```
#Data processing
import pandas as pd
import numpy as np

#Data Visualisation
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline

#Seting option for full column view of Data
pd.set_option('display.max_columns', None)

#Stats & model building
from scipy import stats
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (accuracy_score, confusion_matrix,
                             roc_curve, auc, ConfusionMatrixDisplay,
                             f1_score, recall_score,
                             precision_score, precision_recall_curve,
                             average_precision_score, classification_report)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE

#Hide warnings
import warnings
warnings.filterwarnings("ignore")
```

```
!gdown "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921"
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/003/549/original/logistic_regression.csv?1651045921
To: /content/logistic_regression.csv?1651045921
100% 100M/100M [00:04<00:00, 22.7MB/s]
```

```
df = pd.read_csv("logistic_regression.csv?1651045921")
```

```
df.head(5)
```

llment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	verification_status	issue_d	loan_status	purpose	title
329.48	B	B4	Marketing	10+ years	RENT	117000.0	Not Verified	Jan-2015	Fully Paid	vacation	Vacation
265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	Not Verified	Jan-2015	Fully Paid	debt_consolidation	Debt consolidation
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Credit card

```
#Shape of Data
df.shape
```

```
(396030, 27)
```

```
# Statistical summary

df.describe()
```

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies
count	396030.000000	396030.000000	396030.000000	3.960300e+05	396030.000000	396030.000000	396030.000000	3.960300e+05	395754.000000	396030.000000	358235.000000	395495.000000
mean	14113.888089	13.639400	431.849698	7.420318e+04	17.379514	11.311153	0.178191	1.584454e+04	53.791749	25.414744	1.813991	0.121648
std	8357.441341	4.472157	250.727790	6.163762e+04	18.019092	5.137649	0.530671	2.059184e+04	24.452193	11.886991	2.147930	0.356174
min	500.000000	5.320000	16.080000	0.000000e+00	0.000000	0.000000	0.000000	0.000000e+00	0.000000	2.000000	0.000000	0.000000
25%	8000.000000	10.490000	250.330000	4.500000e+04	11.280000	8.000000	0.000000	6.025000e+03	35.800000	17.000000	0.000000	0.000000
50%	12000.000000	13.330000	375.430000	6.400000e+04	16.910000	10.000000	0.000000	1.118100e+04	54.800000	24.000000	1.000000	0.000000
75%	20000.000000	16.490000	567.300000	9.000000e+04	22.980000	14.000000	0.000000	1.962000e+04	72.900000	32.000000	3.000000	0.000000
max	40000.000000	30.990000	1533.810000	8.706582e+06	9999.000000	90.000000	86.000000	1.743266e+06	892.300000	151.000000	34.000000	8.000000

```
#Data Cleaning
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                 394275 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
```

```
# Non-numeric columns
cat_cols = df.select_dtypes(include='object').columns
cat_cols
```

```
Index(['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'],
      dtype='object')
```

```
# Number of unique values in all non-numeric columns
for col in cat_cols:
  print(f"No. of unique values in {col}: {df[col].nunique()}")
```

```
No. of unique values in term: 2
No. of unique values in grade: 7
No. of unique values in sub_grade: 35
No. of unique values in emp_title: 173105
No. of unique values in emp_length: 11
No. of unique values in home_ownership: 6
No. of unique values in verification_status: 3
No. of unique values in issue_d: 115
No. of unique values in loan_status: 2
No. of unique values in purpose: 14
No. of unique values in title: 48817
No. of unique values in earliest_cr_line: 684
No. of unique values in initial_list_status: 2
No. of unique values in application_type: 3
No. of unique values in address: 393700
```

```
# Convert earliest credit line & issue date to datetime
df['earliest_cr_line'] = pd.to_datetime(df['earliest_cr_line'])
df['issue_d'] = pd.to_datetime(df['issue_d'])
```

```
#Convert employment length to numeric
d = {'10+ years':10, '4 years':4, '< 1 year':0,
     '6 years':6, '9 years':9,'2 years':2, '3 years':3,
     '8 years':8, '7 years':7, '5 years':5, '1 year':1}
df['emp_length']=df['emp_length'].replace(d)
```

```
#Convert columns with less number of unique values to categorical columns
cat_cols = ['term', 'grade','sub_grade','home_ownership',
            'verification_status','loan_status','purpose',
            'initial_list_status','application_type']

df[cat_cols] = df[cat_cols].astype('category')
```

```
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  category
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  category
 5   sub_grade             396030 non-null  category
 6   emp_title             373103 non-null  object
 7   emp_length            377729 non-null  float64
 8   home_ownership        396030 non-null  category
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  category
 11  issue_d               396030 non-null  datetime64[ns]
 12  loan_status           396030 non-null  category
 13  purpose               396030 non-null  category
 14  title                 394275 non-null  object
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  datetime64[ns]
 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  category
 23  application_type      396030 non-null  category
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: category(9), datetime64[ns](2), float64(13), object(3)
memory usage: 57.8+ MB
```

```
#Checking for duplicate values
```

```
df.duplicated().sum()
```

```
0
```

```
#Handling missing values
df.isna().sum()
```

```
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                    1755
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
```

```
#Filling missing values with 'Unknown' for object dtype
fill_values = {'title': 'Unknown', 'emp_title': 'Unknown'}
df.fillna(value=fill_values, inplace=True)
```

```
#Mean aggregation of mort_acc by total_acc to fill missing values

avg_mort = df.groupby('total_acc')['mort_acc'].mean()

def fill_mort(total_acc, mort_acc):
  if np.isnan(mort_acc):
    return avg_mort[total_acc].round()
  else:
    return mort_acc
```

```
df['mort_acc'] = df.apply(lambda x: fill_mort(x['total_acc'],x['mort_acc']), axis=1)
```

```
df.dropna(inplace=True)
```

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

```
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
```

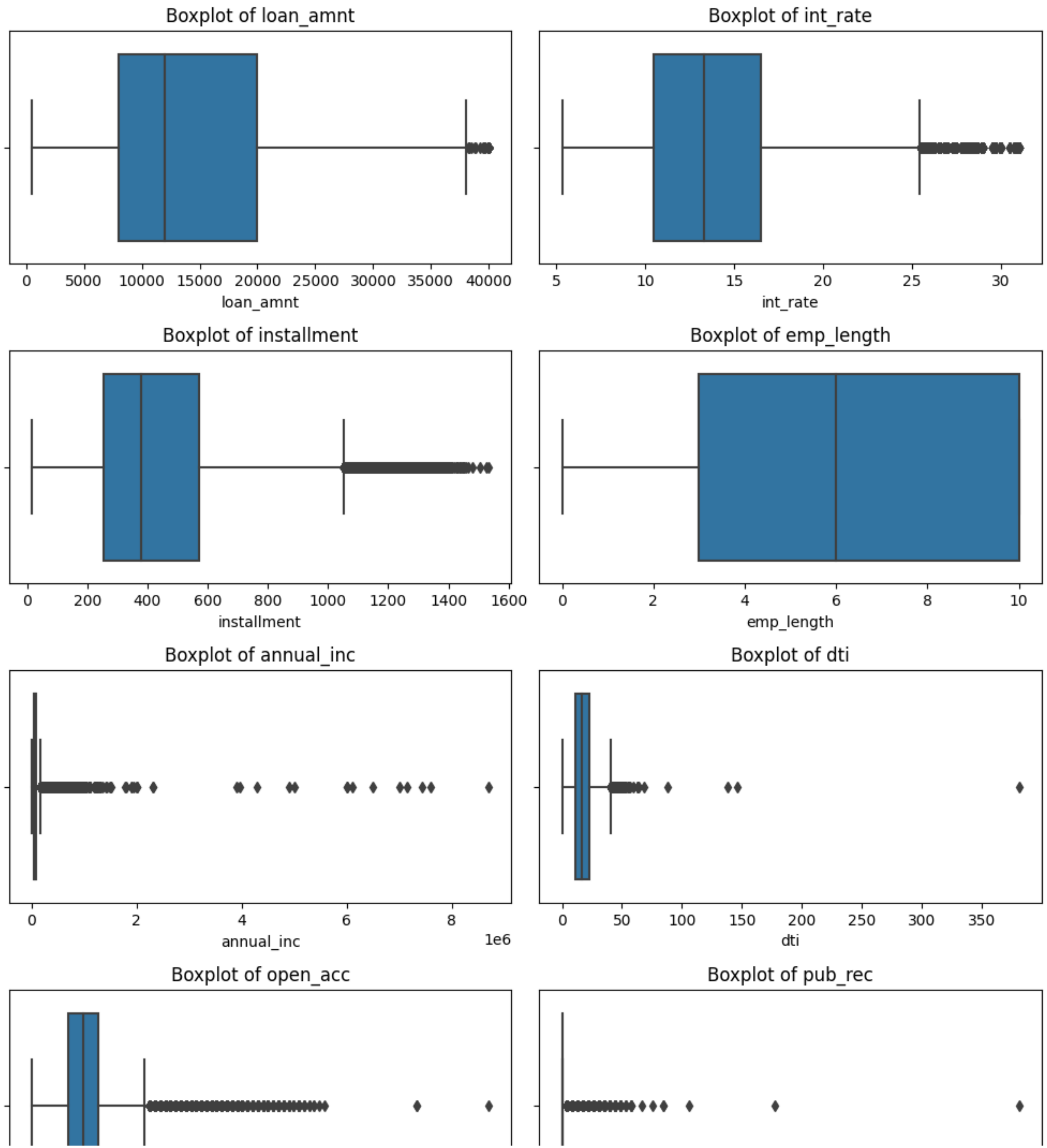
```
df.shape
```

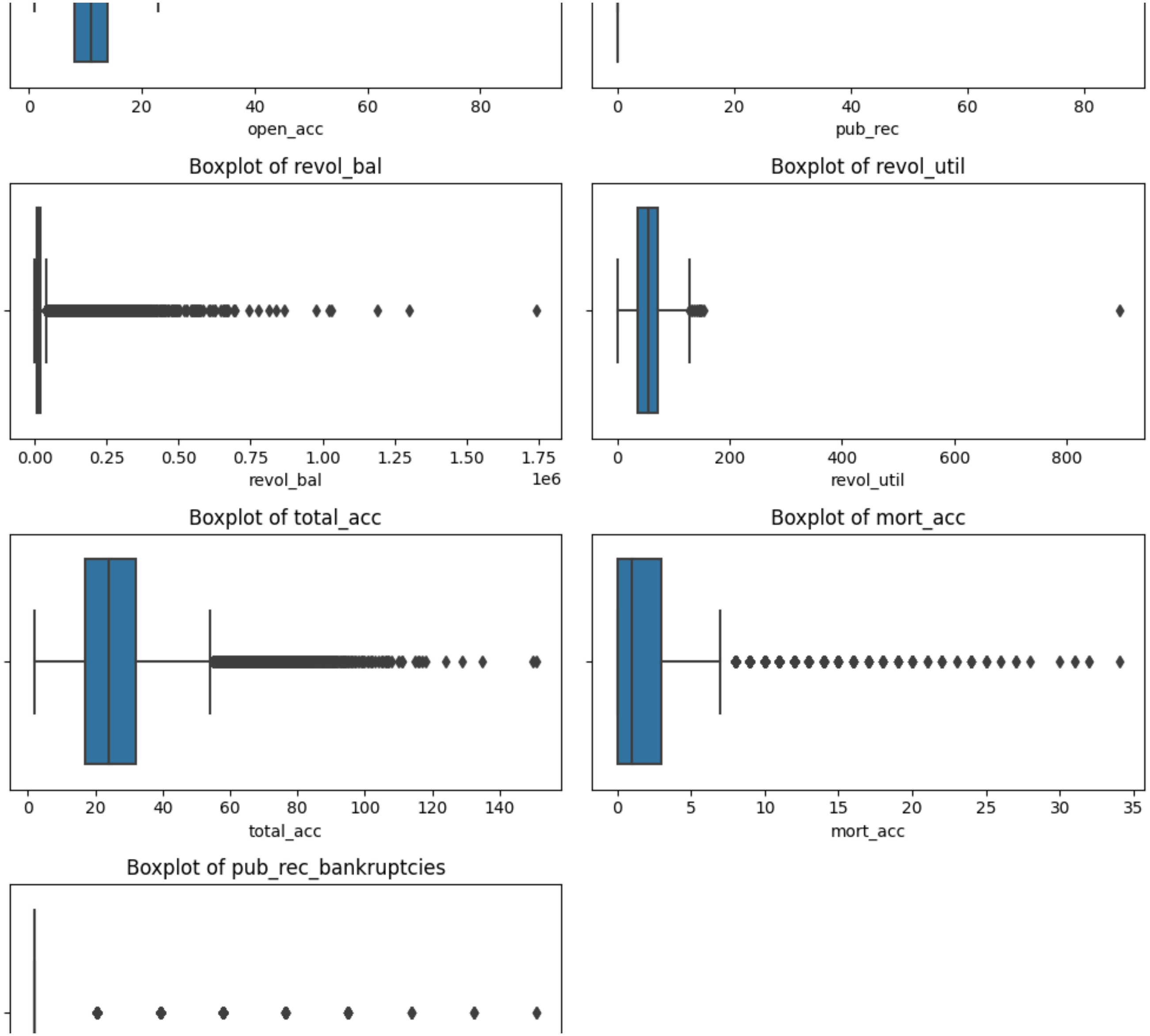
```
(376929, 27)
```

```
#Outlier Treatment
num_cols = df.select_dtypes(include='number').columns
num_cols
```

```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')
```

```
fig = plt.figure(figsize=(10,21))
i=1
for col in num_cols:
  ax = plt.subplot(7,2,i)
  sns.boxplot(x=df[col])
  plt.title(f'Boxplot of {col}')
  i += 1

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


Boxplot of loan_amnt
0 5000 10000 15000 20000 25000 30000 35000 40000
loan_amnt
Boxplot of int_rate
5 10 15 20 25 30
int_rate
Boxplot of installment
0 200 400 600 800 1000 1200 1400 1600
installment
Boxplot of emp_length
0 2 4 6 8 10
emp_length
Boxplot of annual_inc
0 2 4 6 8
1e6
annual_inc
Boxplot of dti
0 50 100 150 200 250 300 350
dti
Boxplot of open_acc
Boxplot of pub_rec




```
#To address outliers in most columns, you can remove rows that fall outside of 3 standard deviations from the mean,
# while for "pub_Rec" and "pub_rec_bankruptcies," consider using a binary approach (0 or 1) to handle their values.
```

```
# Convert pub_rec and pub_rec_bankruptcies to categorical variables

df['pub_rec_bankruptcies'] = np.where(df['pub_rec_bankruptcies']>0,'yes','no')
df['pub_rec'] = np.where(df['pub_rec']>0,'yes','no')
df[['pub_rec_bankruptcies','pub_rec']] = df[['pub_rec_bankruptcies','pub_rec']].astype('category')
```

```
# Numeric columns after converting public records to category
num_cols = df.select_dtypes(include='number').columns
num_cols
```

```
Index(['loan_amnt', 'int_rate', 'installment', 'emp_length', 'annual_inc',
       'dti', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'mort_acc'],
      dtype='object')
```

```
#Removing outliers using standard deviation
for col in num_cols:
  mean=df[col].mean()
  std=df[col].std()
  upper = mean + (3*std)
  df = df[~(df[col]>upper)]
```

```
df.shape
```

```
(350845, 27)
```

```
#Feature Engineering
df['address'].sample(10)
```

```
211009    49291 Arthur Rue Suite 679\r\nWest Melanie, MD...
201209     842 Miller Meadow Apt. 343\r\nKimville, RI 29597
151508        15049 Fernandez Falls\r\nBakershire, MT 70466
124922                     USCGC Montgomery\r\nFPO AE 11650
171812        48654 Kevin Mountain\r\nKristenside, MT 22690
175069                           USCGC Sims\r\nFPO AP 05113
357378     4338 Bell Manors Apt. 980\r\nWest John, WA 11650
155826              99402 Moore Row\r\nAllenburgh, VA 30723
229404    64169 Williams Extensions Suite 560\r\nWatkins...
324905    2278 Hanson Coves Suite 092\r\nWest Willievill...
Name: address, dtype: object
```

```
# Deriving zip code and state from address
df[['state', 'zip_code']] = df['address'].apply(lambda x: pd.Series([x[-8:-6], x[-5:]]))
```

```
#Drop address
df.drop(["address"], axis = 1, inplace=True)
```

```
df.zip_code.nunique()
```

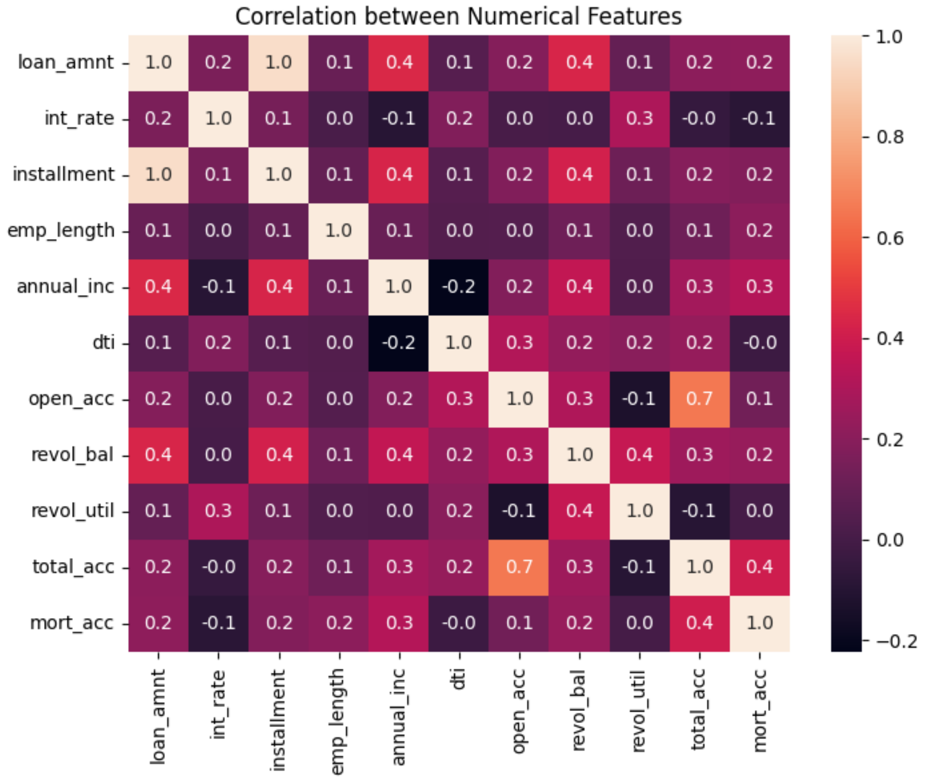
```
10
```

```
# Given that there are only 10 unique zip codes, it's practical to change the data type of zip codes to categorical to
# enhance data analysis and reduce memory usage.
```

```
df['zip_code'] = df['zip_code'].astype('category')
```

```
#Exploratory Data Analysis
```

```
#Correlation between numerical features
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, fmt=".1f")
plt.title('Correlation between Numerical Features')
plt.show()
```

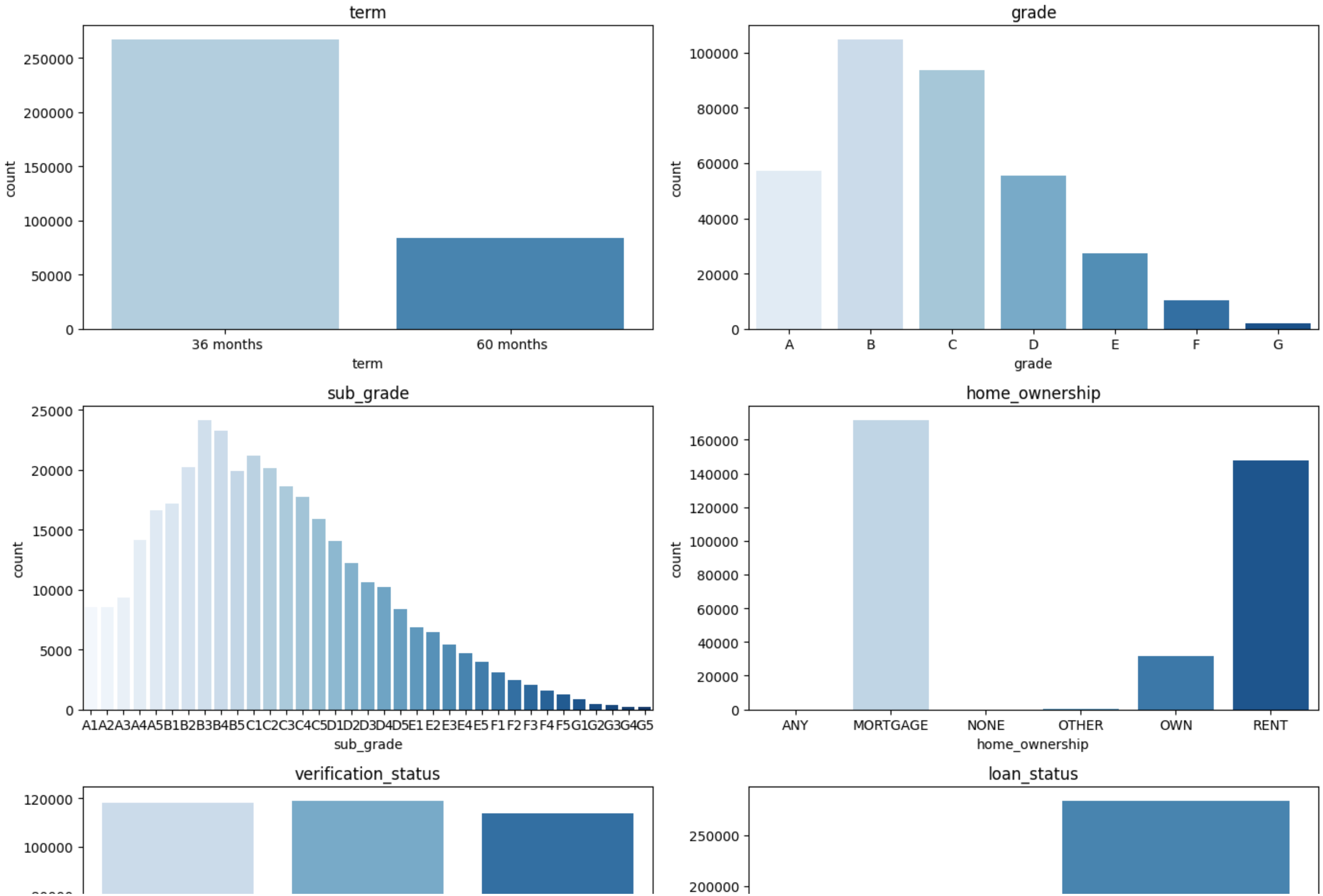



```
#To mitigate multicollinearity, it's advisable to remove either "loan_amnt" or "installment" since they are perfectly correlated,
# and consider eliminating "total_acc" or "mort_acc" to address the high and moderate correlations with "open_acc" and "total_acc," respectively.
```

```
#Drop installment
df.drop(columns=['installment'], inplace=True)
```

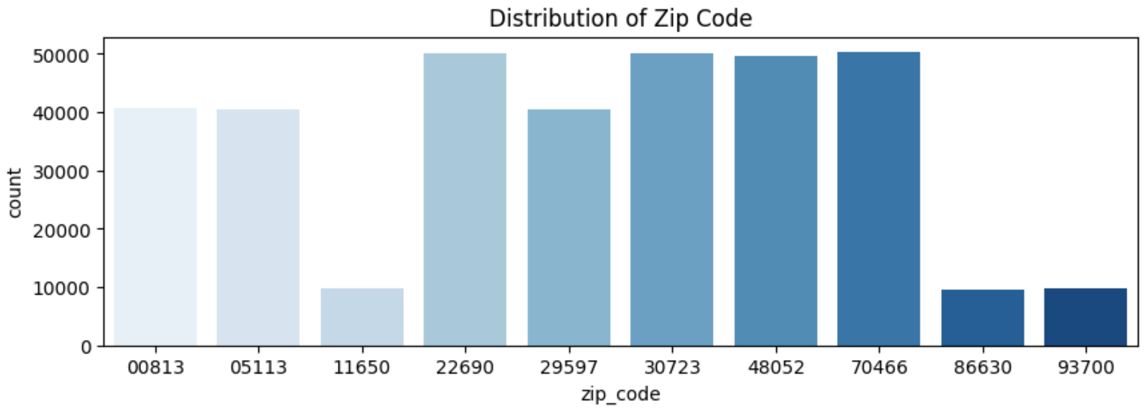
```
#Distribution of categorical variables
plot = ['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'loan_status', 'pub_rec', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies']

plt.figure(figsize=(14,20))
i=1
for col in plot:
  ax=plt.subplot(5,2,i)
  sns.countplot(x=df[col], palette='Blues')
```

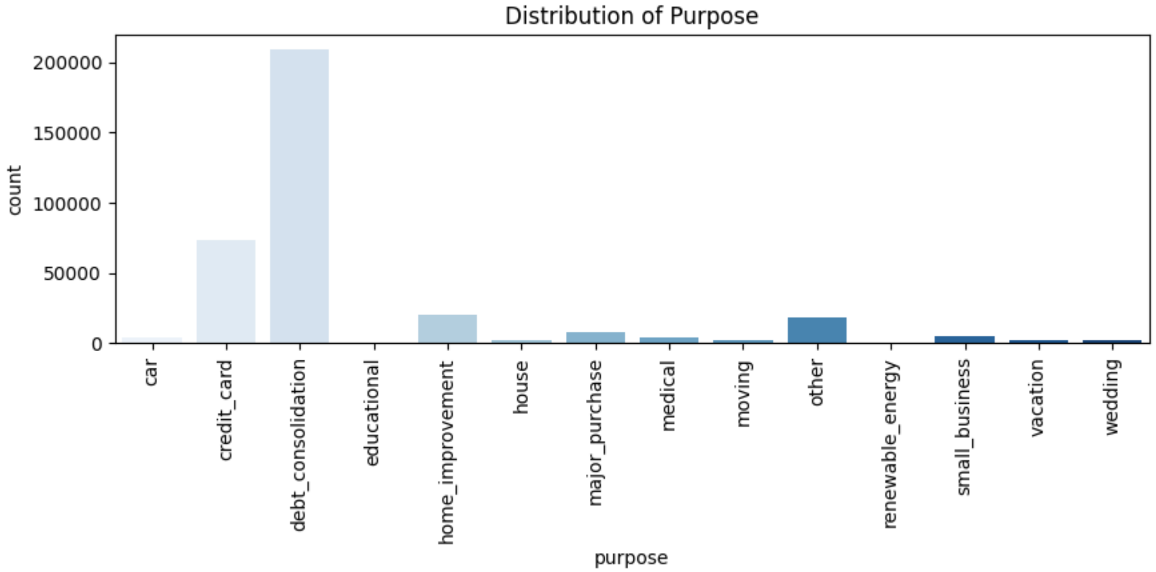
```
    plt.title(f'{col}')
    i += 1

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



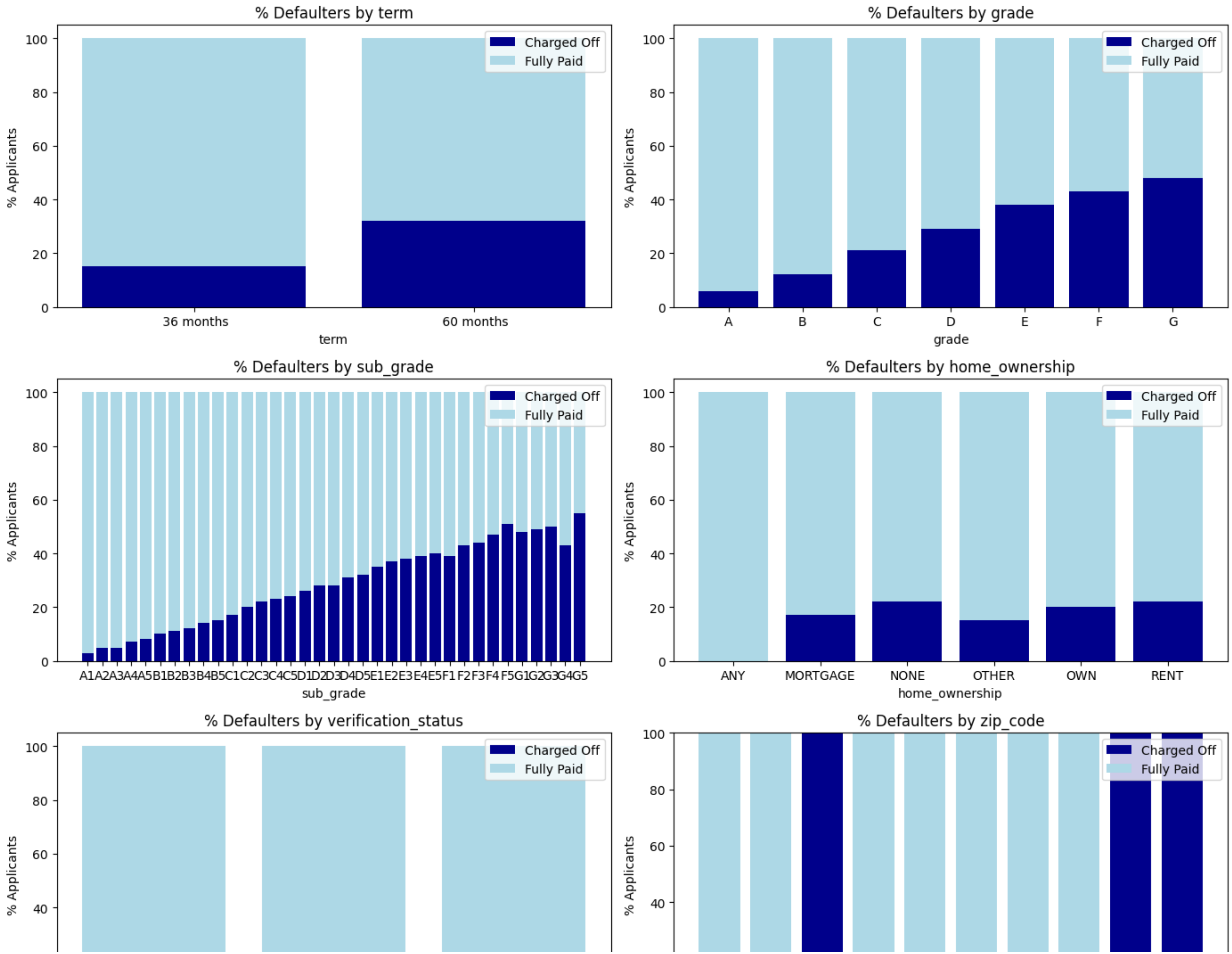

```
plt.figure(figsize=(10,3))
sns.countplot(x=df['zip_code'], palette='Blues')
plt.title('Distribution of Zip Code')

plt.figure(figsize=(10,3))
sns.countplot(x=df['purpose'], palette='Blues')
```

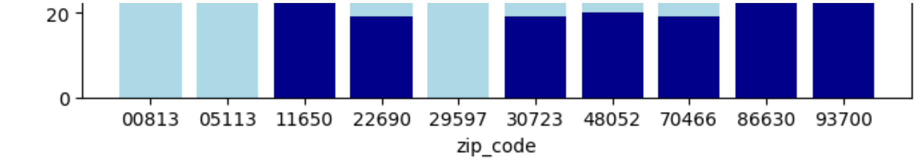
```
plt.xticks(rotation=90)
plt.title('Distribution of Purpose')

plt.show()
```

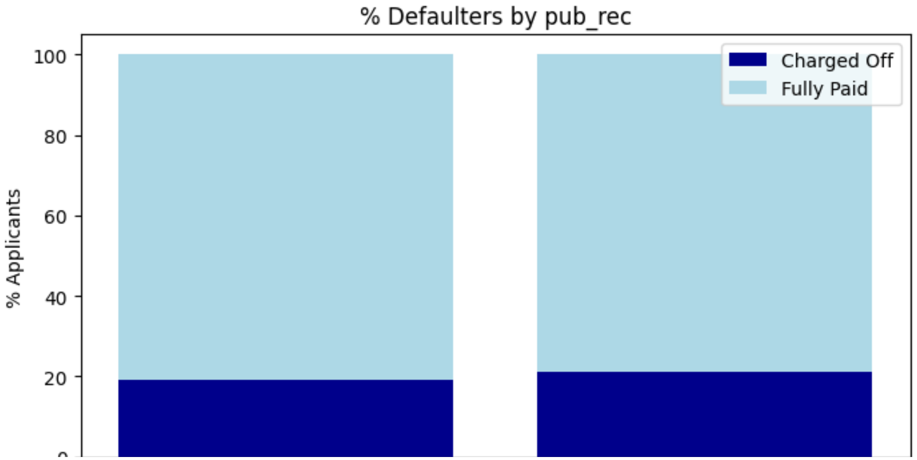


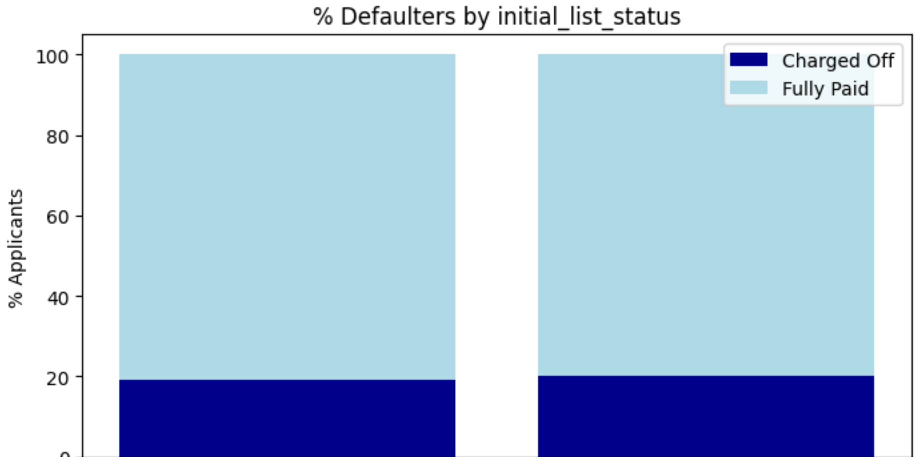


```
# Here are some key observations from the dataset:

# - Approximately 80% of loans have a term of 36 months.
# - The majority of loans (30%) are in the B grade category, followed by C, A, and D grades.
# - Around 50% of borrowers have mortgage as their home ownership type.
# - The dataset is imbalanced with the target variable "loan status" in favor of fully-paid loans, as defaulters make up approximately 25% of
    # fully paid instances.
# - Approximately 85% of applicants do not have a public record or haven't filed for bankruptcy.
# - Almost 99% of applicants have applied under the 'individual' application type.
# - Debt consolidation is the most common purpose for taking out loans, accounting for 55% of cases, followed by credit card purposes at 20%.


# Impact of categorical factors on loan status

plot = ['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'zip_code', 'pub_rec', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies']

plt.figure(figsize=(14,20))
i=1
for col in plot:
  ax=plt.subplot(5,2,i)

  data = df.pivot_table(index=col, columns='loan_status', aggfunc='count', values='purpose')
  data = data.div(data.sum(axis=1), axis=0).multiply(100).round()
  data.reset_index(inplace=True)

  plt.bar(data[col],data['Charged Off'], color='#00008b')
  plt.bar(data[col],data['Fully Paid'], color='#add8e6', bottom=data['Charged Off'])
  plt.xlabel(f'{col}')
  plt.ylabel('% Applicants')
  plt.title(f'% Defaulters by {col}')
  plt.legend(['Charged Off','Fully Paid'])
  i += 1

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

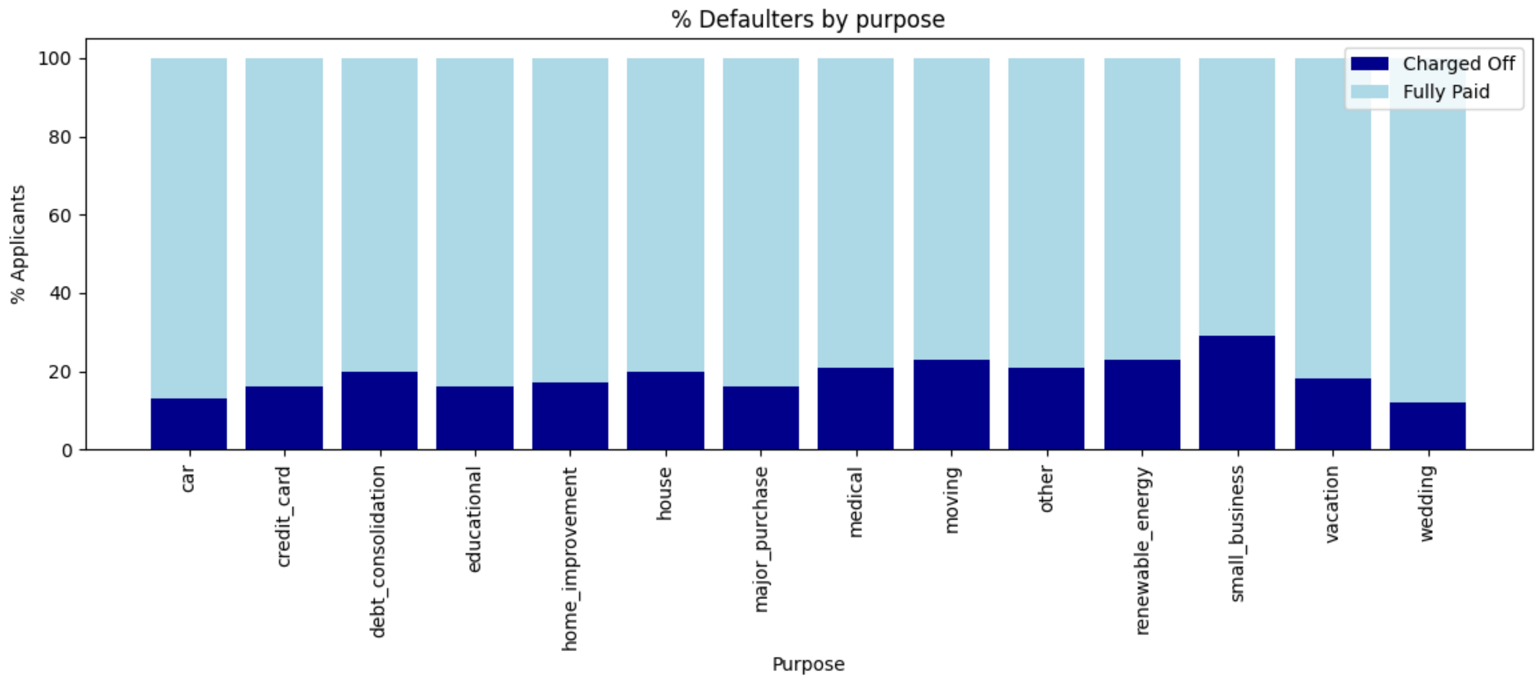

% Defaulters by term
% Applicants
36 months
60 months
term
Charged Off
Fully Paid
% Defaulters by grade
A B C D E F G
grade
% Defaulters by sub_grade
A1A2A3A4A5B1B2B3B4B5C1C2C3C4C5D1D2D3D4D5E1E2E3E4E5F1 F2F3F4 F5G1G2G3G4G5
sub_grade
% Defaulters by home_ownership
ANY
MORTGAGE
NONE
OTHER
OWN
RENT
home_ownership
% Defaulters by verification_status
% Defaulters by zip_code
0
20
40
60
80
100

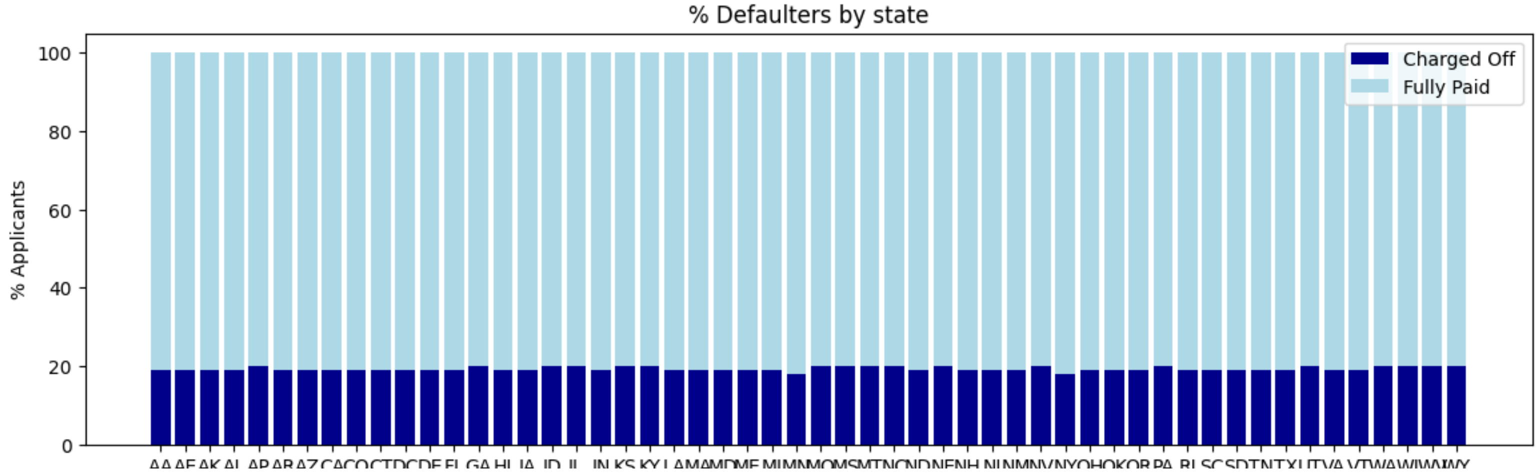






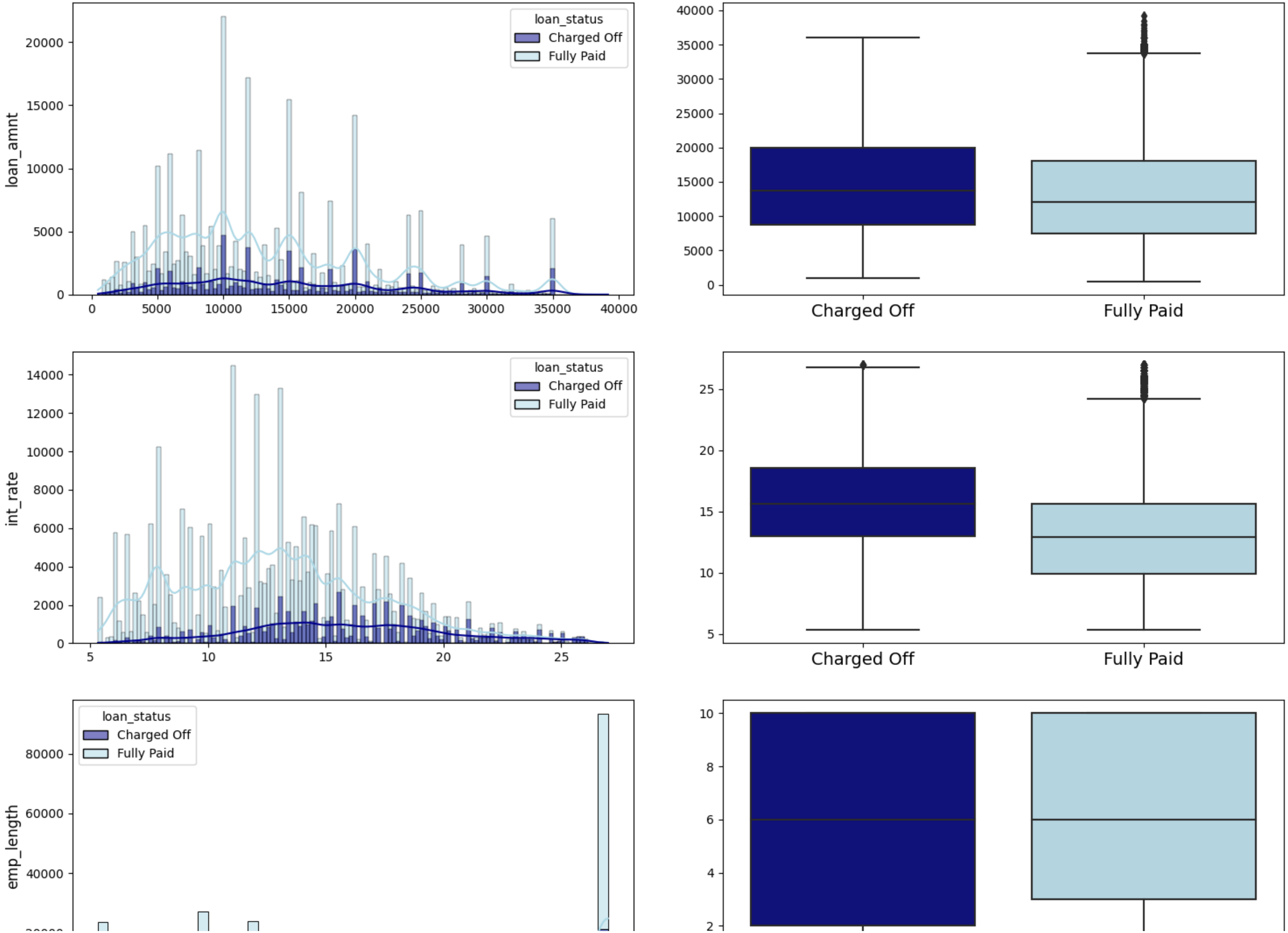

```
# Impact of Purpose/state on loan status

purpose = df.pivot_table(index='purpose', columns='loan_status', aggfunc='count', values='sub_grade')
purpose = purpose.div(purpose.sum(axis=1), axis=0).multiply(100).round()
purpose.reset_index(inplace=True)

plt.figure(figsize=(14,4))
plt.bar(purpose['purpose'],purpose['Charged Off'], color='#00008b')
plt.bar(purpose['purpose'],purpose['Fully Paid'], color='#add8e6', bottom=purpose['Charged Off'])
plt.xlabel('Purpose')
plt.ylabel('% Applicants')
plt.title('% Defaulters by purpose')
plt.legend(['Charged Off','Fully Paid'])
plt.xticks(rotation=90)
plt.show()

state = df.pivot_table(index='state', columns='loan_status', aggfunc='count', values='sub_grade')
state = state.div(state.sum(axis=1), axis=0).multiply(100).round()
state.reset_index(inplace=True)

plt.figure(figsize=(14,4))
plt.bar(state['state'],state['Charged Off'], color='#00008b')
plt.bar(state['state'],state['Fully Paid'], color='#add8e6', bottom=state['Charged Off'])
plt.xlabel('state')
plt.ylabel('% Applicants')
plt.title('% Defaulters by state')
plt.legend(['Charged Off','Fully Paid'])
plt.show()
```

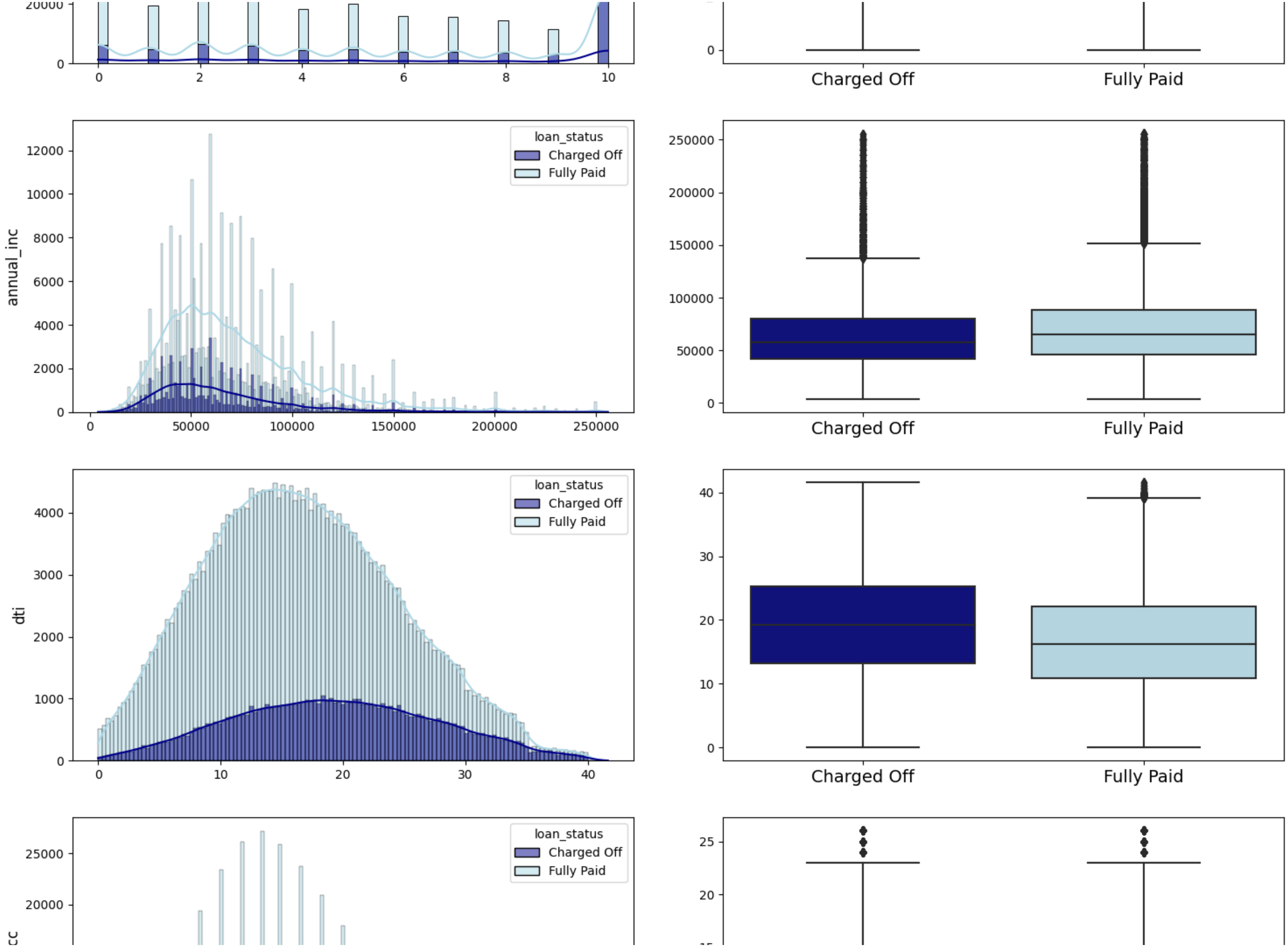


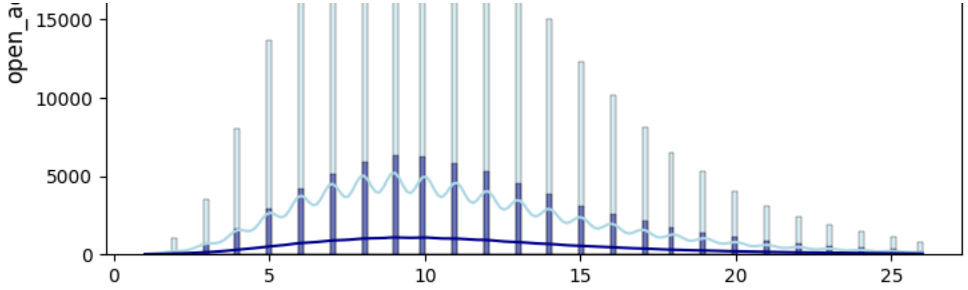


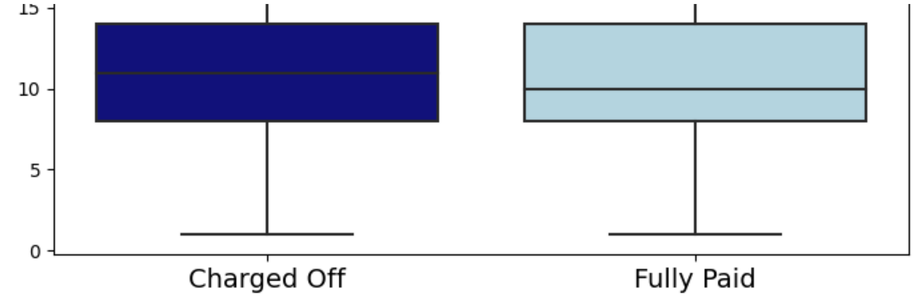

```
# Here are some key observations from the dataset:

# - Default rates are significantly higher for loans with longer (60-month) terms.
# - The grade and sub-grade of the loan have the most significant impact on loan_status, with higher grades having more defaulters.
# - Certain zip codes, such as 11650, 86630, and 93700, have a 100% default rate, suggesting a high risk associated with these areas.
# - It's advisable to remove "initial_list_status" and "state" variables as they do not appear to have any impact on loan_status.
# - Surprisingly, the presence of public records does not seem to have a substantial impact on loan_status.
# - Loans with the "Direct pay" application type have a higher default rate compared to "individual" or "joint" application types.
# - Loans taken for the purpose of "small business" have the highest default rate, indicating a higher risk associated with such loans.
```

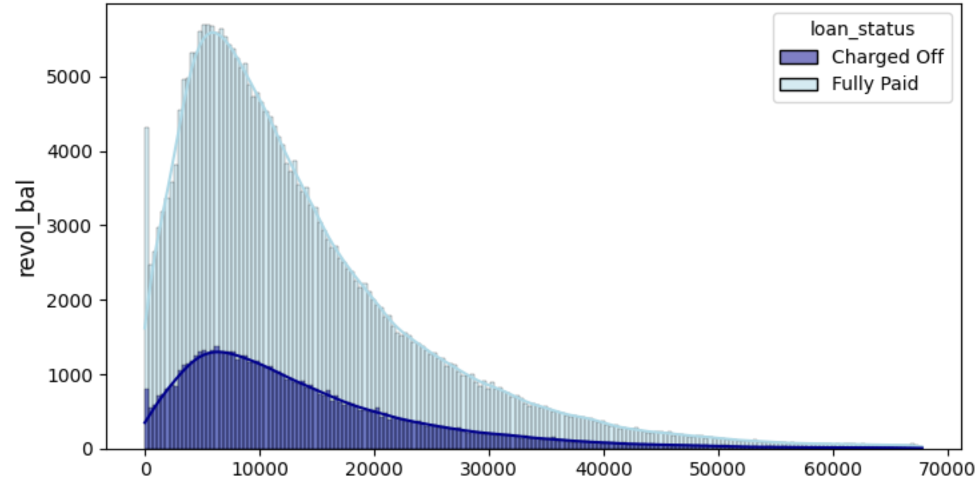
```
# Impact of numerical features on loan_status

num_cols = df.select_dtypes(include='number').columns

fig, ax = plt.subplots(10,2,figsize=(15,40))
i=0
color_dict = {'Fully Paid': matplotlib.colors.to_rgba('#add8e6', 0.5),
              'Charged Off': matplotlib.colors.to_rgba('#00008b', 1)}
for col in num_cols:
    sns.histplot(data=df, x=col, hue='loan_status', ax=ax[i, 0], legend=True,
                 palette=color_dict, kde=True, fill=True)
    sns.boxplot(data=df, y=col, x='loan_status', ax=ax[i,1],
                palette=('#00008b', '#add8e6'))
    ax[i,0].set_ylabel(col, fontsize=12)
    ax[i,0].set_xlabel(' ')
    ax[i,1].set_xlabel(' ')
    ax[i,1].set_ylabel(' ')
    ax[i,1].xaxis.set_tick_params(labelsize=14)
    i += 1

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

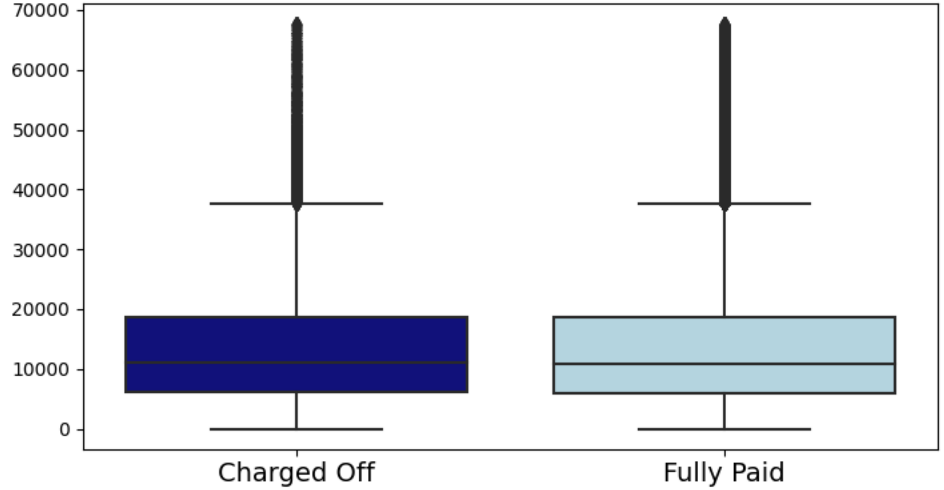

loan_amnt
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
int_rate
loan_status
Charged Off
Fully Paid
Charged Off
Fully Paid
emp_length
loan_status
Charged Off
Fully Paid

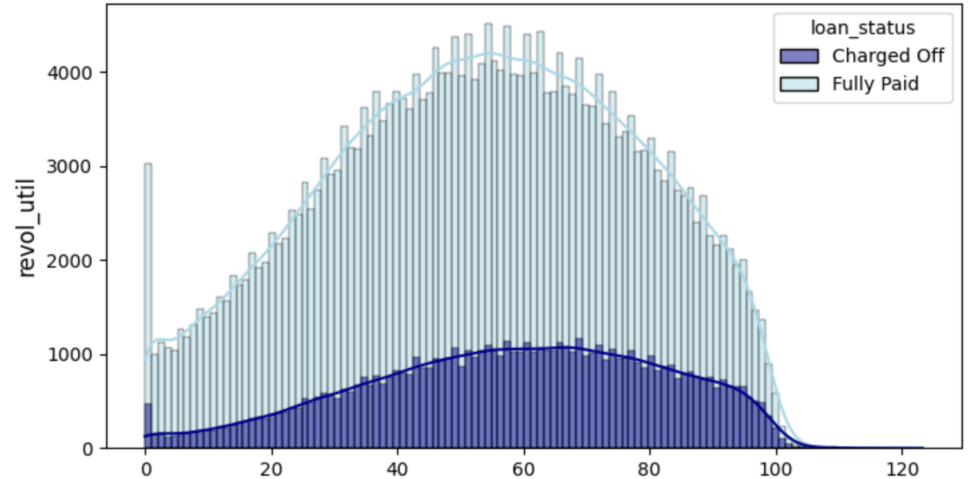

Charged Off
Fully Paid
loan_status
Charged Off
Fully Paid
annual_inc
Charged Off
Fully Paid
loan_status
Charged Off
Fully Paid
dti
Charged Off
Fully Paid
loan_status
Charged Off
Fully Paid

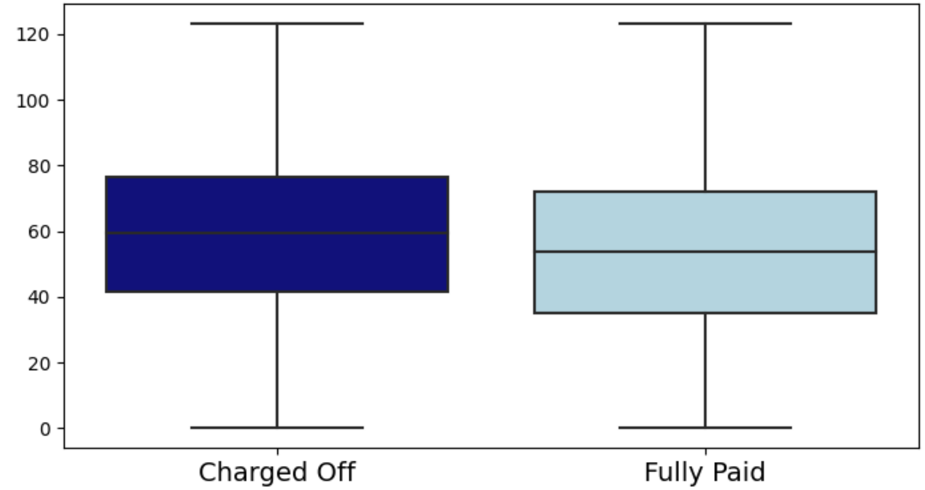

open_a
15000
10000
5000
0
0
5
10
15
20
25


15
10
5
0
Charged Off
Fully Paid


loan_status
Charged Off
Fully Paid
revol_bal
5000
4000
3000
2000
1000
0
0
10000
20000
30000
40000
50000
60000
70000

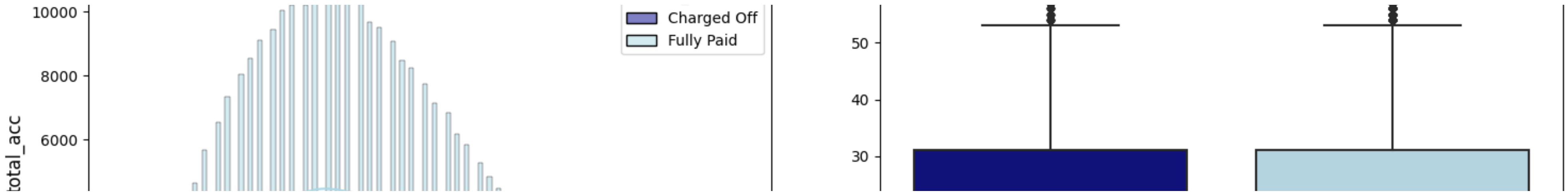

70000
60000
50000
40000
30000
20000
10000
0
Charged Off
Fully Paid


loan_status
Charged Off
Fully Paid
revol_util
4000
3000
2000
1000
0
0
20
40
60
80
100
120


120
100
80
60
40
20
0
Charged Off
Fully Paid


loan_status


60




```
# Here's an observation based on the boxplots:

# - Defaulters tend to have slightly higher mean values for "loan_amnt," "int_rate," "dti," "open_acc," and "revol_util,"
# while their annual income is lower compared to non-defaulters.

# Remove columns which do not have an impact on loan_status
df.drop(columns=['initial_list_status','state',
                 'emp_title', 'title','earliest_cr_line',
                 'issue_d','sub_grade'], inplace=True)

# Subgrade is removed because grade and subgrade are similar features

# Data Pre-Processing

# Encoding Target Variable

df['loan_status']=df['loan_status'].map({'Fully Paid': 0, 'Charged Off':1}).astype(int)

x = df.drop(columns=['loan_status'])
x.reset_index(inplace=True, drop=True)
y = df['loan_status']
y.reset_index(drop=True, inplace=True)

# Encoding Binary features into numerical dtype

x['term']=x['term'].map({' 36 months': 36, ' 60 months':60}).astype(int)
x['pub_rec']=x['pub_rec'].map({'no': 0, 'yes':1}).astype(int)
x['pub_rec_bankruptcies']=x['pub_rec_bankruptcies'].map({'no': 0, 'yes':1}).astype(int)


# One Hot Encoding of Categorical Features


cat_cols = x.select_dtypes('category').columns

encoder = OneHotEncoder(sparse=False)
encoded_data = encoder.fit_transform(x[cat_cols])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(cat_cols))
x = pd.concat([x,encoded_df], axis=1)
x.drop(columns=cat_cols, inplace=True)
x.head()
```

	loan_amnt	term	int_rate	emp_length	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies	grade_A	grade_B	grade_C	grade_D	gra
0	10000.0	36	11.44	10.0	117000.0	26.24	16.0	0	36369.0	41.8	25.0	0.0	0	0.0	1.0	0.0	0.0	
1	8000.0	36	11.99	4.0	65000.0	22.05	17.0	0	20131.0	53.3	27.0	3.0	0	0.0	1.0	0.0	0.0	
2	15600.0	36	10.49	0.0	43057.0	12.79	13.0	0	11987.0	92.2	26.0	0.0	0	0.0	1.0	0.0	0.0	
3	7200.0	36	6.49	6.0	54000.0	2.60	6.0	0	5472.0	21.5	13.0	0.0	0	1.0	0.0	0.0	0.0	
4	24375.0	60	17.27	9.0	55000.0	33.95	13.0	0	24584.0	69.8	43.0	1.0	0	0.0	0.0	1.0	0.0	

```
# Train-Test Split
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.20,stratify=y,random_state=42)
```

```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

    ((280676, 56), (280676,), (70169, 56), (70169,))

```
# Scaling Numeric Features
```

```
scaler = MinMaxScaler()
x_train = pd.DataFrame(scaler.fit_transform(x_train), columns=x_train.columns)
x_test = pd.DataFrame(scaler.transform(x_test), columns=x_test.columns)
```

```
x_train.tail()
```

	loan_amnt	term	int_rate	emp_length	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies	grade_A	grade_B	grade_C	grade
280671	0.167959	0.0	0.141671	0.7	0.194444	0.255954	0.60	0.0	0.104275	0.271695	0.578947	0.428571	0.0	1.0	0.0	0.0	
280672	0.497416	0.0	0.445778	0.4	0.182540	0.414482	0.24	0.0	0.224536	0.670722	0.263158	0.285714	0.0	0.0	0.0	1.0	
280673	0.064599	0.0	0.686664	0.7	0.238095	0.220111	0.32	0.0	0.249454	0.622871	0.385965	0.428571	0.0	0.0	0.0	0.0	
280674	0.245478	1.0	0.177665	0.9	0.313492	0.134953	0.92	0.0	0.080701	0.039740	0.842105	0.428571	0.0	0.0	1.0	0.0	
280675	0.646641	1.0	0.885095	0.6	0.349206	0.747173	0.88	1.0	0.213775	0.543390	0.596491	0.714286	1.0	0.0	0.0	0.0	

```
# Oversampling with SMOTE
```

```
# Oversampling to balance the target variable

sm=SMOTE(random_state=42)
x_train_res, y_train_res = sm.fit_resample(x_train,y_train.ravel())

print(f"Before OverSampling, count of label 1: {sum(y_train == 1)}")
print(f"Before OverSampling, count of label 0: {sum(y_train == 0)}")
print(f"After OverSampling, count of label 1: {sum(y_train_res == 1)}")
print(f"After OverSampling, count of label 0: {sum(y_train_res == 0)}")
```

```
Before OverSampling, count of label 1: 54200
Before OverSampling, count of label 0: 226476
After OverSampling, count of label 1: 226476
After OverSampling, count of label 0: 226476
```

```
# Logistic Regression
```

```
model = LogisticRegression()
model.fit(x_train_res, y_train_res)
train_preds = model.predict(x_train)
test_preds = model.predict(x_test)

#Model Evaluation
print('Train Accuracy :', model.score(x_train, y_train).round(2))
print('Train F1 Score:',f1_score(y_train,train_preds).round(2))
print('Train Recall Score:',recall_score(y_train,train_preds).round(2))
print('Train Precision Score:',precision_score(y_train,train_preds).round(2))

print('\nTest Accuracy :',model.score(x_test,y_test).round(2))
print('Test F1 Score:',f1_score(y_test,test_preds).round(2))
print('Test Recall Score:',recall_score(y_test,test_preds).round(2))
print('Test Precision Score:',precision_score(y_test,test_preds).round(2))

# Confusion Matrix
cm = confusion_matrix(y_test, test_preds)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```

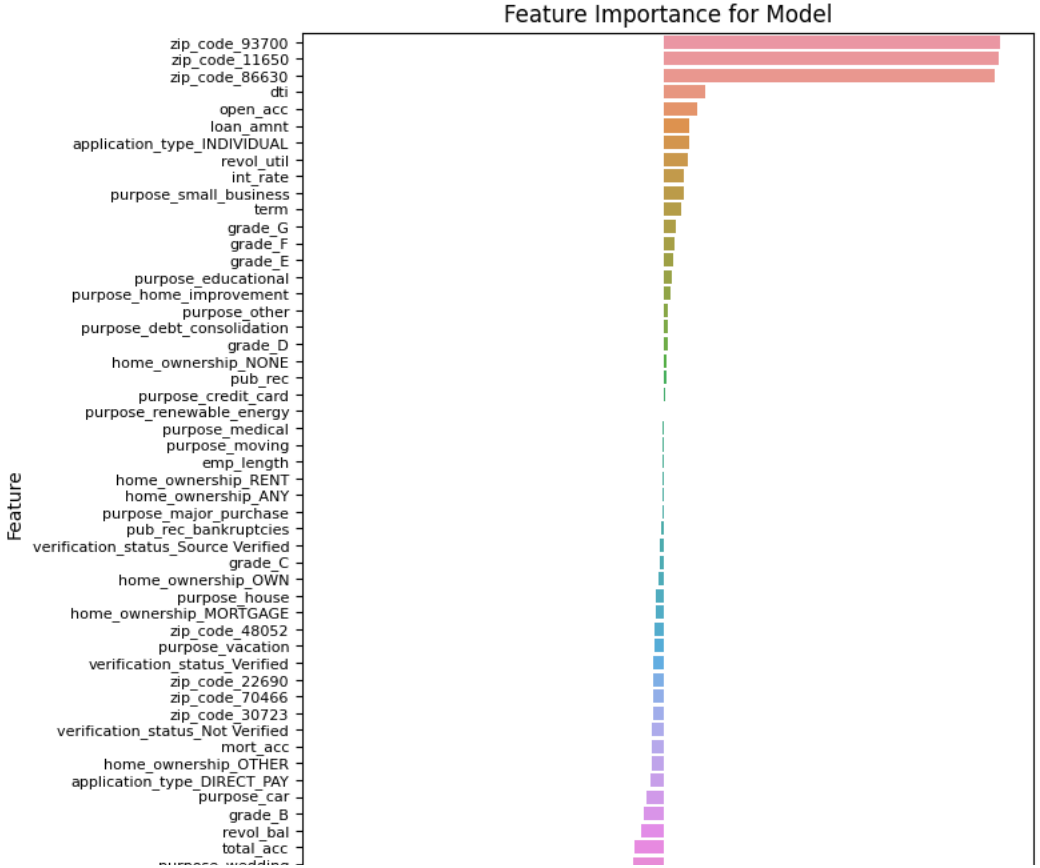
```
Train Accuracy : 0.8
Train F1 Score: 0.61
Train Recall Score: 0.81
Train Precision Score: 0.49

Test Accuracy : 0.8
Test F1 Score: 0.61
Test Recall Score: 0.81
Test Precision Score: 0.49
```

```
# Classification Report
```


```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87     56619
           1       0.49      0.81      0.61     13550

    accuracy                           0.80     70169
   macro avg       0.72      0.80      0.74     70169
weighted avg       0.86      0.80      0.82     70169
```

```
# Here are some key observations related to the model's performance metrics:

# - The recall score is high, indicating that the model can identify approximately 80% of actual defaulters, reducing the risk of missing
# potential default cases.

# - However, the precision for the positive class is low, meaning that among all the predicted defaulters, only 50% are actually defaulters.
# This may result in a significant number of false positives.

# - While the model is effective in reducing Non-Performing Assets (NPAs) by identifying most of the defaulters, it could lead to the denial of
# loans to deserving customers due to the high false positive rate, potentially causing a loss of business opportunities.

# - The low precision has also caused the F1 score to drop to 60%, indicating a trade-off between accuracy and precision in the model's
# performance evaluation.


feature_imp = pd.DataFrame({'Columns':x_train.columns, 'Coefficients':model.coef_[0]}).round(2).sort_values('Coefficients', ascending=False)

plt.figure(figsize=(8,8))
sns.barplot(y = feature_imp['Columns'],
           x =  feature_imp['Coefficients'])
plt.title("Feature Importance for Model")
plt.yticks(fontsize=8)
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```




```
# Here are some key observations regarding the model's feature importance and coefficients:

# - The model assigns significant weightage to the "zip_code" features, indicating that they play a crucial role in predicting loan default.

# - After "zip_code," the features with the next highest weightage are "dti," "open_acc," and "loan_amnt," suggesting their importance in
# the model's predictions.

# - On the other hand, certain "zip_code" values have large negative coefficients, which implies that loans associated with these specific
# zip codes are less likely to default.

# - Additionally, "annual income" and the "joint" application type feature also have large negative coefficients, indicating that higher income
# levels and joint applications reduce the likelihood of loan default.


# **ROC Curve & AUC**

# The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model.
# It helps evaluate and compare different models by illustrating the trade-off between the true positive rate (TPR) and false positive rate (FPR)
# at various classification thresholds.

# The ROC curve is created by plotting the TPR on the y-axis against the FPR on the x-axis for different threshold values.
```

```
# * TPR: Also known as sensitivity or recall, is the proportion of true positive predictions out of all actual positive instances.
# * FPR: Proportion of false positive predictions out of all actual negative instances.

# A perfect classifier would have a TPR of 1 and an FPR of 0, resulting in a point at the top-left corner of the ROC curve. On the other hand,
# a random classifier would have an ROC curve following the diagonal line, as it has an equal chance of producing true positive and false positive
# predictions.

# The area under the ROC curve (AUC) is a commonly used metric to quantify the overall performance of a classifier.

# A perfect classifier would have an AUC of 1, while a random classifier would have an AUC of 0.5. The higher the AUC value, the better the classifier's
# performance in distinguishing between positive and negative instances.


# Predict probabilities for the test set
probs = model.predict_proba(x_test)[:,1]

# Compute the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```