows × 25 columns

In [36]:
```
df = df_filtered
```

Model Building

In [37]:

```
# Initialize the logistic regression model
model = LogisticRegression(max_iter=1000, random_state=42)

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

# Make predictions
y_pred = model.predict(X_test_final)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.81
```

In [38]:
```
x = df[numerical_cols]
vif_data = pd.DataFrame()
vif_data['feature'] = x.columns
vif_data['VIF'] = [variance_inflation_factor (x.values, i) for i in range(

print(vif_data)
```

```
C:\Users\avesh\anaconda3\Lib\site-packages\statsmodels\regression\linear_
model.py:1783: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.uncentered_tss

                      feature        VIF
0                   loan_amnt  45.625979
1                    int_rate  13.572356
2                 installment  47.691474
3                  emp_length   3.881996
4                  annual_inc   9.782015
5                         dti   8.008916
6                    open_acc  14.323047
7                   revol_bal   5.777238
8                   revol_util   9.462281
9                   total_acc  12.418297
10                      month   4.903451
11                       year  32.810813
12              pub_rec_flag        NaN
13             mort_acc_flag   3.050562
14  pub_rec_bankruptcies_flag        NaN
```

In [39]:
```
x = x.drop(columns=['installment'])

# Recalculate the VIF values after removing the feature
vif_data = pd.DataFrame()
vif_data['feature'] = x.columns
vif_data['VIF'] = [variance_inflation_factor(x.values, i) for i in range(l

print("VIF Data after removing 'int_rate':")
print(vif_data)
```

```
C:\Users\avesh\anaconda3\Lib\site-packages\statsmodels\regression\linear_
model.py:1783: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.uncentered_tss

VIF Data after removing 'int_rate':
                      feature        VIF
0                   loan_amnt   6.235608
1                    int_rate  13.505666
2                  emp_length   3.881995
3                  annual_inc   9.779897
4                         dti   8.008027
5                    open_acc  14.276533
6                   revol_bal   5.774884
7                  revol_util   9.306702
8                   total_acc  12.401497
9                       month   4.903306
10                       year  32.426036
11               pub_rec_flag         NaN
12              mort_acc_flag   3.018070
13  pub_rec_bankruptcies_flag         NaN
```

In [40]:
```
x = x.drop(columns=['year'])

# Recalculate the VIF values after removing the feature
vif_data = pd.DataFrame()
vif_data['feature'] = x.columns
vif_data['VIF'] = [variance_inflation_factor(x.values, i) for i in range(l

print("VIF Data after removing 'int_rate':")
print(vif_data)
```

```
C:\Users\avesh\anaconda3\Lib\site-packages\statsmodels\regression\linear_
model.py:1783: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.uncentered_tss

VIF Data after removing 'int_rate':
                      feature        VIF
0                   loan_amnt   6.235590
1                    int_rate  11.174284
2                  emp_length   3.754047
3                  annual_inc   8.567544
4                         dti   7.567699
5                    open_acc  13.445192
6                   revol_bal   5.539522
7                  revol_util   8.711820
8                   total_acc  12.391310
9                       month   4.306063
10              pub_rec_flag        NaN
11             mort_acc_flag   2.940505
12  pub_rec_bankruptcies_flag        NaN
```

In [41]:
```
x = x.drop(columns=['open_acc'])

# Recalculate the VIF values after removing the feature
vif_data = pd.DataFrame()
vif_data['feature'] = x.columns
vif_data['VIF'] = [variance_inflation_factor(x.values, i) for i in range(l

print("VIF Data after removing 'int_rate':")
print(vif_data)
```

```
C:\Users\avesh\anaconda3\Lib\site-packages\statsmodels\regression\linear_
model.py:1783: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.uncentered_tss

VIF Data after removing 'int_rate':
                      feature        VIF
0                   loan_amnt   6.214897
1                    int_rate  10.639142
2                  emp_length   3.752388
3                  annual_inc   8.375660
4                         dti   7.036847
5                   revol_bal   5.294030
6                  revol_util   8.285435
7                   total_acc   8.232678
8                       month   4.259850
9               pub_rec_flag        NaN
10              mort_acc_flag   2.917615
11  pub_rec_bankruptcies_flag        NaN
```

In [42]:
```
x = x.drop(columns=['pub_rec_flag','pub_rec_bankruptcies_flag'])

# Recalculate the VIF values after removing the feature
vif_data = pd.DataFrame()
vif_data['feature'] = x.columns
vif_data['VIF'] = [variance_inflation_factor(x.values, i) for i in range(l

print("VIF Data after removing 'int_rate':")
print(vif_data)
```

```
VIF Data after removing 'int_rate':
         feature        VIF
0      loan_amnt   6.214897
1       int_rate  10.639142
2     emp_length   3.752388
3     annual_inc   8.375660
4            dti   7.036847
5      revol_bal   5.294030
6     revol_util   8.285435
7      total_acc   8.232678
8          month   4.259850
9  mort_acc_flag   2.917615
```

Insight:

```
Not removing int_rate as it feels as an important feature while ap
plying for a loan.
```

Hyperparameter Tuning

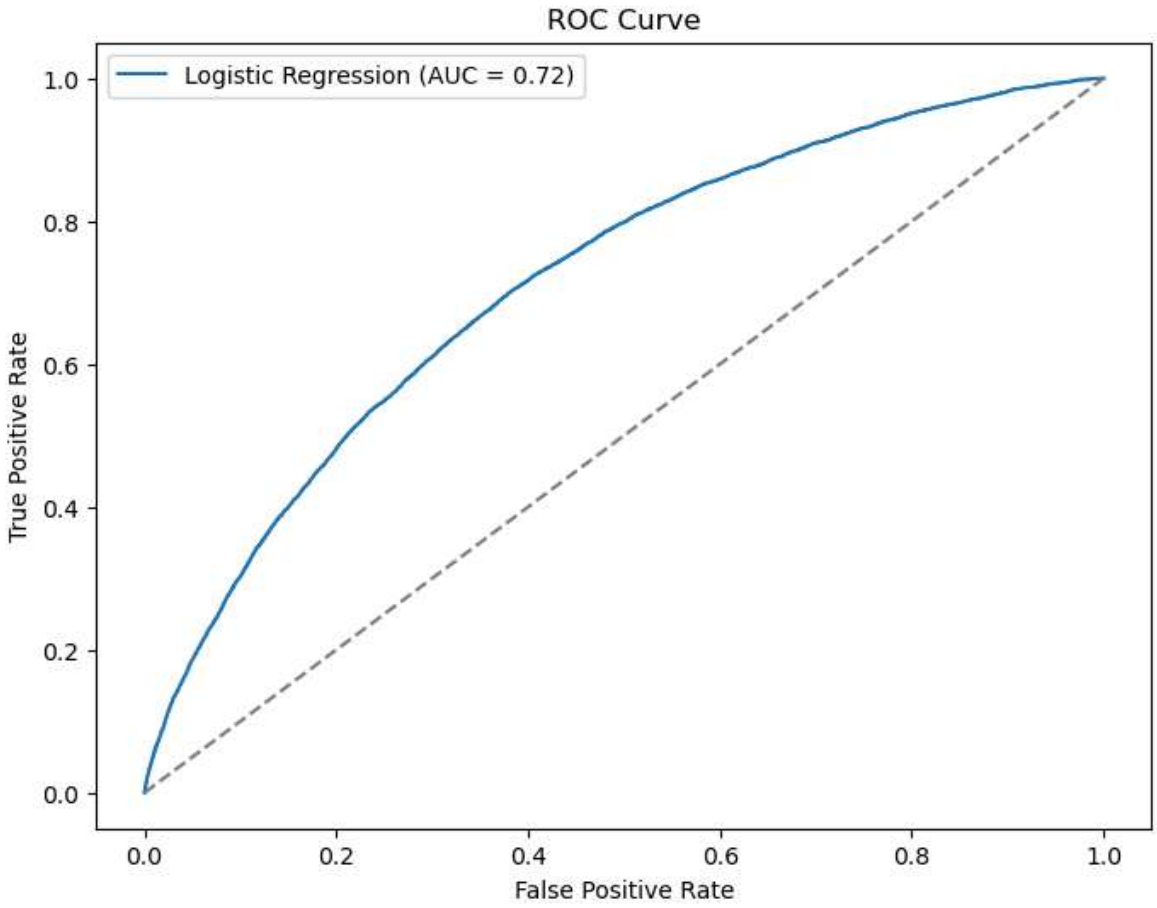
In [43]:

```
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularization strength
    'penalty': ['l1', 'l2'],  # Penalty type (Note: 'l1' works only with '
    'solver': ['liblinear', 'saga']  # Solvers that support 'l1' penalty
}


grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, s

# Perform hyperparameter tuning using the final training dataset
grid_search.fit(X_train_final, y_train)

# Get the best parameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("Best Hyperparameters:", best_params)

# Evaluate the best model on the test set
y_test_pred = best_model.predict(X_test_final)
accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy of Best Model: {accuracy:.2f}")
```

```
Best Hyperparameters: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Test Accuracy of Best Model: 0.81
```

In [44]:
```
from sklearn.metrics import classification_report, confusion_matrix, roc_a
import matplotlib.pyplot as plt

# Get predictions on the test set
y_test_pred = best_model.predict(X_test_final)
y_test_pred_proba = best_model.predict_proba(X_test_final)[:, 1]

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_test_pre

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:\n", conf_matrix)

# ROC AUC Score
roc_auc = roc_auc_score(y_test, y_test_pred_proba)
print(f"ROC AUC Score: {roc_auc:.2f}")

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

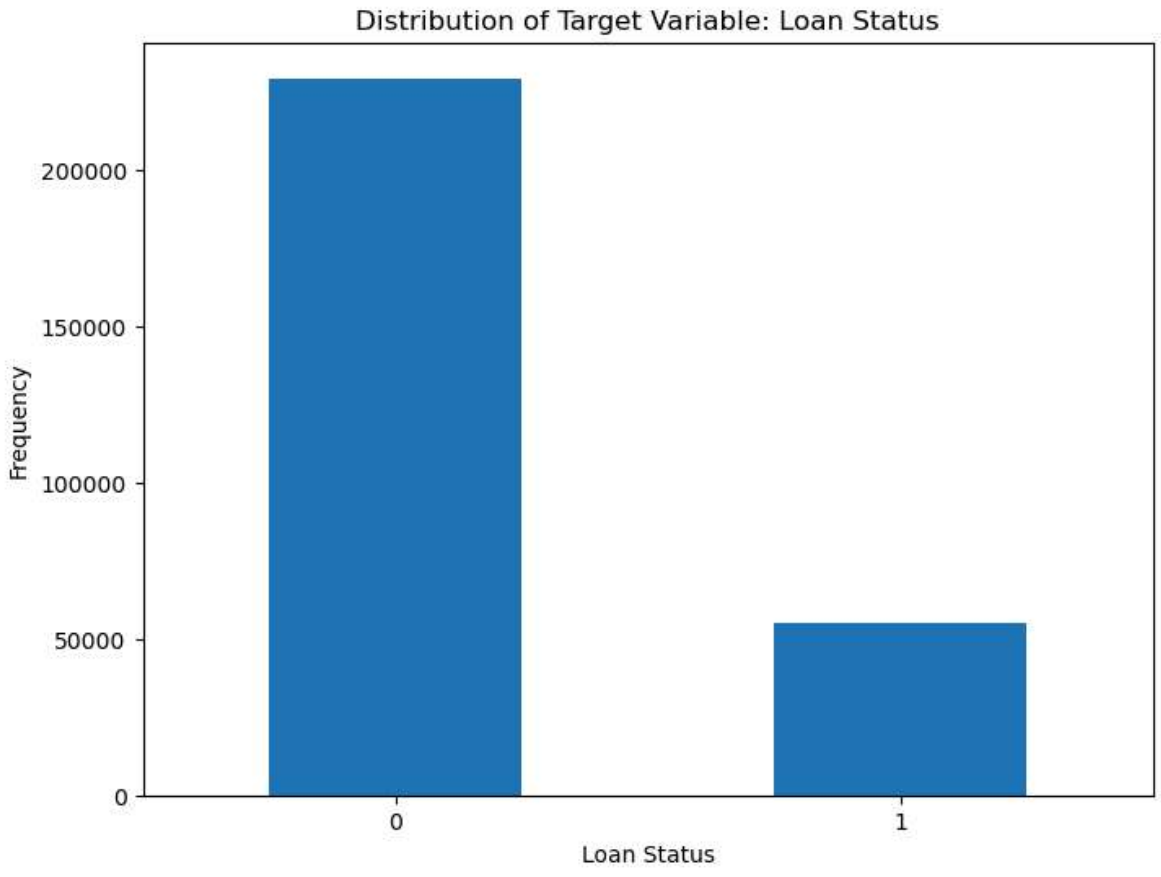
```
Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.98      0.89     45842
           1       0.54      0.08      0.14     11039

    accuracy                           0.81     56881
   macro avg       0.68      0.53      0.51     56881
weighted avg       0.76      0.81      0.75     56881

Confusion Matrix:
 [[45092   750]
 [10166   873]]
ROC AUC Score: 0.72
```


ROC Curve
Logistic Regression (AUC = 0.72)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

In [45]:

```
target_distribution = df_filtered['loan_status'].value_counts()

print("Target Variable Distribution:")
print(target_distribution)

# Plot the distribution of the target variable
plt.figure(figsize=(8, 6))
target_distribution.plot(kind='bar')
plt.title('Distribution of Target Variable: Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Frequency')
plt.xticks(rotation=0)
plt.show()
```

```
Target Variable Distribution:
loan_status
0    229288
1     55114
Name: count, dtype: int64
```



Insight:

This indicates class imbalance, which can negatively impact model performance. The model may become biased towards the majority clas s.

In [50]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)

X_train_resampled, y_train_resampled = smote.fit_resample(X_train_final, y

print("Class distribution after applying SMOTE:")
print(y_train_resampled.value_counts())
```

```
Class distribution after applying SMOTE:
loan_status
0    183446
1    183446
Name: count, dtype: int64
```

In [51]:
```
model.fit(X_train_resampled, y_train_resampled)

# Predict on the test set
y_test_pred = model.predict(X_test_final)

# Evaluate the model's performance
print("Classification Report:\n", classification_report(y_test, y_test_pre
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
```

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

           0       0.89      0.66      0.76     45842
           1       0.32      0.65      0.43     11039

    accuracy                           0.66     56881
   macro avg       0.60      0.66      0.59     56881
weighted avg       0.78      0.66      0.69     56881

Confusion Matrix:
 [[30346 15496]
 [ 3827  7212]]
```

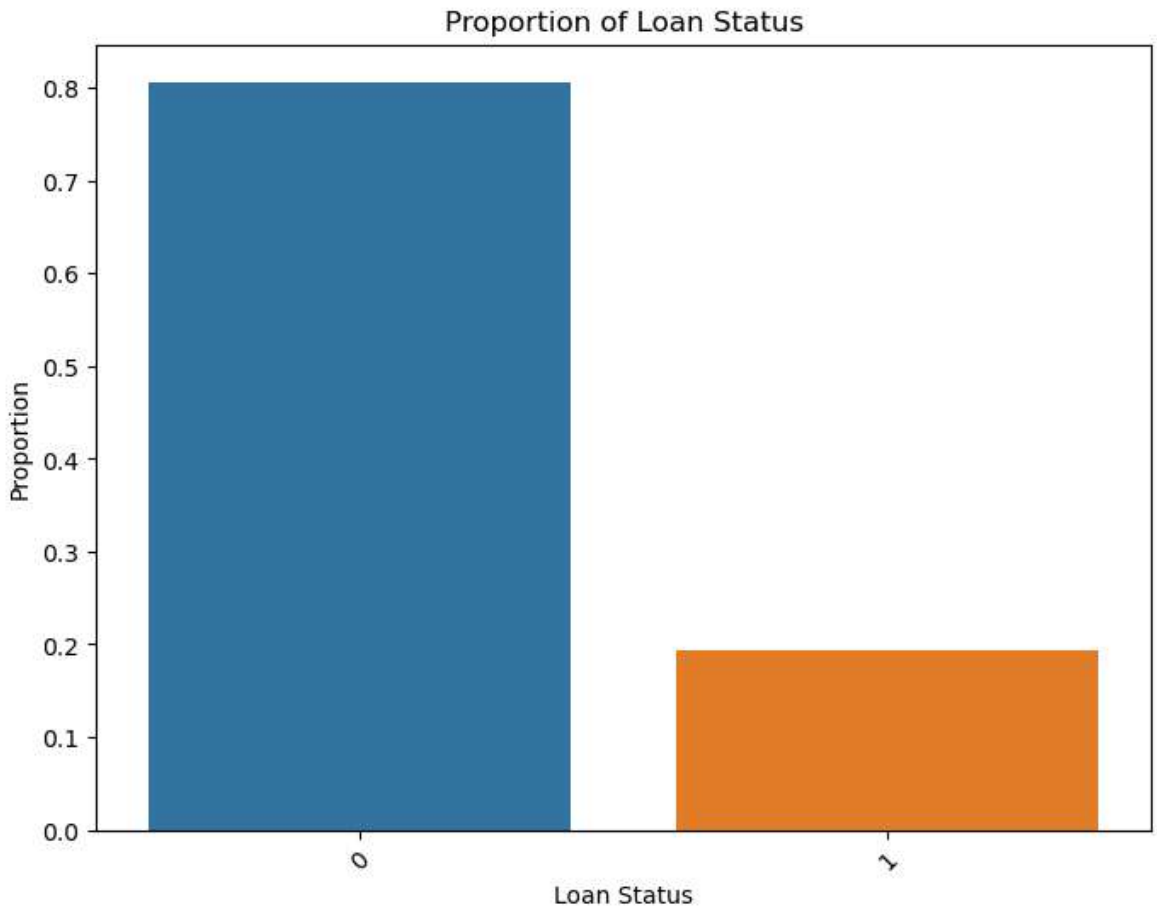
In [52]:
```
best_model = grid_search.best_estimator_

# Extract coefficients and feature names
coefficients = best_model.coef_[0]
feature_names = X_train_final.columns

# Create a DataFrame to map coefficients to predictor names
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})

# Sort the DataFrame by absolute coefficient values for better interpretat
feature_importance['Absolute Coefficient'] = feature_importance['Coefficie
feature_importance = feature_importance.sort_values(by='Absolute Coefficie

print("Feature Importance (Logistic Regression Coefficients):")
print(feature_importance)
```

```
Feature Importance (Logistic Regression Coefficients):
                      Feature  Coefficient  Absolute Coefficient
20                    issue_d     5.205523              5.205523
15                       term     2.098969              2.098969
18            home_ownership     1.975587              1.975587
17                 sub_grade     1.637007              1.637007
19       verification_status     1.547227              1.547227
22       initial_list_status    -1.181623              1.181623
21                   purpose     0.927462              0.927462
23          application_type    -0.623458              0.623458
16                     grade     0.360811              0.360811
1                    int_rate     0.271385              0.271385
4                  annual_inc    -0.230956              0.230956
5                         dti     0.145962              0.145962
0                   loan_amnt     0.142860              0.142860
6                    open_acc     0.120966              0.120966
8                   revol_util     0.107237              0.107237
9                    total_acc    -0.097349              0.097349
7                    revol_bal    -0.075218              0.075218
24                      region    -0.065428              0.065428
11                        year    -0.046990              0.046990
2                  installment    -0.040528              0.040528
13             mort_acc_flag    -0.038052              0.038052
3                  emp_length    -0.016425              0.016425
10                      month    -0.015291              0.015291
14  pub_rec_bankruptcies_flag     0.000000              0.000000
12               pub_rec_flag     0.000000              0.000000
```

What percentage of customers have fully paid their Loan Amount?

In [61]:
```
fully_paid_count = (df['loan_status'] == 0).sum()
total_count = len(df['loan_status'])
fully_paid_proportion = fully_paid_count / total_count

print(f"Proportion of Fully Paid Loans: {fully_paid_proportion:.2%}")

# Calculate the proportions of each category in the 'loan_status' column
loan_status_proportion = df['loan_status'].value_counts(normalize=True)

# Plotting the proportion of each loan status using seaborn barplot
plt.figure(figsize=(8, 6))
sns.barplot(x=loan_status_proportion.index, y=loan_status_proportion.value
plt.title('Proportion of Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Proportion')
plt.xticks(rotation=45)
plt.show()
```

```
Proportion of Fully Paid Loans: 80.62%
```



Insight:

80% of the loan are fully paid.

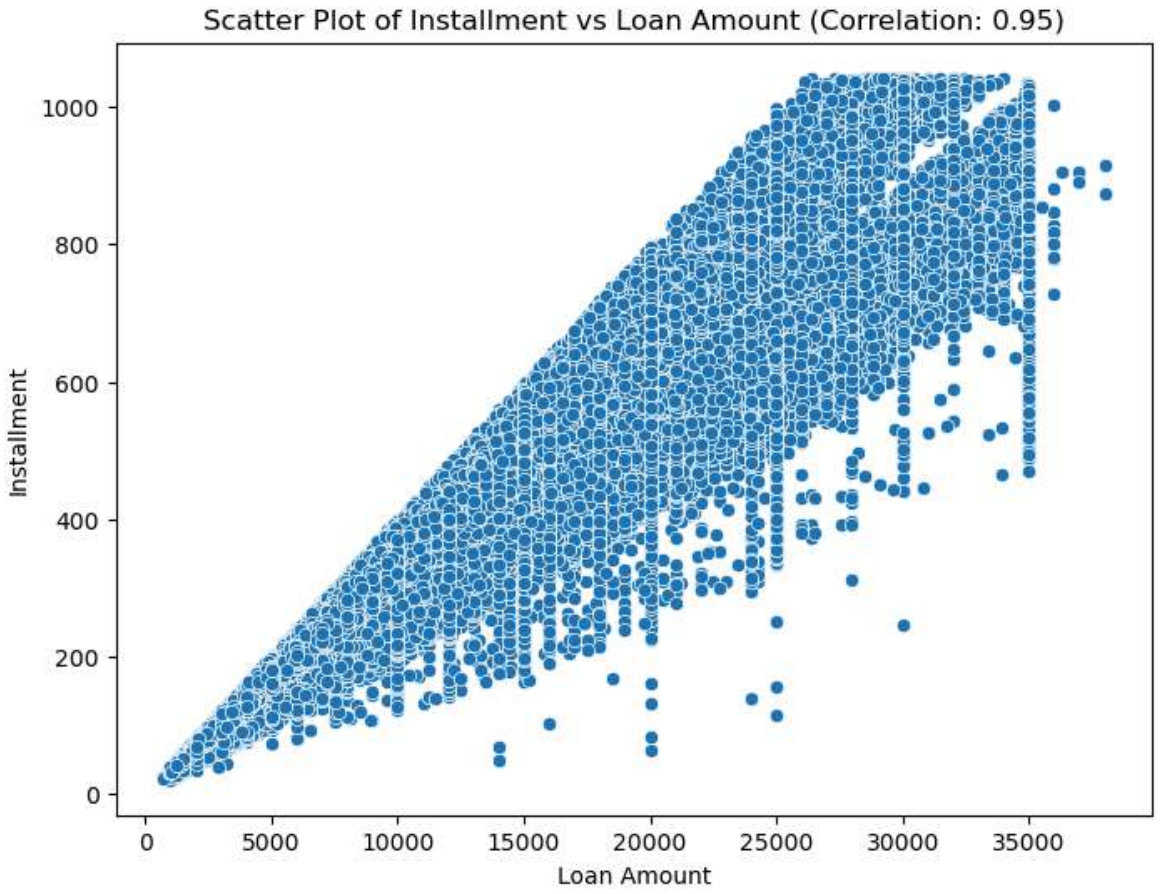
Comment about the correlation between Loan Amount and Installment features.

In [63]:
```
df.columns
```

Out[63]:
```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'verification_stat
us',
       'issue_d', 'purpose', 'initial_list_status', 'application_type',
       'region', 'loan_amnt', 'int_rate', 'installment', 'emp_length',
       'annual_inc', 'dti', 'open_acc', 'revol_bal', 'revol_util', 'total
_acc',
       'month', 'year', 'pub_rec_flag', 'mort_acc_flag',
       'pub_rec_bankruptcies_flag', 'loan_status'],
      dtype='object')
```

In [64]:
```
correlation = df['installment'].corr(df['loan_amnt'])
print(f"Correlation between 'installment' and 'loan_amnt': {correlation:.2

# Visualize the relationship using a scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='loan_amnt', y='installment', data=df)
plt.title(f'Scatter Plot of Installment vs Loan Amount (Correlation: {corr
plt.xlabel('Loan Amount')
plt.ylabel('Installment')
plt.show()
```

```
Correlation between 'installment' and 'loan_amnt': 0.95
```



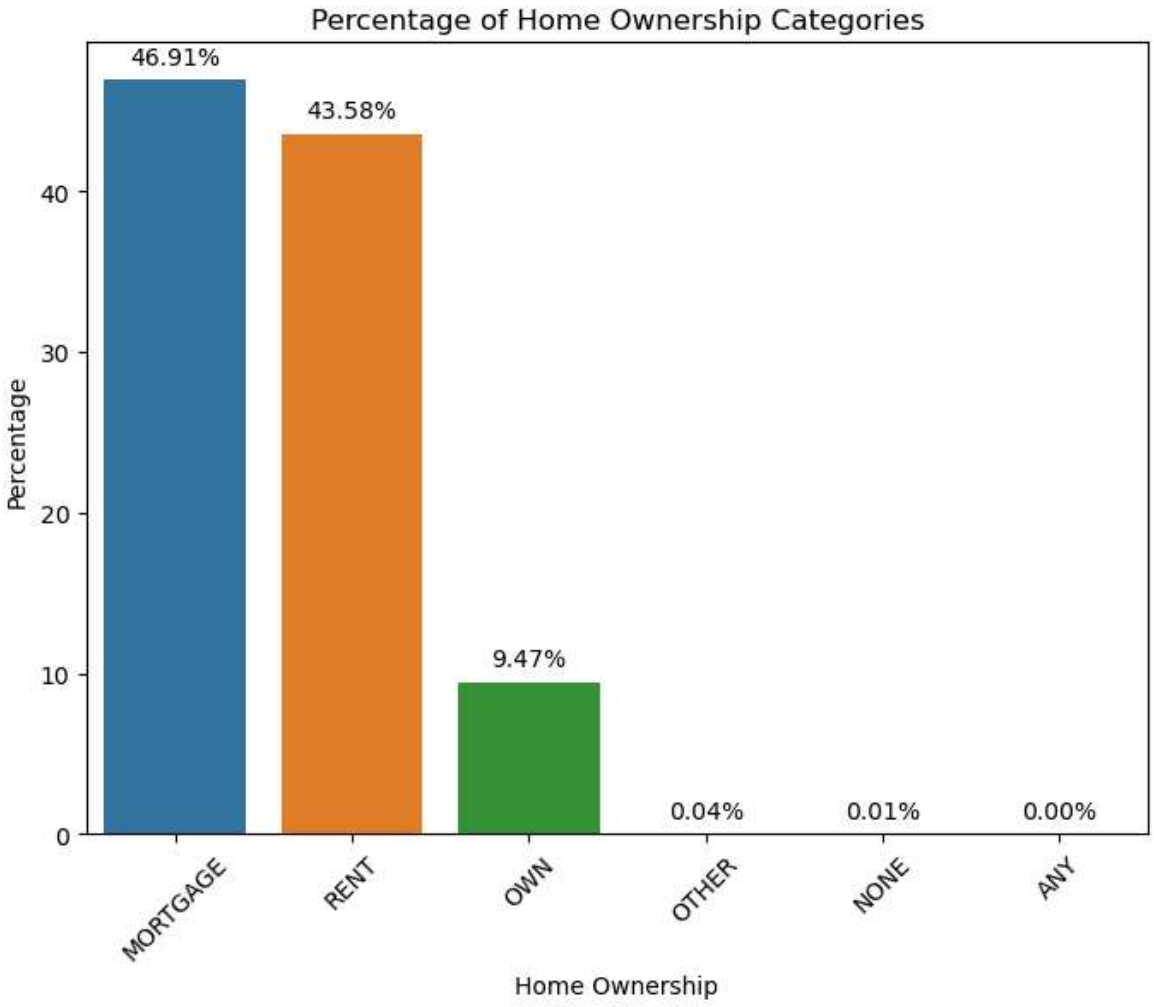
Insight:

There is a strong co-relation between loan amount and installment.

The majority of people have home ownership as?

In [69]:
```
home_ownership_counts = df['home_ownership'].value_counts(normalize=True)
print("Percentage of each Home Ownership category:")
print(home_ownership_counts)

# Convert counts to DataFrame for visualization
home_ownership_df = home_ownership_counts.reset_index()
home_ownership_df.columns = ['home_ownership', 'percentage']

# Plotting the percentage of each category using seaborn barplot
plt.figure(figsize=(8, 6))
sns.barplot(data=home_ownership_df, x='home_ownership', y='percentage')
plt.title('Percentage of Home Ownership Categories')
plt.xlabel('Home Ownership')
plt.ylabel('Percentage')
plt.xticks(rotation=45)

# Annotate percentages on the bars
for index, row in home_ownership_df.iterrows():
    plt.text(row.name, row.percentage + 1, f'{row.percentage:.2f}%', ha='c

plt.show()
```

```
Percentage of each Home Ownership category:
home_ownership
MORTGAGE    46.905788
RENT        43.578456
OWN          9.471101
OTHER        0.035865
NONE         0.008087
ANY          0.000703
Name: proportion, dtype: float64
```



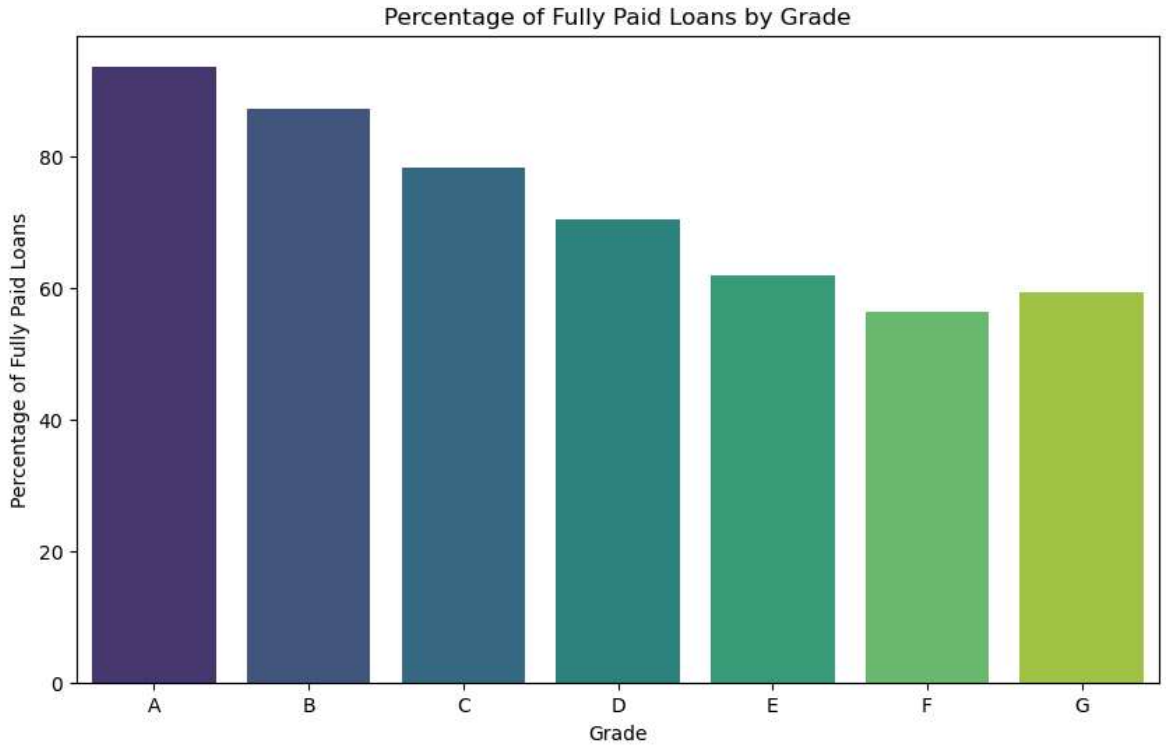
Insight:

Majority of people have Home Ownership as Mortgage(47%) followed b
y Rent(43.5%) and 9.5%
people own their properties.

In [77]:

```
# Calculate the percentage of loans that are 'Fully Paid' for each grade
grade_loan_status = df.groupby('grade')['loan_status'].value_counts(normal

print("Percentage of 'Fully Paid' Loans by Grade:")
print(grade_loan_status[0])

# Plot the percentages using a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=grade_loan_status.index, y=grade_loan_status[0], palette='vi
plt.title('Percentage of Fully Paid Loans by Grade')
plt.xlabel('Grade')
plt.ylabel('Percentage of Fully Paid Loans')
plt.show()

# Check if people with grades 'A' are more likely to fully pay their loan
is_grade_a_more_likely = grade_loan_status[0]['A'] > grade_loan_status[0].
print(f"People with grades 'A' are more likely to fully pay their loan: {i
```

```
Percentage of 'Fully Paid' Loans by Grade:
grade
A    93.682274
B    87.216290
C    78.303653
D    70.474109
E    61.892253
F    56.349851
G    59.302326
Name: 0, dtype: float64
```




```
People with grades 'A' are more likely to fully pay their loan: True
```

Thinking from a bank's perspective, which metric should our primary focus be

on..

1. ROC AUC
2. Precision
3. Recall
4. F1 Score

Insight

Precison And Recall should be the primary focus for the bank

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

model is very selective and cautious. It approves loans only when it is very confident that the loan will be "Fully Paid."

This reduces the risk of defaults (false positives) but may result in many missed opportunities (false negatives), where potentially good customers are rejected.

Business Impact: Risk Mitigation: Low risk of bad loans (defaults), which is good for maintaining the bank's financial health. Opportunity Loss: Misses out on approving many potentially good loans, leading to lower growth and customer acquisition.

Customer Dissatisfaction: Rejection of eligible customers can result in negative customer experience and loss of market reputation.

Recommendations

In [ ]:

```
1.Create a tiered interest rate structure that not only offers lower rates
but also dynamically adjusts interest rates based on a combination of cred
debt-to-income (DTI) ratios, and other relevant features.

2.Encourages a broader range of customers to apply for loans while still m
to lower-grade customers who represent higher risk.
Leverage Alternative Data for Enhanced Credit Scoring:

3.Use alternative data sources such as utility payments, rental history, a
credit scoring models, especially for customers with limited credit histor
Expands the pool of potential customers, especially in underserved segment
This can help in identifying "hidden gems" among applicants who may not ha
Develop a Predictive Early Warning System for Loan Defaults:

4.Utilize customer segmentation to run targeted marketing campaigns for di
    interest rates, higher loan amounts, and loyalty benefits.
Middle-Grade Customers: Provide conditional offers such as lower rates for
    small rate reduction for demonstrating consistent payment behavior.
Low-Grade Customers: Offer credit counseling, educational materials, and s
    them build or repair their credit.

5.Provide free or low-cost financial literacy programs and loan advisory s
borrowers about managing credit, maintaining good grades, and understandin
Increases trust and customer loyalty, improves creditworthiness over time,
as customers become more financially savvy.

6.Optimize the Approval Process Using AI and Automation:
Use AI-driven automation to streamline the loan approval process, reduce t
AI can help in assessing creditworthiness more accurately by analyzing str
Faster approval process leads to better customer experience, reduces opera
overall efficiency of the loan approval process.

7.Implement a continuous feedback loop to monitor the performance of loan
and trends, retrain models regularly, and adjust strategies accordingly.
Ensures that models remain accurate, robust, and aligned with current busi
conditions, allowing for agile decision-making.
```