In [ ]:

```
### Import necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import pylab

%matplotlib inline
from scipy import stats


from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, accuracy_score


# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
```

Reading Loan Data Set

In [ ]:

```
loandf =pd.read_csv("./loan.csv", index_col=None, na_values=['NA'],sep=',',low_memory=F
loandf.head(10)
```

Out[ ]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	162.87	E
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	59.83	C
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	84.33	C
3	1076863	1277178	10000	10000	10000.0	36 months	13.49%	339.31	C
4	1075358	1311748	3000	3000	3000.0	60 months	12.69%	67.79	E
5	1075269	1311441	5000	5000	5000.0	36 months	7.90%	156.46	A
6	1069639	1304742	7000	7000	7000.0	60 months	15.96%	170.08	C
7	1072053	1288686	3000	3000	3000.0	36 months	18.64%	109.43	E
8	1071795	1306957	5600	5600	5600.0	60 months	21.28%	152.39	F
9	1071570	1306721	5375	5375	5350.0	60	12.69%	121.45	E

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade
						months			

10 rows × 111 columns

View the dimensions of the dataframe to get an idea about the dataset

In [ ]:

```
loandf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Columns: 111 entries, id to total_il_high_credit_limit
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB
```

In [ ]:

```
loandf.describe
```

Out[ ]:

```
<bound method NDFrame.describe of             id  member_id  loan_amnt  funded_amnt  fun
ded_amnt_inv  \
0      1077501    1296599       5000         5000          4975.0
1      1077430    1314167       2500         2500          2500.0
2      1077175    1313524       2400         2400          2400.0
3      1076863    1277178      10000        10000         10000.0
4      1075358    1311748       3000         3000          3000.0
...        ...        ...        ...          ...            ...
39712    92187      92174       2500         2500          1075.0
39713    90665      90607       8500         8500           875.0
39714    90395      90390       5000         5000          1325.0
39715    90376      89243       5000         5000           650.0
39716    87023      86999       7500         7500           800.0

             term int_rate  installment grade sub_grade  ...   \
0       36 months   10.65%       162.87     B        B2  ...
1       60 months   15.27%        59.83     C        C4  ...
2       36 months   15.96%        84.33     C        C5  ...
3       36 months   13.49%       339.31     C        C1  ...
4       60 months   12.69%        67.79     B        B5  ...
...           ...      ...          ...   ...       ...  ...
39712   36 months    8.07%        78.42     A        A4  ...
39713   36 months   10.28%       275.38     C        C1  ...
39714   36 months    8.07%       156.84     A        A4  ...
39715   36 months    7.43%       155.38     A        A2  ...
39716   36 months   13.75%       255.43     E        E2  ...

      num_tl_90g_dpd_24m num_tl_op_past_12m pct_tl_nvr_dlq  percent_bc_gt_75  \
0                    NaN                NaN            NaN               NaN
1                    NaN                NaN            NaN               NaN
2                    NaN                NaN            NaN               NaN
3                    NaN                NaN            NaN               NaN
4                    NaN                NaN            NaN               NaN
...                  ...                ...            ...               ...
39712                NaN                NaN            NaN               NaN
39713                NaN                NaN            NaN               NaN
39714                NaN                NaN            NaN               NaN
39715                NaN                NaN            NaN               NaN
39716                NaN                NaN            NaN               NaN

      pub_rec_bankruptcies tax_liens tot_hi_cred_lim total_bal_ex_mort  \
0                      0.0       0.0             NaN               NaN
```

```
1                      0.0       0.0              NaN                NaN
2                      0.0       0.0              NaN                NaN
3                      0.0       0.0              NaN                NaN
4                      0.0       0.0              NaN                NaN
...                    ...       ...              ...                ...
39712                  NaN       NaN              NaN                NaN
39713                  NaN       NaN              NaN                NaN
39714                  NaN       NaN              NaN                NaN
39715                  NaN       NaN              NaN                NaN
39716                  NaN       NaN              NaN                NaN

      total_bc_limit total_il_high_credit_limit
0                NaN                        NaN
1                NaN                        NaN
2                NaN                        NaN
3                NaN                        NaN
4                NaN                        NaN
...              ...                        ...
39712            NaN                        NaN
39713            NaN                        NaN
39714            NaN                        NaN
39715            NaN                        NaN
39716            NaN                        NaN

[39717 rows x 111 columns]>
```

In [ ]:

```
loandf.shape
```

Out[ ]:

```
(39717, 111)
```

Identifying missing data

In [ ]:

```
loandf.isnull().sum()
```

Out[ ]:

```
id                                0
member_id                         0
loan_amnt                         0
funded_amnt                       0
funded_amnt_inv                   0
                              ...  
tax_liens                        39
tot_hi_cred_lim               39717
total_bal_ex_mort             39717
total_bc_limit                39717
total_il_high_credit_limit    39717
Length: 111, dtype: int64
```

Check for NA values in dataset

In [ ]:

```
loandf.isnull().sum()*100/loandf.shape[0]
```

Out[ ]:

```
id                              0.000000
member_id                       0.000000
loan_amnt                       0.000000
funded_amnt                     0.000000
funded_amnt_inv                 0.000000
                                 ...    
tax_liens                       0.098195
tot_hi_cred_lim               100.000000
total_bal_ex_mort             100.000000
total_bc_limit                100.000000
```

```
total_il_high_credit_limit    100.000000
Length: 111, dtype: float64
```

Perform Data Cleanup

Drop all the column having 100% null values

In [ ]:
```
loandf = loandf.dropna(axis=1, how='all')
```

Check the % of NAs columnwise

In [ ]:
```
loandf.isnull().sum()*100/loandf.shape[0]
```

Out[ ]:
```
id                           0.000000
member_id                    0.000000
loan_amnt                    0.000000
funded_amnt                  0.000000
funded_amnt_inv              0.000000
term                         0.000000
int_rate                     0.000000
installment                  0.000000
grade                        0.000000
sub_grade                    0.000000
emp_title                    6.191303
emp_length                   2.706650
home_ownership               0.000000
annual_inc                   0.000000
verification_status          0.000000
issue_d                      0.000000
loan_status                  0.000000
pymnt_plan                   0.000000
url                          0.000000
desc                        32.580507
purpose                      0.000000
title                        0.027696
zip_code                     0.000000
addr_state                   0.000000
dti                          0.000000
delinq_2yrs                  0.000000
earliest_cr_line             0.000000
inq_last_6mths               0.000000
mths_since_last_delinq      64.662487
mths_since_last_record      92.985372
open_acc                     0.000000
pub_rec                      0.000000
revol_bal                    0.000000
revol_util                   0.125891
total_acc                    0.000000
initial_list_status          0.000000
out_prncp                    0.000000
out_prncp_inv                0.000000
total_pymnt                  0.000000
total_pymnt_inv              0.000000
total_rec_prncp              0.000000
total_rec_int                0.000000
total_rec_late_fee           0.000000
recoveries                   0.000000
collection_recovery_fee      0.000000
last_pymnt_d                 0.178765
last_pymnt_amnt              0.000000
```

```
next_pymnt_d               97.129693
last_credit_pull_d          0.005036
collections_12_mths_ex_med  0.140998
policy_code                 0.000000
application_type            0.000000
acc_now_delinq              0.000000
chargeoff_within_12_mths    0.140998
delinq_amnt                 0.000000
pub_rec_bankruptcies        1.754916
tax_liens                   0.098195
dtype: float64
```

Identify the columns having 50% or more null values and remove such columns

In [ ]:

```
loandf = loandf.dropna(thresh=len(loandf) * 0.5, axis=1)
```

In [ ]:

```
loandf.shape
```

Out[ ]:

```
(39717, 54)
```

In [ ]:

```
loandf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 54 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          39717 non-null  int64
 1   member_id                   39717 non-null  int64
 2   loan_amnt                   39717 non-null  int64
 3   funded_amnt                 39717 non-null  int64
 4   funded_amnt_inv             39717 non-null  float64
 5   term                        39717 non-null  object
 6   int_rate                    39717 non-null  object
 7   installment                 39717 non-null  float64
 8   grade                       39717 non-null  object
 9   sub_grade                   39717 non-null  object
 10  emp_title                   37258 non-null  object
 11  emp_length                  38642 non-null  object
 12  home_ownership              39717 non-null  object
 13  annual_inc                  39717 non-null  float64
 14  verification_status         39717 non-null  object
 15  issue_d                     39717 non-null  object
 16  loan_status                 39717 non-null  object
 17  pymnt_plan                  39717 non-null  object
 18  url                         39717 non-null  object
 19  desc                        26777 non-null  object
 20  purpose                     39717 non-null  object
 21  title                       39706 non-null  object
 22  zip_code                    39717 non-null  object
 23  addr_state                  39717 non-null  object
 24  dti                         39717 non-null  float64
 25  delinq_2yrs                 39717 non-null  int64
 26  earliest_cr_line            39717 non-null  object
 27  inq_last_6mths              39717 non-null  int64
 28  open_acc                    39717 non-null  int64
 29  pub_rec                     39717 non-null  int64
 30  revol_bal                   39717 non-null  int64
 31  revol_util                  39667 non-null  object
 32  total_acc                   39717 non-null  int64
```

```
 33  initial_list_status         39717 non-null  object
 34  out_prncp                   39717 non-null  float64
 35  out_prncp_inv               39717 non-null  float64
 36  total_pymnt                 39717 non-null  float64
 37  total_pymnt_inv             39717 non-null  float64
 38  total_rec_prncp             39717 non-null  float64
 39  total_rec_int               39717 non-null  float64
 40  total_rec_late_fee          39717 non-null  float64
 41  recoveries                  39717 non-null  float64
 42  collection_recovery_fee     39717 non-null  float64
 43  last_pymnt_d                39646 non-null  object
 44  last_pymnt_amnt             39717 non-null  float64
 45  last_credit_pull_d          39715 non-null  object
 46  collections_12_mths_ex_med  39661 non-null  float64
 47  policy_code                 39717 non-null  int64
 48  application_type            39717 non-null  object
 49  acc_now_delinq              39717 non-null  int64
 50  chargeoff_within_12_mths    39661 non-null  float64
 51  delinq_amnt                 39717 non-null  int64
 52  pub_rec_bankruptcies        39020 non-null  float64
 53  tax_liens                   39678 non-null  float64
dtypes: float64(18), int64(13), object(23)
memory usage: 16.4+ MB
```

In [ ]:

```
sum(loandf.duplicated(subset = "id")) == 0
```

Out[ ]:

```
True
```

Creating Loan Period as a derived variable from term column as Numeric variable

In [ ]:

```
loandf['loanPeriod'] = loandf['term'].str[1:4].astype(int)
loandf.head(10)
```

Out[ ]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	162.87	B
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	59.83	C
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	84.33	C
3	1076863	1277178	10000	10000	10000.0	36 months	13.49%	339.31	C
4	1075358	1311748	3000	3000	3000.0	60 months	12.69%	67.79	B
5	1075269	1311441	5000	5000	5000.0	36 months	7.90%	156.46	A
6	1069639	1304742	7000	7000	7000.0	60 months	15.96%	170.08	C
7	1072053	1288686	3000	3000	3000.0	36 months	18.64%	109.43	E
8	1071795	1306957	5600	5600	5600.0	60 months	21.28%	152.39	F

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade
9	1071570	1306721	5375	5375	5350.0	60 months	12.69%	121.45	E

10 rows × 55 columns

Create a derived attribute zip_code_num based on column zip_code which contain only numeric value.

In [ ]:
```
loandf["zip_code_num"] = loandf["zip_code"].str.replace('x','')
loandf["zip_code_num"] = loandf["zip_code_num"].astype(int)
```

Identify the unique value counts in the dataframe

In [ ]:
```
loandf.nunique()
```

Out[ ]:
```
id                          39717
member_id                   39717
loan_amnt                     885
funded_amnt                  1041
funded_amnt_inv              8205
term                            2
int_rate                      371
installment                 15383
grade                           7
sub_grade                      35
emp_title                   28820
emp_length                     11
home_ownership                  5
annual_inc                   5318
verification_status             3
issue_d                        55
loan_status                     3
pymnt_plan                      1
url                         39717
desc                        26527
purpose                        14
title                       19615
zip_code                      823
addr_state                     50
dti                          2868
delinq_2yrs                    11
earliest_cr_line              526
inq_last_6mths                  9
open_acc                       40
pub_rec                         5
revol_bal                   21711
revol_util                   1089
total_acc                      82
initial_list_status             1
out_prncp                    1137
out_prncp_inv                1138
total_pymnt                 37850
total_pymnt_inv             37518
total_rec_prncp              7976
total_rec_int               35148
total_rec_late_fee           1356
recoveries                   4040
```

```
collection_recovery_fee        2616
last_pymnt_d                    101
last_pymnt_amnt               34930
last_credit_pull_d              106
collections_12_mths_ex_med        1
policy_code                       1
application_type                  1
acc_now_delinq                    1
chargeoff_within_12_mths          1
delinq_amnt                       1
pub_rec_bankruptcies              3
tax_liens                         1
loanPeriod                        2
zip_code_num                    823
dtype: int64
```

Remove columns with only 1 uniques value as it will not add much value to analysis

In [ ]:
```
loandf = loandf.loc[:, loandf.nunique() != 1]
```

Check the dimensions of the dataframe once again

In [ ]:
```
loandf.shape
```

Out[ ]:
```
(39717, 47)
```

Drop few columns 'emp_title', 'url' ,'desc', 'title', 'zip_code', 'term' as it does not add much value for the analysis

In [ ]:
```
loandf = loandf.drop([ 'emp_title', 'url' ,'desc', 'title', 'zip_code', 'term'], axis=1
```

View dataframe

In [ ]:
```
loandf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 41 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       39717 non-null  int64
 1   member_id                39717 non-null  int64
 2   loan_amnt                39717 non-null  int64
 3   funded_amnt              39717 non-null  int64
 4   funded_amnt_inv          39717 non-null  float64
 5   int_rate                 39717 non-null  object
 6   installment              39717 non-null  float64
 7   grade                    39717 non-null  object
 8   sub_grade                39717 non-null  object
 9   emp_length               38642 non-null  object
 10  home_ownership           39717 non-null  object
 11  annual_inc               39717 non-null  float64
 12  verification_status      39717 non-null  object
 13  issue_d                  39717 non-null  object
 14  loan_status              39717 non-null  object
 15  purpose                  39717 non-null  object
 16  addr_state               39717 non-null  object
 17  dti                      39717 non-null  float64
 18  delinq_2yrs              39717 non-null  int64
```

```
 19  earliest_cr_line         39717 non-null  object
 20  inq_last_6mths           39717 non-null  int64
 21  open_acc                 39717 non-null  int64
 22  pub_rec                  39717 non-null  int64
 23  revol_bal                39717 non-null  int64
 24  revol_util               39667 non-null  object
 25  total_acc                39717 non-null  int64
 26  out_prncp                39717 non-null  float64
 27  out_prncp_inv            39717 non-null  float64
 28  total_pymnt              39717 non-null  float64
 29  total_pymnt_inv          39717 non-null  float64
 30  total_rec_prncp          39717 non-null  float64
 31  total_rec_int            39717 non-null  float64
 32  total_rec_late_fee       39717 non-null  float64
 33  recoveries               39717 non-null  float64
 34  collection_recovery_fee  39717 non-null  float64
 35  last_pymnt_d             39646 non-null  object
 36  last_pymnt_amnt          39717 non-null  float64
 37  last_credit_pull_d       39715 non-null  object
 38  pub_rec_bankruptcies     39020 non-null  float64
 39  loanPeriod               39717 non-null  int64
 40  zip_code_num             39717 non-null  int64
dtypes: float64(15), int64(12), object(14)
memory usage: 12.4+ MB
```

Verify column earliest_cr_line -The month the borrower's earliest reported credit line was opened

In [ ]:
```
loandf['earliest_cr_line']
```

Out[ ]:
```
0        Jan-85
1        Apr-99
2        Nov-01
3        Feb-96
4        Jan-96
          ...
39712    Nov-90
39713    Dec-86
39714    Oct-98
39715    Nov-88
39716    Oct-03
Name: earliest_cr_line, Length: 39717, dtype: object
```

As this information does not add much value for our analysis we can drop this column

In [ ]:
```
loandf = loandf.drop(['earliest_cr_line'],axis =1)
```

In [ ]:
```
loandf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 40 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       39717 non-null  int64
 1   member_id                39717 non-null  int64
 2   loan_amnt                39717 non-null  int64
 3   funded_amnt              39717 non-null  int64
 4   funded_amnt_inv          39717 non-null  float64
 5   int_rate                 39717 non-null  object
 6   installment              39717 non-null  float64
 7   grade                    39717 non-null  object
```

```
 8   sub_grade                39717 non-null  object
 9   emp_length               38642 non-null  object
 10  home_ownership           39717 non-null  object
 11  annual_inc               39717 non-null  float64
 12  verification_status      39717 non-null  object
 13  issue_d                  39717 non-null  object
 14  loan_status              39717 non-null  object
 15  purpose                  39717 non-null  object
 16  addr_state               39717 non-null  object
 17  dti                      39717 non-null  float64
 18  delinq_2yrs              39717 non-null  int64
 19  inq_last_6mths           39717 non-null  int64
 20  open_acc                 39717 non-null  int64
 21  pub_rec                  39717 non-null  int64
 22  revol_bal                39717 non-null  int64
 23  revol_util               39667 non-null  object
 24  total_acc                39717 non-null  int64
 25  out_prncp                39717 non-null  float64
 26  out_prncp_inv            39717 non-null  float64
 27  total_pymnt              39717 non-null  float64
 28  total_pymnt_inv          39717 non-null  float64
 29  total_rec_prncp          39717 non-null  float64
 30  total_rec_int            39717 non-null  float64
 31  total_rec_late_fee       39717 non-null  float64
 32  recoveries               39717 non-null  float64
 33  collection_recovery_fee  39717 non-null  float64
 34  last_pymnt_d             39646 non-null  object
 35  last_pymnt_amnt          39717 non-null  float64
 36  last_credit_pull_d       39715 non-null  object
 37  pub_rec_bankruptcies     39020 non-null  float64
 38  loanPeriod               39717 non-null  int64
 39  zip_code_num             39717 non-null  int64
dtypes: float64(15), int64(12), object(13)
memory usage: 12.1+ MB
```

EDA

As the dataset contains the information about past loan applicants and whether they 'defaulted' or not. The aim is to identify patterns which indicate if a person is likely to default, which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc.

Here we need to identify and understand which consumer attributes and loan attributes influence the tendency of default.

We will analyze all customer and loan attribute and find the impact of this attribute on loan status, whether fully paid or defaulted

```
      Verify the loan status and count the number of records for each
status.
      Here Status Charged Off corresponds to loan default
```

In [ ]:

```
loandf['loan_status'].value_counts()
```

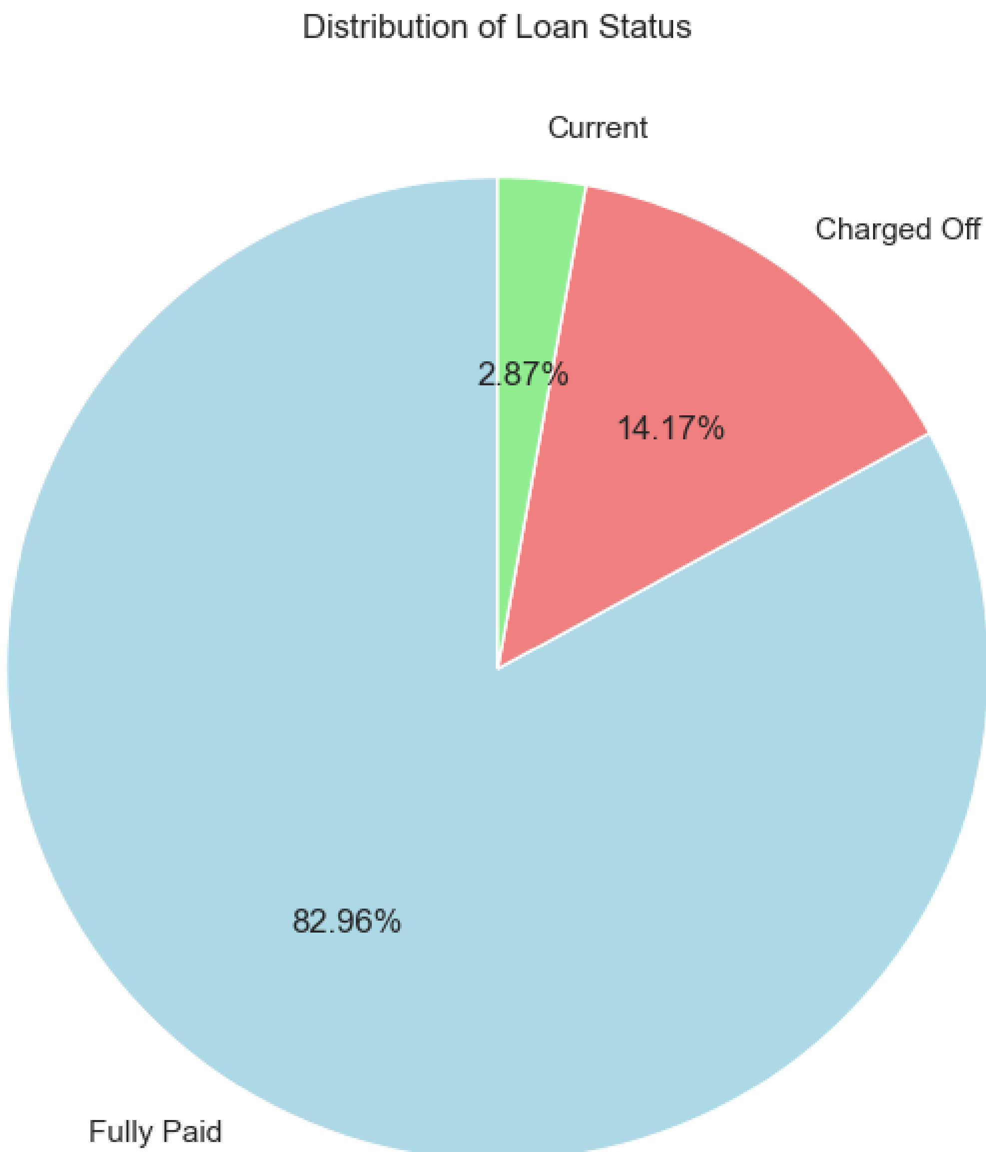
Out[ ]:

```
Fully Paid     32950
Charged Off     5627
```

```
Current        1140
Name: loan_status, dtype: int64
```

Pie Chart for distribution of loan status

In [ ]:
```
sns.set(style="whitegrid")


plt.figure(figsize=(8, 8))
loan_status_counts = loandf['loan_status'].value_counts()
plt.pie(loan_status_counts, labels=loan_status_counts.index, autopct='%1.2f%%', startan
plt.title('Distribution of Loan Status')
plt.show()
```



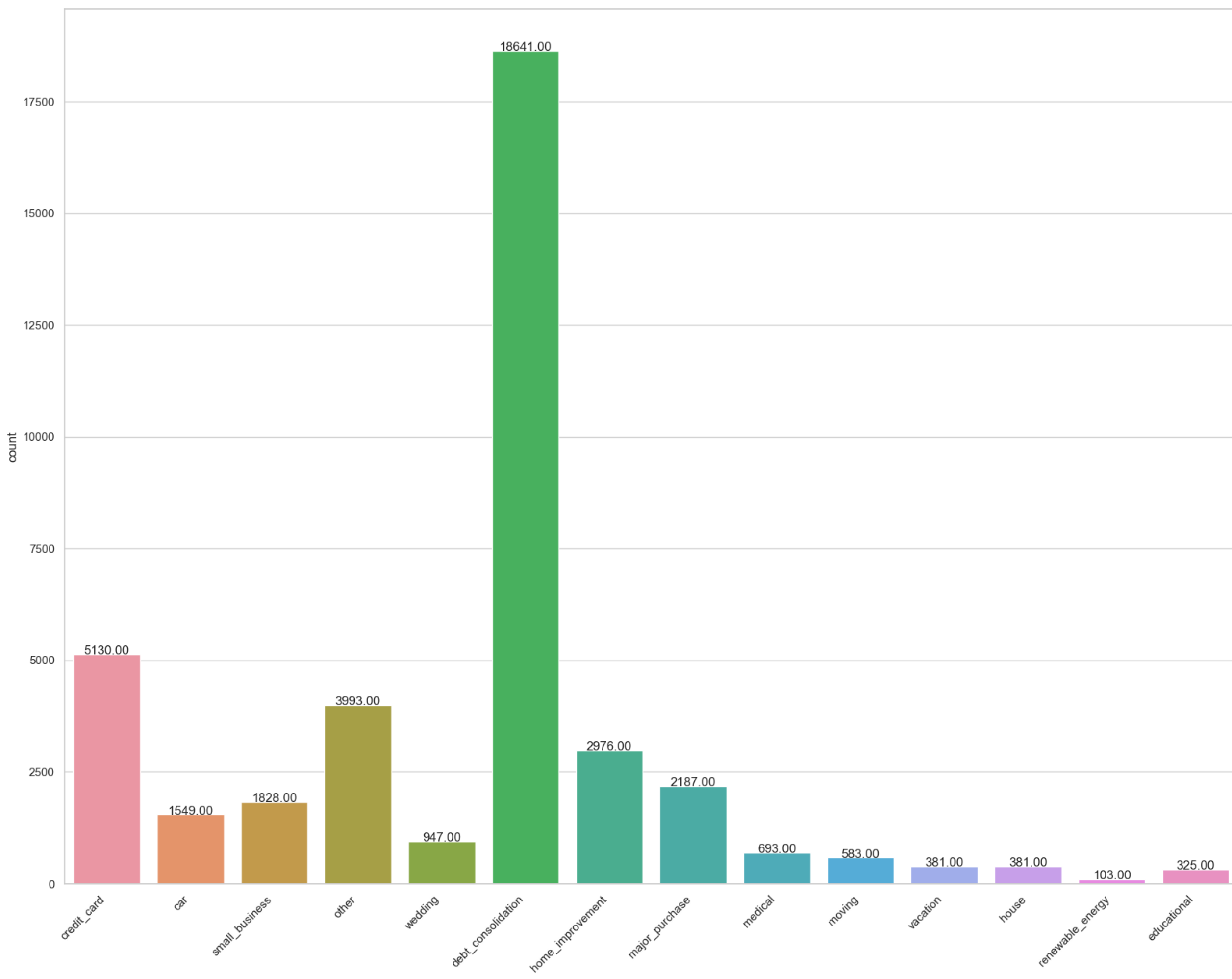
View the dataSet for loan purpose and find number of record for each purpose

In [ ]:
```
loandf['purpose'].value_counts()
```

Out[ ]:
```
debt_consolidation    18641
credit_card            5130
other                  3993
home_improvement       2976
major_purchase         2187
small_business         1828
car                    1549
wedding                 947
medical                 693
moving                  583
vacation                381
house                   381
educational             325
renewable_energy        103
Name: purpose, dtype: int64
```

Countplot for loan Purpose

In [ ]:
```
plt.figure(figsize=(20, 15))
ax = sns.countplot(x="purpose", data=loandf)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2., height + 0.5, '{:1.2f}'.format(height), ha="c

plt.show()
```



Lets check how loan purpose impacts the loan status

Create pivot table loandf_purpose from given loan dataset for Loan status and Loan Purpose

In [ ]:

```
loandf_purpose = pd.pivot_table(loandf, values='loan_amnt', index='purpose', columns='l
loandf_purpose
```

Out[ ]:

loan_status purpose	Charged Off	Current	Fully Paid
car	160.0	50.0	1339.0
credit_card	542.0	103.0	4485.0
debt_consolidation	2767.0	586.0	15288.0
educational	56.0	NaN	269.0
home_improvement	347.0	101.0	2528.0
house	59.0	14.0	308.0
major_purchase	222.0	37.0	1928.0
medical	106.0	12.0	575.0
moving	92.0	7.0	484.0
other	633.0	128.0	3232.0
renewable_energy	19.0	1.0	83.0
small_business	475.0	74.0	1279.0
vacation	53.0	6.0	322.0
wedding	96.0	21.0	830.0

Perform data cleaning for pivot table loandf_purpose

Replace the NaN value with 0

In [ ]:

```
loandf_purpose.loc[pd.isnull(loandf_purpose['Current']), ['Current']] = 0
loandf_purpose
```

Out[ ]:

loan_status purpose	Charged Off	Current	Fully Paid
car	160.0	50.0	1339.0
credit_card	542.0	103.0	4485.0
debt_consolidation	2767.0	586.0	15288.0
educational	56.0	0.0	269.0
home_improvement	347.0	101.0	2528.0
house	59.0	14.0	308.0
major_purchase	222.0	37.0	1928.0

loan_status purpose	Charged Off	Current	Fully Paid
medical	106.0	12.0	575.0
moving	92.0	7.0	484.0
other	633.0	128.0	3232.0
renewable_energy	19.0	1.0	83.0
small_business	475.0	74.0	1279.0
vacation	53.0	6.0	322.0
wedding	96.0	21.0	830.0

Adding new columns Aggregate and percentage of loan for each status for given purpose to pivot table

In [ ]:

```
loandf_purpose['Aggregate'] = loandf_purpose['Charged Off'] + loandf_purpose['Current']
loandf_purpose['Charged Off%'] = round(loandf_purpose['Charged Off']/loandf_purpose['Ag
loandf_purpose['Current%'] = round(loandf_purpose['Current']/loandf_purpose['Aggregate'
loandf_purpose['Fully Paid %'] = round(loandf_purpose['Fully Paid']/loandf_purpose['Agg
loandf_purpose
```

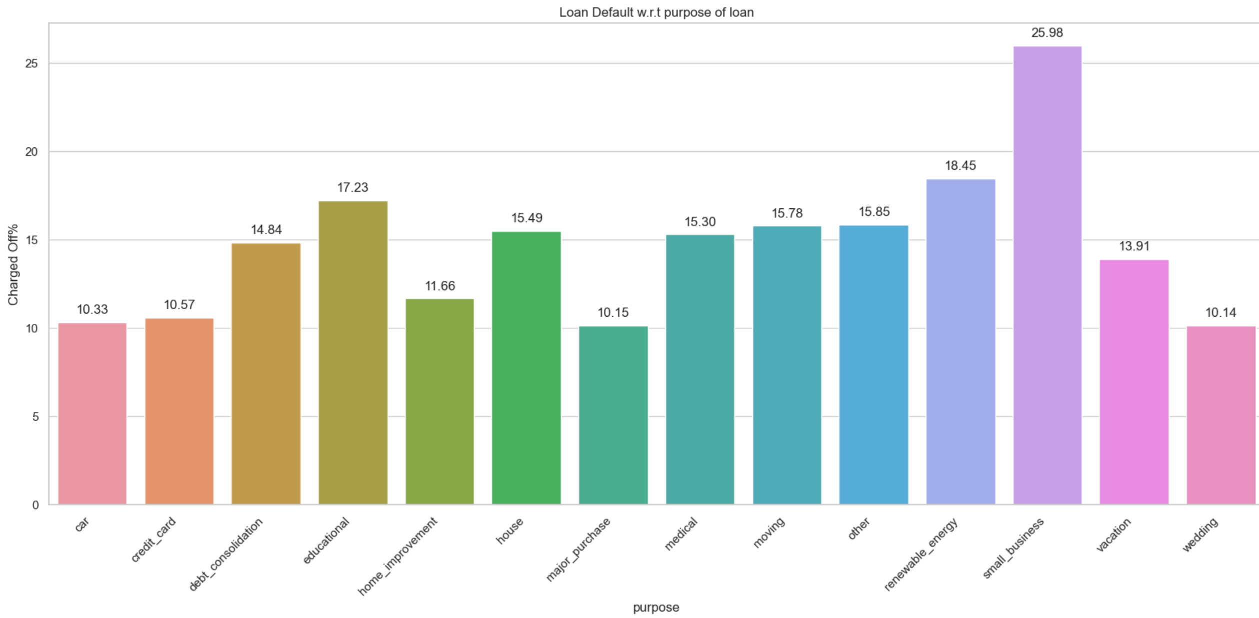
Out[ ]:

loan_status purpose	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
car	160.0	50.0	1339.0	1549.0	10.33	3.23	86.44
credit_card	542.0	103.0	4485.0	5130.0	10.57	2.01	87.43
debt_consolidation	2767.0	586.0	15288.0	18641.0	14.84	3.14	82.01
educational	56.0	0.0	269.0	325.0	17.23	0.00	82.77
home_improvement	347.0	101.0	2528.0	2976.0	11.66	3.39	84.95
house	59.0	14.0	308.0	381.0	15.49	3.67	80.84
major_purchase	222.0	37.0	1928.0	2187.0	10.15	1.69	88.16
medical	106.0	12.0	575.0	693.0	15.30	1.73	82.97
moving	92.0	7.0	484.0	583.0	15.78	1.20	83.02
other	633.0	128.0	3232.0	3993.0	15.85	3.21	80.94
renewable_energy	19.0	1.0	83.0	103.0	18.45	0.97	80.58
small_business	475.0	74.0	1279.0	1828.0	25.98	4.05	69.97
vacation	53.0	6.0	322.0	381.0	13.91	1.57	84.51
wedding	96.0	21.0	830.0	947.0	10.14	2.22	87.65

Barplot for loan purpose for percentage of loan charged off (Defaulted)

In [ ]:

```
plt.figure(figsize=(20, 8))
plt.title('Loan Default w.r.t purpose of loan')
ax=sns.barplot(x='purpose',y = "Charged Off%", data=loandf_purpose.reset_index())
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2., height + 0.5,'{:1.2f}'.format(height), ha="cent
```



From the above bar chat, loan for small_business contribute highest number of loan Default followed by renewable_energy

Verify the dataSet for loan period and find number of record for each period Values are in months and can be either 36 or 60.

In [ ]:

```
loandf['loanPeriod'].value_counts()
```

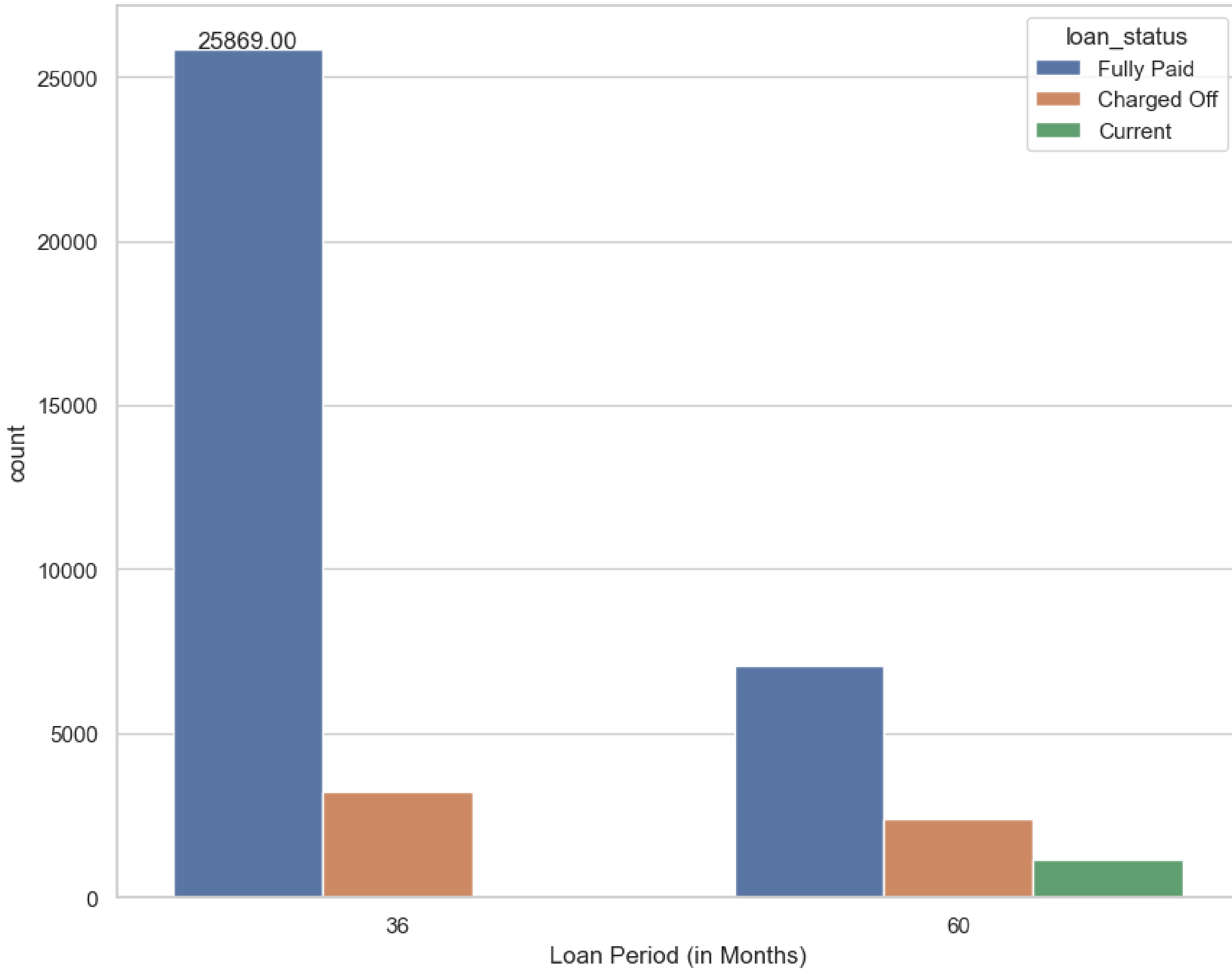
Out[ ]:

```
36    29096
60    10621
Name: loanPeriod, dtype: int64
```

Countplot for loan period based on loan status

In [ ]:

```
from matplotlib.pyplot import show
plt.figure(figsize=(10, 8))
ax=sns.countplot(x = "loanPeriod", hue = "loan_status", data = loandf)

for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2., height + 0.5,'{:1.2f}'.format(height), ha="cent
    ax.set_xlabel("Loan Period (in Months)")
    show()
```



Create Pivot table loandf_duration for the attribute loan period and loan status, which will help us analyze the impact of loan period on loan status

In [ ]:
```
loandf_duration = pd.pivot_table(loandf, values='loan_amnt', index='loanPeriod', column
loandf_duration
```

Out[ ]:

loan_status	Charged Off	Current	Fully Paid
loanPeriod			
36	3227.0	NaN	25869.0
60	2400.0	1140.0	7081.0

Convert NaN value in Pivot table to 0

In [ ]:
```
loandf_duration.loc[pd.isnull(loandf_duration['Current']), ['Current']] = 0
loandf_duration
```

Out[ ]:

loan_status	Charged Off	Current	Fully Paid
loanPeriod			
36	3227.0	0.0	25869.0
60	2400.0	1140.0	7081.0

Adding new columns Aggregate for aggregate number of loan for each duration and percentage of loan for each status for given duration to pivot table

In [ ]:

```
loandf_duration['Aggregate'] = loandf_duration['Charged Off'] + loandf_duration['Curren
loandf_duration['Charged Off%'] = round(loandf_duration['Charged Off']/loandf_duration[
loandf_duration['Current%'] = round(loandf_duration['Current']/loandf_duration['Aggrega
loandf_duration['Fully Paid %'] = round(loandf_duration['Fully Paid']/loandf_duration['


loandf_duration
```

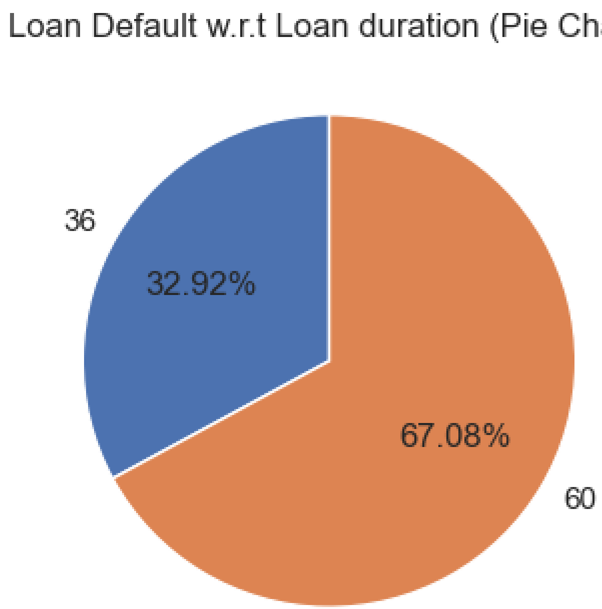
Out[ ]:

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
loanPeriod							
36	3227.0	0.0	25869.0	29096.0	11.09	0.00	88.91
60	2400.0	1140.0	7081.0	10621.0	22.60	10.73	66.67

Pie Chart for loan purpose for percentage of loan charged off (Defaulted)

In [ ]:

```
plt.figure(figsize=(6, 4))
plt.title('Loan Default w.r.t Loan duration')
ax = sns.barplot(x='loanPeriod', y="Charged Off%", data=loandf_duration.reset_index(),


for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.25, '{:1.2f}'.format(height), ha


labels = loandf_duration.reset_index()['loanPeriod'].tolist()
sizes = [p.get_height() for p in ax.patches]


total = sum(sizes)
sizes = [(size / total) * 100 for size in sizes]


plt.clf()
plt.pie(sizes, labels=labels, autopct='%1.2f%%', startangle=90)
plt.title('Loan Default w.r.t Loan duration (Pie Chart)')


plt.show()
```



Check the dataSet for dti and find the impact loan status based on dti range

Create new derived attribute dti_level for bucketing dti range

Here dti indicates - A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.

In [ ]:

```
def dti_level(x):
    'divide the time of the day into four categories'
    if x < 5:
        return "A(<5)"
    elif 5 <= x < 10:
        return "B(5-10)"
    elif 10 <= x < 15:
        return "C(10-15)"
    elif 15 <= x < 20:
        return "D(15-20)"
    else:
        return "E(>20)"

loandf['dti_level'] = loandf.dti.apply(lambda x: dti_level(x))
```

In [ ]:

```
loandf['dti_level'].value_counts()
```

Out[ ]:

```
C(10-15)    9893
D(15-20)    9108
B(5-10)     8062
E(>20)      7514
A(<5)       5140
Name: dti_level, dtype: int64
```

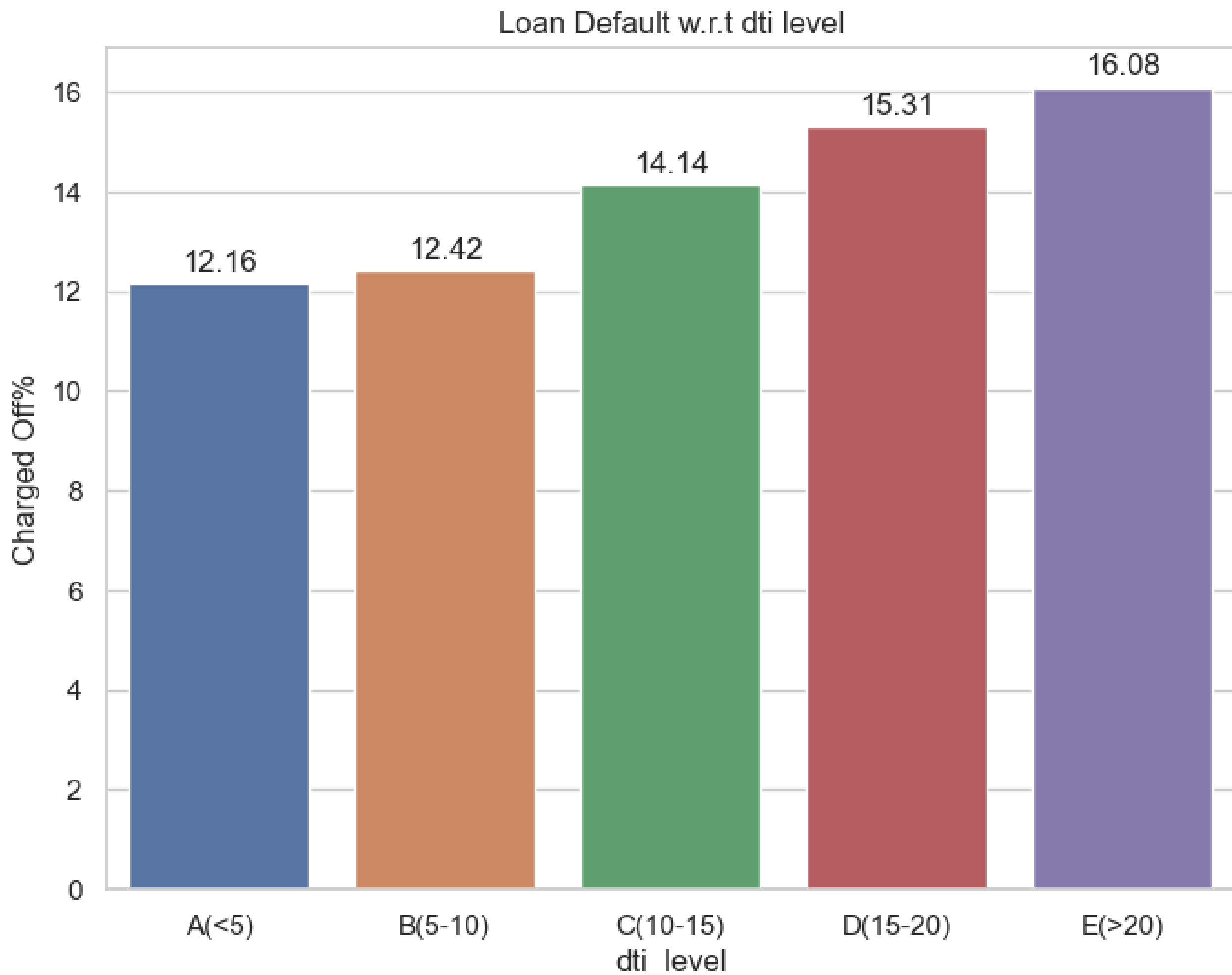
Create Pivot table loandf_dti for the attribute dti and loan status, which will help us analyze the impact of dti on loan status Adding new columns Aggregate for aggregate number of loan for each dti_level and loan status to pivot table

In [ ]:
```
loandf_dti = pd.pivot_table(loandf, values='loan_amnt', index='dti_level', columns='loa
loandf_dti['Aggregate'] = loandf_dti['Charged Off'] + loandf_dti['Current'] + loandf_dt
loandf_dti['Charged Off%'] = round(loandf_dti['Charged Off']/loandf_dti['Aggregate'] *1
loandf_dti['Current%'] = round(loandf_dti['Current']/loandf_dti['Aggregate'] *100, 2)
loandf_dti['Fully Paid %'] = round(loandf_dti['Fully Paid']/loandf_dti['Aggregate'] *10
loandf_dti
```

Out[ ]:

loan_status dti_level	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
A(<5)	625	96	4419	5140	12.16	1.87	85.97
B(5-10)	1001	201	6860	8062	12.42	2.49	85.09
C(10-15)	1399	269	8225	9893	14.14	2.72	83.14
D(15-20)	1394	284	7430	9108	15.31	3.12	81.58
E(>20)	1208	290	6016	7514	16.08	3.86	80.06

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.title('Loan Default w.r.t dti level')
ax=sns.barplot(x='dti_level',y = "Charged Off%", data=loandf_dti.reset_index())
#plt.show()
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2., height + 0.25,'{:1.2f}'.format(height),ha="cent
```



From the above bar chart, chance for loan default increases with increase in dti level

In [ ]:
```
loandf['total_acc'].value_counts()
```

Out[ ]:
```
16    1471
15    1462
17    1457
14    1445
20    1428
      ...
74       1
77       1
78       1
87       1
90       1
Name: total_acc, Length: 82, dtype: int64
```

```
Lets check the impact of total_acc on loan default
Here total_acc indicates the total number of credit lines currently in
the borrower's credit file.
```

Aggregate the 10 or more total_acc into one called 10+

In [ ]:
```
loandf['total_acc'] = loandf['total_acc'].apply(lambda x: x if x< 10 else '10+')
```

In [ ]:
```
loandf_total_acc = pd.pivot_table(loandf, values='loan_amnt', index='total_acc', column
loandf_total_acc
```

Out[ ]:

loan_status total_acc	Charged Off	Current	Fully Paid
2	1.0	NaN	3.0
3	42.0	3.0	137.0
4	79.0	5.0	336.0
5	91.0	9.0	452.0
6	107.0	9.0	567.0
7	132.0	15.0	681.0
8	172.0	17.0	817.0
9	166.0	24.0	890.0
10+	4837.0	1058.0	29067.0

Create Pivot table loandf_total_acc for the attribute total_acc and loan_status, which will help us analyze the impact of total_acc on loan status

Convert the NaN value to 0

Adding new columns Aggregate for aggregate number of total_acc for each loan_status and percentage of total_acc for each loan_status to pivot table

In [ ]:
```
loandf_total_acc.loc[pd.isnull(loandf_total_acc['Current']), ['Current']] = 0
loandf_total_acc['Aggregate'] = loandf_total_acc['Charged Off'] + loandf_total_acc['Cur
loandf_total_acc['Charged Off%'] = round(loandf_total_acc['Charged Off']/loandf_total_a
loandf_total_acc['Current%'] = round(loandf_total_acc['Current']/loandf_total_acc['Aggr
loandf_total_acc['Fully Paid %'] = round(loandf_total_acc['Fully Paid']/loandf_total_ac
loandf_total_acc
```

Out[ ]:

loan_status total_acc	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
2	1.0	0.0	3.0	4.0	25.00	0.00	75.00
3	42.0	3.0	137.0	182.0	23.08	1.65	75.27
4	79.0	5.0	336.0	420.0	18.81	1.19	80.00
5	91.0	9.0	452.0	552.0	16.49	1.63	81.88
6	107.0	9.0	567.0	683.0	15.67	1.32	83.02
7	132.0	15.0	681.0	828.0	15.94	1.81	82.25
8	172.0	17.0	817.0	1006.0	17.10	1.69	81.21
9	166.0	24.0	890.0	1080.0	15.37	2.22	82.41

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
total_acc							
10+	4837.0	1058.0	29067.0	34962.0	13.84	3.03	83.14

home_ownership - The home ownership status provided by the borrower during registration. The available values are: RENT, OWN, MORTGAGE, OTHER.

```
Lets check the impact of home_ownership on loan default
```

In [ ]:

```
loandf['home_ownership'].value_counts()
```

Out[ ]:

```
RENT        18899
MORTGAGE    17659
OWN          3058
OTHER          98
NONE            3
Name: home_ownership, dtype: int64
```

Combine NONE into OTHER category

In [ ]:

```
loandf['home_ownership'] = loandf['home_ownership'].apply(lambda x: x if x != 'NONE' el
loandf['home_ownership'].value_counts()
```

Out[ ]:

```
RENT        18899
MORTGAGE    17659
OWN          3058
OTHER         101
Name: home_ownership, dtype: int64
```

Create Pivot Table, handle NAN values and add new columns aggregate and percentage of loan_status for each home_ownership

In [ ]:

```
loandf_home_ownership = pd.pivot_table(loandf, values='loan_amnt', index='home_ownershi
loandf_home_ownership.loc[pd.isnull(loandf_home_ownership['Current']), ['Current']] = 0
loandf_home_ownership['Aggregate'] = loandf_home_ownership['Charged Off'] + loandf_home_
loandf_home_ownership['Charged Off%'] = round(loandf_home_ownership['Charged Off']/loan
loandf_home_ownership['Current%'] = round(loandf_home_ownership['Current']/loandf_home_
loandf_home_ownership['Fully Paid %'] = round(loandf_home_ownership['Fully Paid']/loand
loandf_home_ownership
```

Out[ ]:

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
home_ownership							
MORTGAGE	2327.0	638.0	14694.0	17659.0	13.18	3.61	83.21
OTHER	18.0	0.0	83.0	101.0	17.82	0.00	82.18
OWN	443.0	83.0	2532.0	3058.0	14.49	2.71	82.80
RENT	2839.0	419.0	15641.0	18899.0	15.02	2.22	82.76

```
Lets check the impact of annual income with loan default status
Here, annual_inc is the self-reported annual income provided by the
borrower during registration.
```

In [ ]:
```
print(loandf.annual_inc.max())
print(loandf.annual_inc.min())
```

```
6000000.0
4000.0
```

Apply bucketing for Annual income

In [ ]:
```
def sal_range(x):
    'divide the time of the day into four categories'
    if x < 10000:
        return "A(<10K)"
    elif 10000 <= x < 20000:
        return "B(10K-20K)"
    elif 20000 <= x < 50000:
        return "C(20K-50K)"
    elif 50000 <= x < 750000:
        return "D(50K-75K)"
    elif 75000 <= x < 100000:
        return "E(75K-100K)"
    else:
        return "F(>100K)"

loandf['salary_range'] = loandf.annual_inc.apply(lambda x: sal_range(x))
loandf['salary_range'].value_counts()
```

Out[ ]:
```
D(50K-75K)    24998
C(20K-50K)    13621
B(10K-20K)      986
A(<10K)          80
F(>100K)         32
Name: salary_range, dtype: int64
```

Create Pivot Table, handle NAN values and add new columns aggregate and percentage of losalary_range for each loan_status

In [ ]:
```
loandf_salary_range = pd.pivot_table(loandf, values='loan_amnt', index='salary_range',
loandf_salary_range.loc[pd.isnull(loandf_salary_range['Current']), ['Current']] = 0
loandf_salary_range['Aggregate'] = loandf_salary_range['Charged Off'] + loandf_salary_r
loandf_salary_range['Charged Off%'] = round(loandf_salary_range['Charged Off']/loandf_s
loandf_salary_range['Current%'] = round(loandf_salary_range['Current']/loandf_salary_ra
loandf_salary_range['Fully Paid %'] = round(loandf_salary_range['Fully Paid']/loandf_sa
loandf_salary_range
```

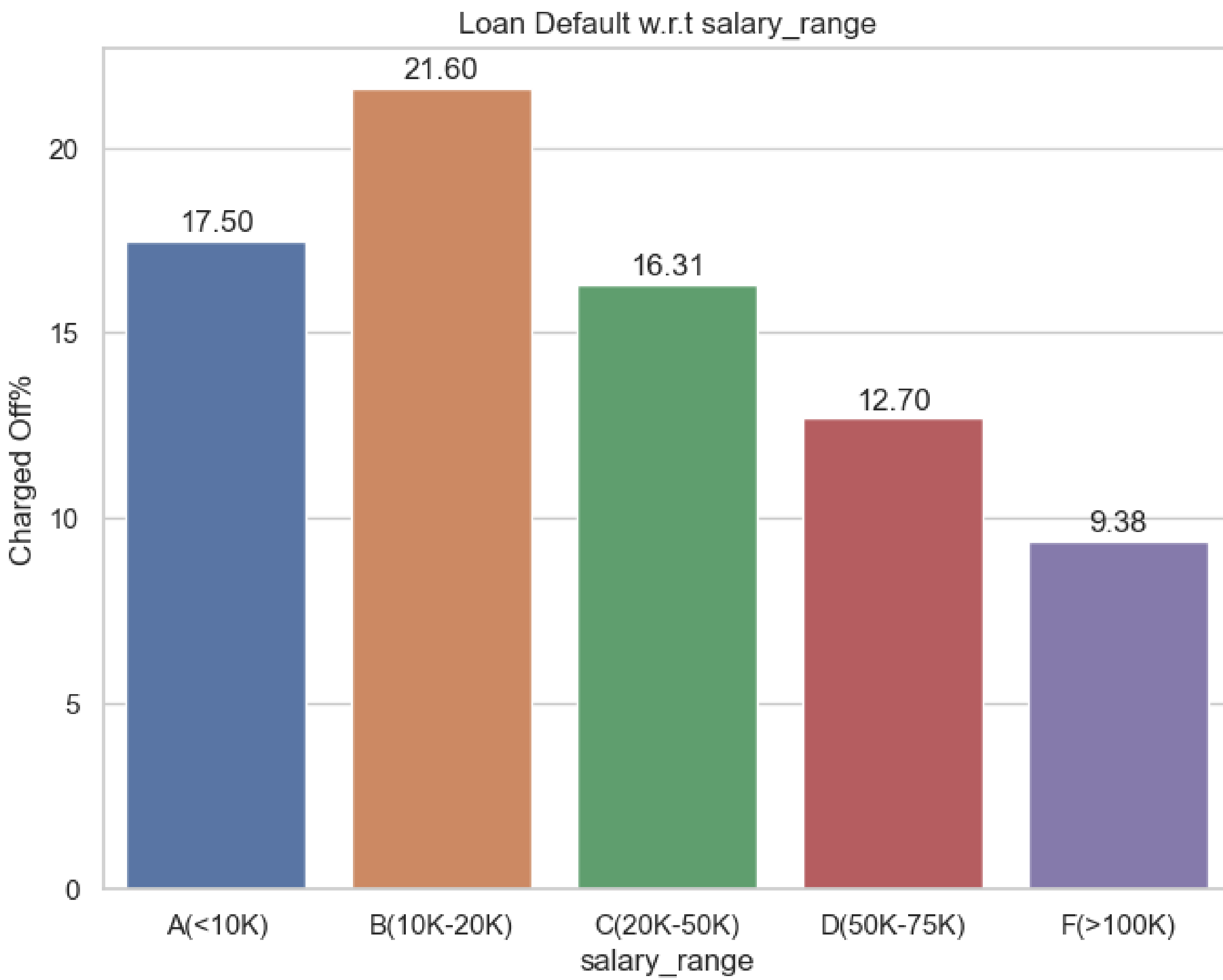
Out[ ]:

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
salary_range							
A(<10K)	14.0	1.0	65.0	80.0	17.50	1.25	81.25
B(10K-20K)	213.0	7.0	766.0	986.0	21.60	0.71	77.69
C(20K-50K)	2222.0	319.0	11080.0	13621.0	16.31	2.34	81.34

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
salary_range							
D(50K-75K)	3175.0	813.0	21010.0	24998.0	12.70	3.25	84.05
F(>100K)	3.0	0.0	29.0	32.0	9.38	0.00	90.62

In [ ]:

```
plt.figure(figsize=(8, 6))
plt.title('Loan Default w.r.t salary_range')
ax=sns.barplot(x= 'salary_range',y = "Charged Off%", data=loandf_salary_range.reset_ind
#plt.show()
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2., height + 0.25,'{:1.2f}'.format(height),ha="cent
```



From the above bar chart we can see that for salary range 10,000 to 20,000 there are more chances of defaulting the loan

Interest Rates Lets check the impact of Interest Rate on the loan on loan default

Convert int_rate to int_rate% as float from string and find the minimum and maximum values

In [ ]:

```
loandf['int_rate%'] = loandf['int_rate'].str[:-1].astype(float)
loandf = loandf.drop('int_rate', axis=1)
```

```
print(loandf['int_rate%'].max())
print(loandf['int_rate%'].min())
```

```
24.59
5.42
```

apply bucketing for int_rate%

In [ ]:
```
def int_rate(x):
    'Create int_rate range'
    if x < 10:
        return "A(<10%)"
    elif 10 <= x < 15:
        return "B(10%-15%)"
    elif 15 <= x < 20:
        return "C(15%-20%)"
    else:
        return "D(>20%)"

loandf['int_rate_range'] = loandf['int_rate%'].apply(lambda x: int_rate(x))
loandf['int_rate_range'].value_counts()
```

Out[ ]:
```
B(10%-15%)    19045
A(<10%)       12142
C(15%-20%)     7658
D(>20%)         872
Name: int_rate_range, dtype: int64
```

Create Pivot Table, handle NAN values and add new columns aggregate and percentage of int_rate_range for each loan_status

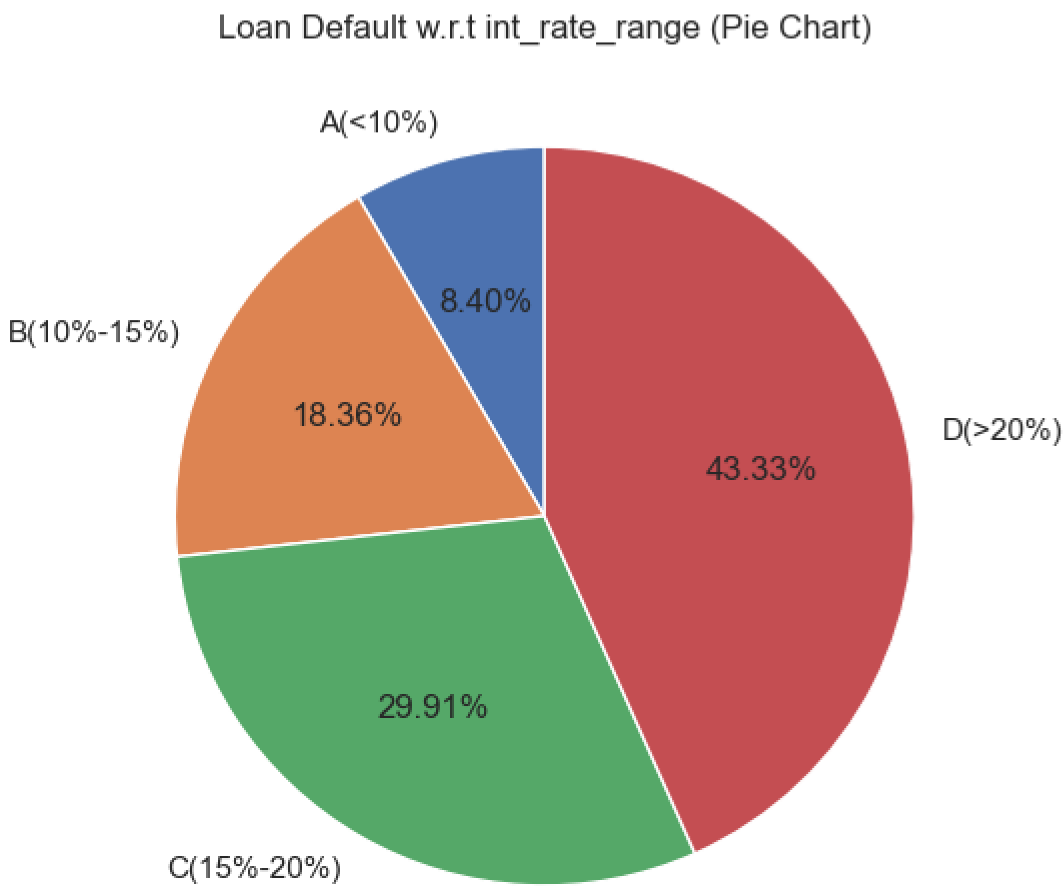
In [ ]:
```
loandf_int_rate_range = pd.pivot_table(loandf, values='loan_amnt', index='int_rate_rang
loandf_int_rate_range['Aggregate'] = loandf_int_rate_range['Charged Off'] + loandf_int_
loandf_int_rate_range['Charged Off%'] = round(loandf_int_rate_range['Charged Off']/loan
loandf_int_rate_range['Current%'] = round(loandf_int_rate_range['Current']/loandf_int_r
loandf_int_rate_range['Fully Paid %'] = round(loandf_int_rate_range['Fully Paid']/loand
loandf_int_rate_range
```

Out[ ]:

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
int_rate_range							
A(<10%)	799	75	11268	12142	6.58	0.62	92.80
B(10%-15%)	2738	531	15776	19045	14.38	2.79	82.84
C(15%-20%)	1794	432	5432	7658	23.43	5.64	70.93
D(>20%)	296	102	474	872	33.94	11.70	54.36

In [ ]:
```
# Your existing data and plot code
plt.figure(figsize=(8, 6))
plt.title('Loan Default w.r.t int_rate_range')
ax = sns.barplot(x='int_rate_range', y="Charged Off%", data=loandf_int_rate_range.reset_

# Add labels to each bar
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2., height + 0.5, '{:1.2f}'.format(height), ha=
```

```
# Extract data from the bar plot
labels = loandf_int_rate_range.reset_index()['int_rate_range'].tolist()
sizes = [p.get_height() for p in ax.patches]

# Convert the bar plot data into percentages
total = sum(sizes)
sizes = [(size / total) * 100 for size in sizes]

# Create a pie chart
plt.clf()  # Clear the existing figure
plt.pie(sizes, labels=labels, autopct='%1.2f%%', startangle=90)
plt.title('Loan Default w.r.t int_rate_range (Pie Chart)')

# Display the pie chart
plt.show()
```



Loan Amt Lets check the impact of loan amount on chance of loan default

In [ ]:
```
print(loandf['loan_amnt'].max())
print(loandf['loan_amnt'].min())
```

```
35000
500
```

APPLY BUCKETING ON LOAN AMOUNT

In [ ]:

```
def loan_amt_range(x):
    'Craete int_rate range'
    if x < 1000:
        return "A(<1K)"
    elif 1000 <= x < 5000:
        return "B(1K-5K)"
    elif 5000 <= x < 10000:
        return "C(5K-10K)"
    elif 10000 <= x < 15000:
        return "D(10K-15K)"
    elif 15000 <= x < 20000:
        return "E(15K-20K)"
    elif 20000 <= x < 30000:
        return "F(20K-30K)"
    else:
        return "G(>30K)"

loandf['loan_amt_range'] = loandf['loan_amnt'].apply(lambda x: loan_amt_range(x))
loandf['loan_amt_range'].value_counts()
```

Out[ ]:

```
C(5K-10K)     12178
D(10K-15K)     8924
B(1K-5K)       7505
F(20K-30K)     5033
E(15K-20K)     4860
G(>30K)        1205
A(<1K)           12
Name: loan_amt_range, dtype: int64
```

Create Pivot Table, handle NAN values and add new columns aggregate and percentage of iloan_amt_range for each loan_status

In [ ]:

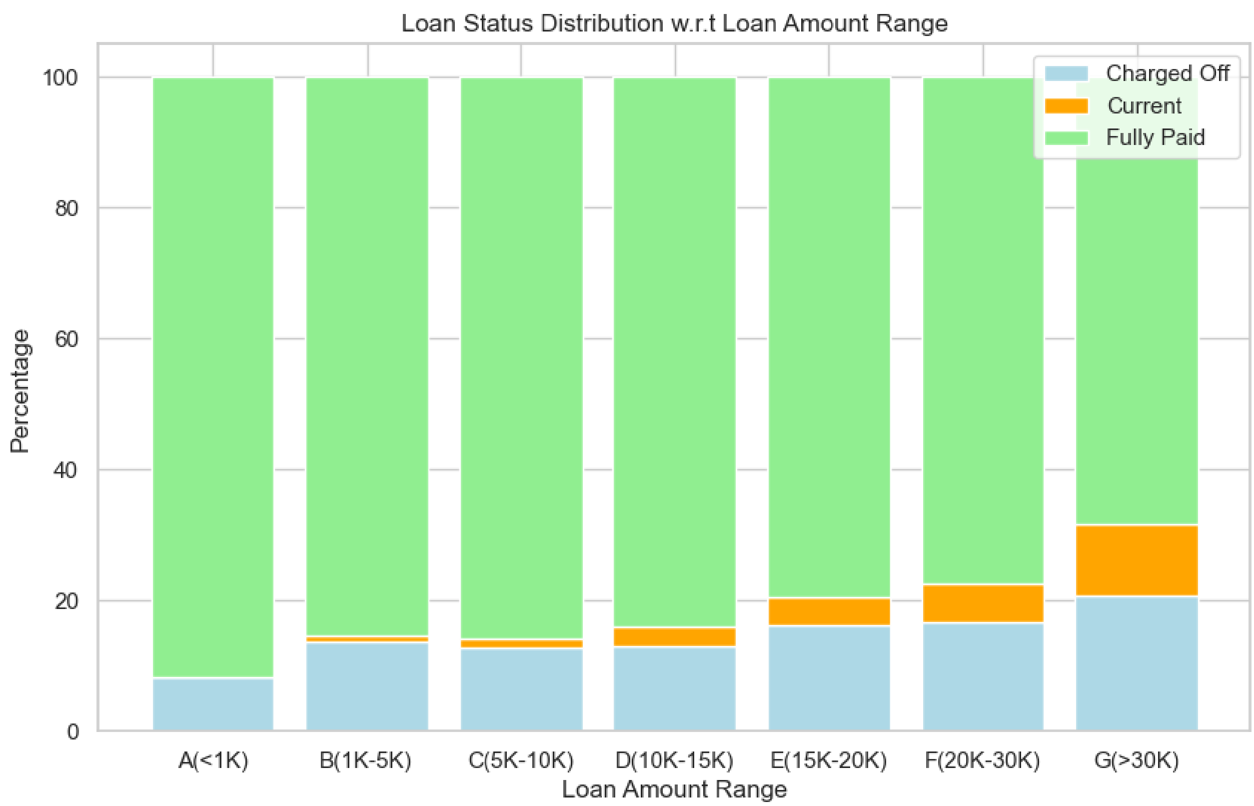
```
loandf_loan_amt_range = pd.pivot_table(loandf, values='loan_amnt', index='loan_amt_rang
loandf_loan_amt_range.loc[pd.isnull(loandf_loan_amt_range['Current']), ['Current']] = 0
loandf_loan_amt_range['Aggregate'] = loandf_loan_amt_range['Charged Off'] + loandf_loan_
loandf_loan_amt_range['Charged Off%'] = round(loandf_loan_amt_range['Charged Off']/loan
loandf_loan_amt_range['Current%'] = round(loandf_loan_amt_range['Current']/loandf_loan_
loandf_loan_amt_range['Fully Paid %'] = round(loandf_loan_amt_range['Fully Paid']/loand
loandf_loan_amt_range
```

Out[ ]:

loan_status loan_amt_range	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
A(<1K)	1.0	0.0	11.0	12.0	8.33	0.00	91.67
B(1K-5K)	1026.0	73.0	6406.0	7505.0	13.67	0.97	85.36
C(5K-10K)	1567.0	157.0	10454.0	12178.0	12.87	1.29	85.84
D(10K-15K)	1158.0	270.0	7496.0	8924.0	12.98	3.03	84.00
E(15K-20K)	785.0	209.0	3866.0	4860.0	16.15	4.30	79.55
F(20K-30K)	841.0	298.0	3894.0	5033.0	16.71	5.92	77.37
G(>30K)	249.0	133.0	823.0	1205.0	20.66	11.04	68.30

In [ ]:

```
plt.figure(figsize=(10, 6))

charged_off = loandf_loan_amt_range['Charged Off%']
current = loandf_loan_amt_range['Current%']
fully_paid = loandf_loan_amt_range['Fully Paid %']

plt.bar(loandf_loan_amt_range.index, charged_off, label='Charged Off', color='lightblue
plt.bar(loandf_loan_amt_range.index, current, bottom=charged_off, label='Current', colo
plt.bar(loandf_loan_amt_range.index, fully_paid, bottom=charged_off + current, label='F

plt.xlabel('Loan Amount Range')
plt.ylabel('Percentage')
plt.title('Loan Status Distribution w.r.t Loan Amount Range')
plt.legend()

plt.show()
```



Chance for loan default is highest for loan amount greater than 30,000.

Installments

Lets check the impact of loan installment with loan default

In [ ]:

```
print(loandf['installment'].max())
print(loandf['installment'].min())
```

```
1305.19
15.69
```

APPLY BUCKETING ON LOAN INSTALLMET BASED ON MIN AND MAX VALUE

In [ ]:

```
def loan_installment_range(x):
    'Craete int_rate range'
    if x < 50:
        return "A(<50)"
    elif 50 <= x < 100:
        return "B(50-100)"
    elif 100 <= x < 200:
        return "C(100-200)"
    elif 200 <= x < 500:
        return "D(200-500)"
    elif 500 <= x < 750:
        return "E(500-750)"
    elif 750 <= x < 1000:
        return "F(750-1000)"
    else:
        return "G(>1000)"

loandf['loan_installment_range'] = loandf['installment'].apply(lambda x: loan_installme
loandf['loan_installment_range'].value_counts()
```

Out[ ]:

```
D(200-500)     19296
C(100-200)      9249
E(500-750)      5065
B(50-100)       3190
F(750-1000)     1840
A(<50)           842
G(>1000)         235
Name: loan_installment_range, dtype: int64
```

Create Pivot Table, handle NAN values and add new columns aggregate and percentage of installment_range for each loan_status

In [ ]:

```
loandf_installment_range = pd.pivot_table(loandf, values='loan_amnt', index='loan_insta
loandf_installment_range.loc[pd.isnull(loandf_installment_range['Current']), ['Current'
loandf_installment_range['Aggregate'] = loandf_installment_range['Charged Off'] + loand
loandf_installment_range['Charged Off%'] = round(loandf_installment_range['Charged Off'
loandf_installment_range['Current%'] = round(loandf_installment_range['Current']/loandf
loandf_installment_range['Fully Paid %'] = round(loandf_installment_range['Fully Paid']
loandf_installment_range
```

Out[ ]:

loan_status	Charged Off	Current	Fully Paid	Aggregate	Charged Off%	Current%	Fully Paid %
loan_installment_range							
A(<50)	135.0	10.0	697.0	842.0	16.03	1.19	82.78
B(50-100)	464.0	47.0	2679.0	3190.0	14.55	1.47	83.98
C(100-200)	1214.0	150.0	7885.0	9249.0	13.13	1.62	85.25
D(200-500)	2673.0	608.0	16015.0	19296.0	13.85	3.15	83.00
E(500-750)	792.0	244.0	4029.0	5065.0	15.64	4.82	79.55
F(750-1000)	324.0	81.0	1435.0	1840.0	17.61	4.40	77.99
G(>1000)	25.0	0.0	210.0	235.0	10.64	0.00	89.36

Grade and Sub Grade

Grade Vs Loan Status

In [ ]:

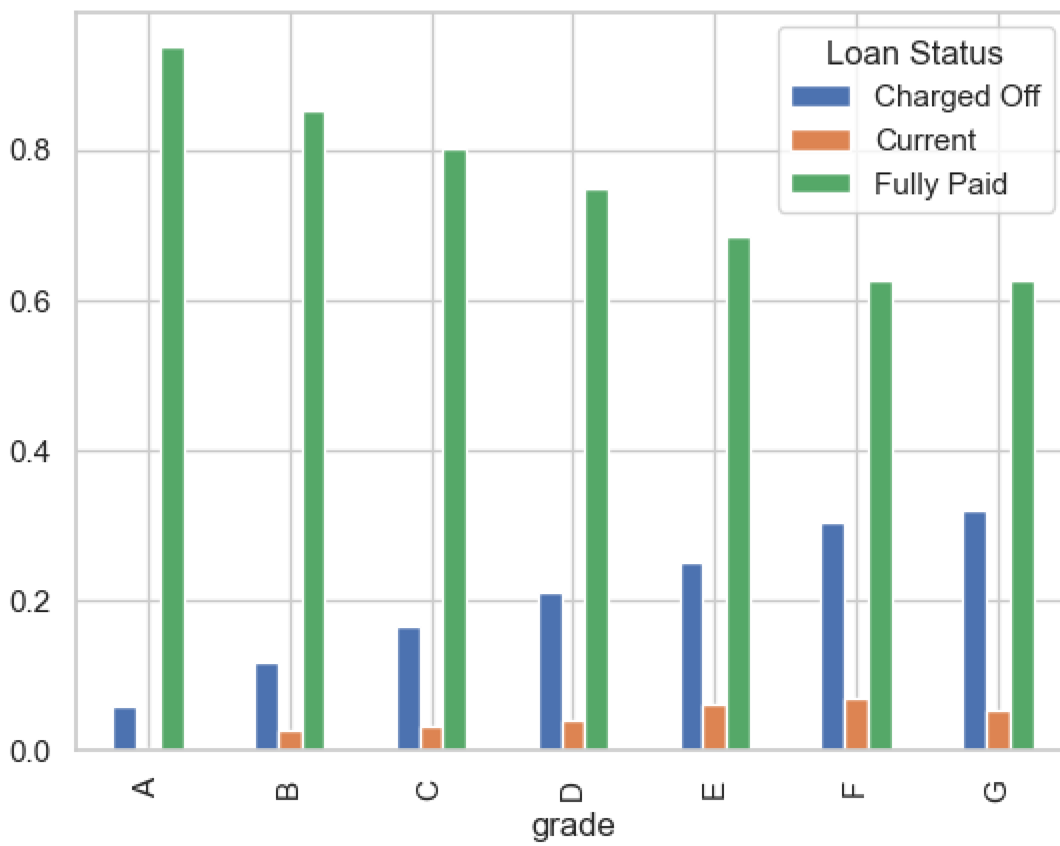
```
Grade_loanstatus = pd.crosstab(index=loandf["grade"],columns=loandf["loan_status"]).app
print(Grade_loanstatus)
```

```
loan_status  Charged Off   Current  Fully Paid
grade
A               0.059693  0.003966    0.936341
B               0.118552  0.028702    0.852745
C               0.166337  0.032601    0.801062
D               0.210665  0.041832    0.747503
E               0.251583  0.062984    0.685433
F               0.304099  0.069590    0.626311
G               0.319620  0.053797    0.626582
```

In [ ]:

```
Grade_loanstatus.plot.bar(stacked=False)
plt.legend(title='Loan Status')
```

Out[ ]: <matplotlib.legend.Legend at 0x14f1d15d0>



Loans to Grade 'G' - is prone to more default. Plot clearly shows from Grade A to G the Chargeoff i.e., Loan-default increases and 'Fully Paid' decreases

In [ ]:

```
Subgrade_loanstatus = pd.crosstab(index=loandf["sub_grade"],columns=loandf["loan_status
print(Subgrade_loanstatus)
```

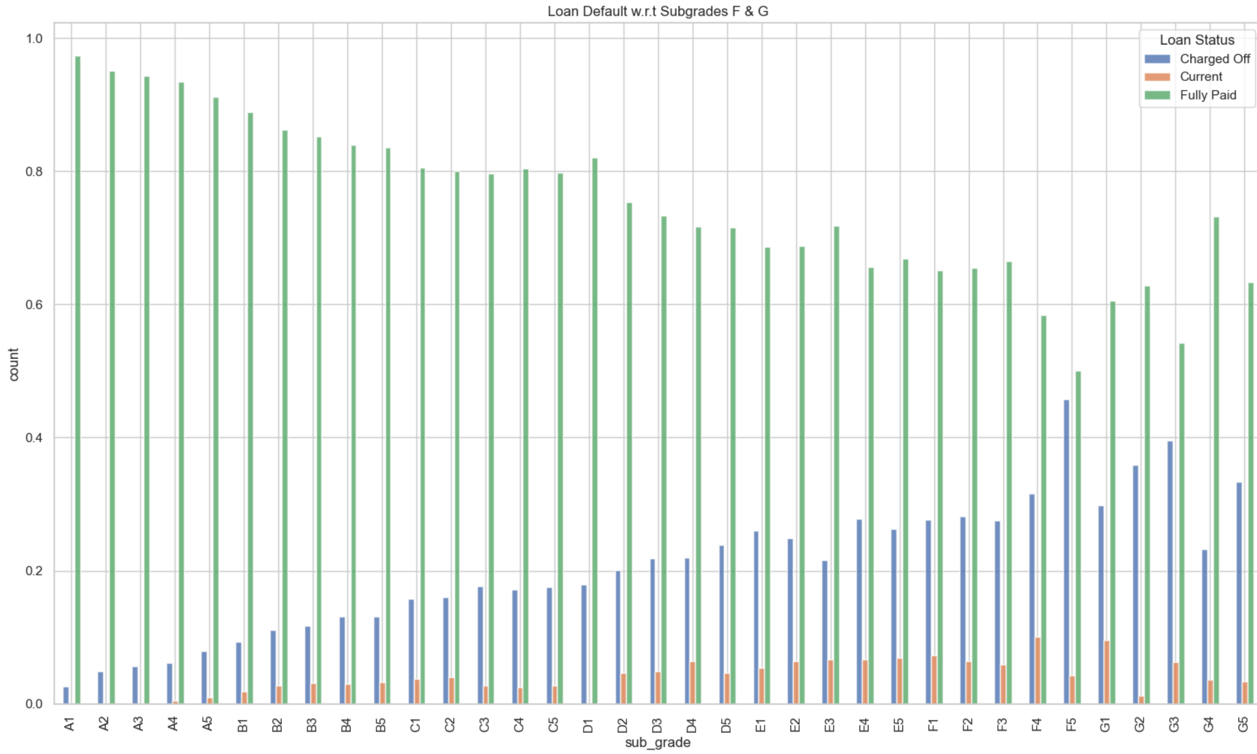
```
loan_status  Charged Off   Current  Fully Paid
sub_grade
A1              0.026339  0.000000    0.973661
```

A2	0.049072	0.000000	0.950928
A3	0.056906	0.000000	0.943094
A4	0.061677	0.004505	0.933818
A5	0.079139	0.009847	0.911014
B1	0.093443	0.018033	0.888525
B2	0.110841	0.027224	0.861935
B3	0.116901	0.031539	0.851560
B4	0.130971	0.029857	0.839172
B5	0.131657	0.032914	0.835429
C1	0.157303	0.037921	0.804775
C2	0.159622	0.039781	0.800597
C3	0.176586	0.026815	0.796599
C4	0.171521	0.024272	0.804207
C5	0.175379	0.026981	0.797639
D1	0.179377	0.000000	0.820623
D2	0.201039	0.045994	0.752967
D3	0.218244	0.048593	0.733163
D4	0.219164	0.064220	0.716616
D5	0.239130	0.045767	0.715103
E1	0.259502	0.053735	0.686763
E2	0.248476	0.064024	0.687500
E3	0.215190	0.066908	0.717902
E4	0.277533	0.066079	0.656388
E5	0.262019	0.069712	0.668269
F1	0.276596	0.072948	0.650456
F2	0.281124	0.064257	0.654618
F3	0.275676	0.059459	0.664865
F4	0.315476	0.101190	0.583333
F5	0.457627	0.042373	0.500000
G1	0.298077	0.096154	0.605769
G2	0.358974	0.012821	0.628205
G3	0.395833	0.062500	0.541667
G4	0.232143	0.035714	0.732143
G5	0.333333	0.033333	0.633333

In [ ]:

```
plt.figure(figsize=(10, 8))
ax = Subgrade_loanstatus.plot(alpha=0.8, kind='bar', stacked=False, figsize=(18.5, 10.5
plt.title('Loan Default w.r.t Subgrades F & G')
plt.legend(title='Loan Status')
ax.set_ylabel('count')
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```

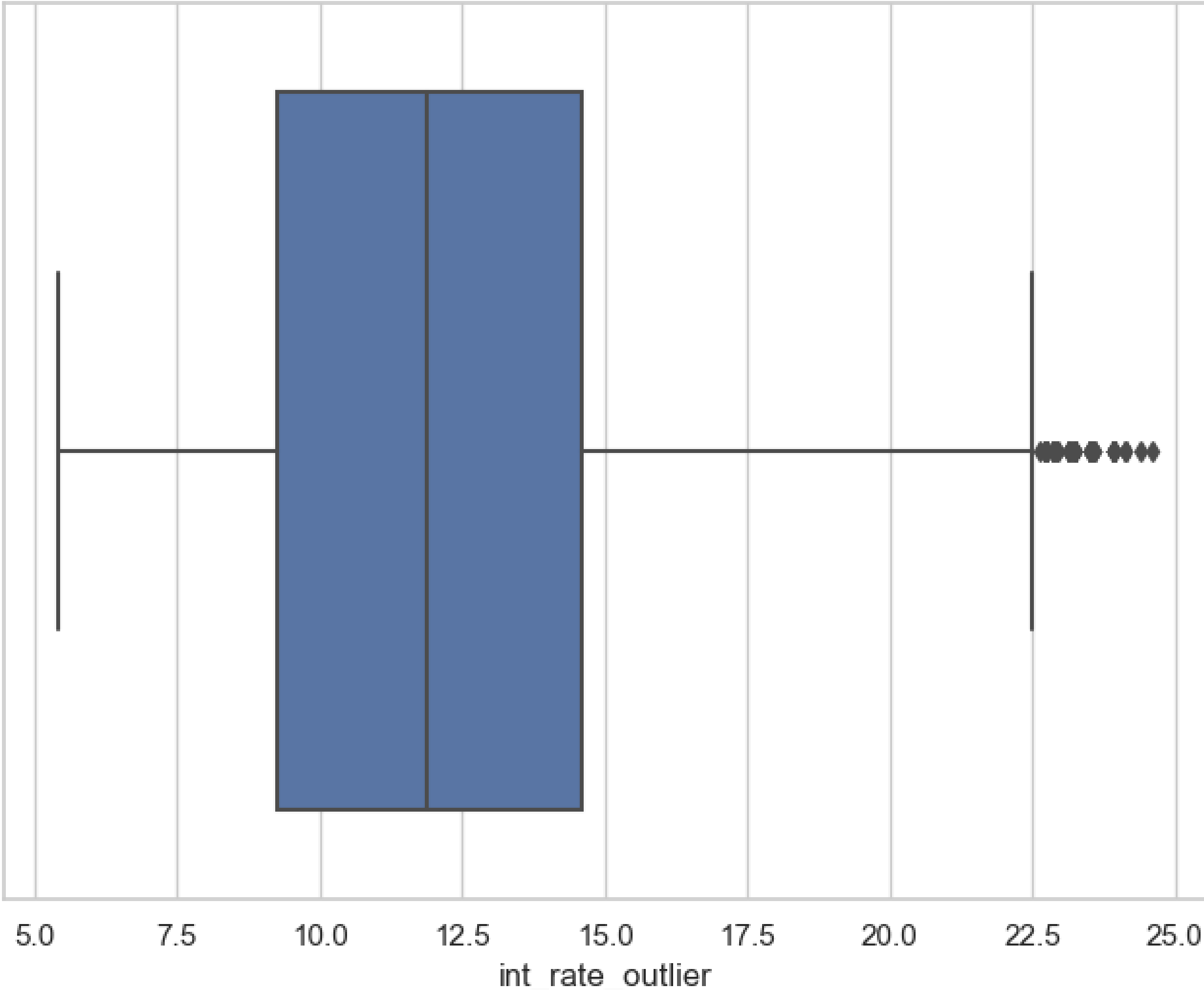


Outlier Analysis

In [ ]:

```
sns.set_style("whitegrid")
loandf["int_rate_outlier"] = loandf["int_rate%"]
loandf["int_rate_outlier"] = loandf["int_rate_outlier"].astype(float)
plt.figure(figsize=(8, 6))
sns.boxplot(x=loandf["int_rate_outlier"])
```

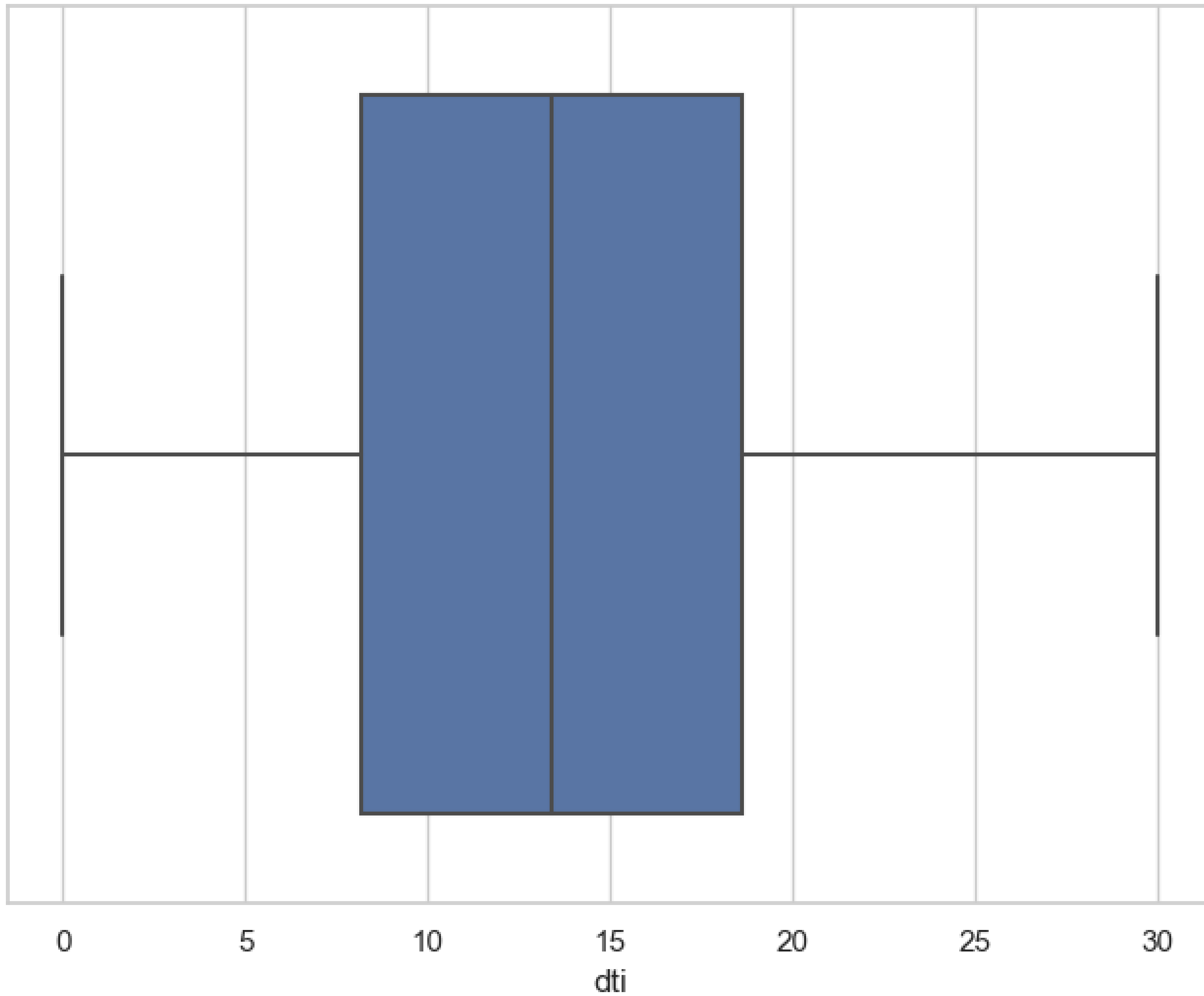
Out[ ]: <Axes: xlabel='int_rate_outlier'>

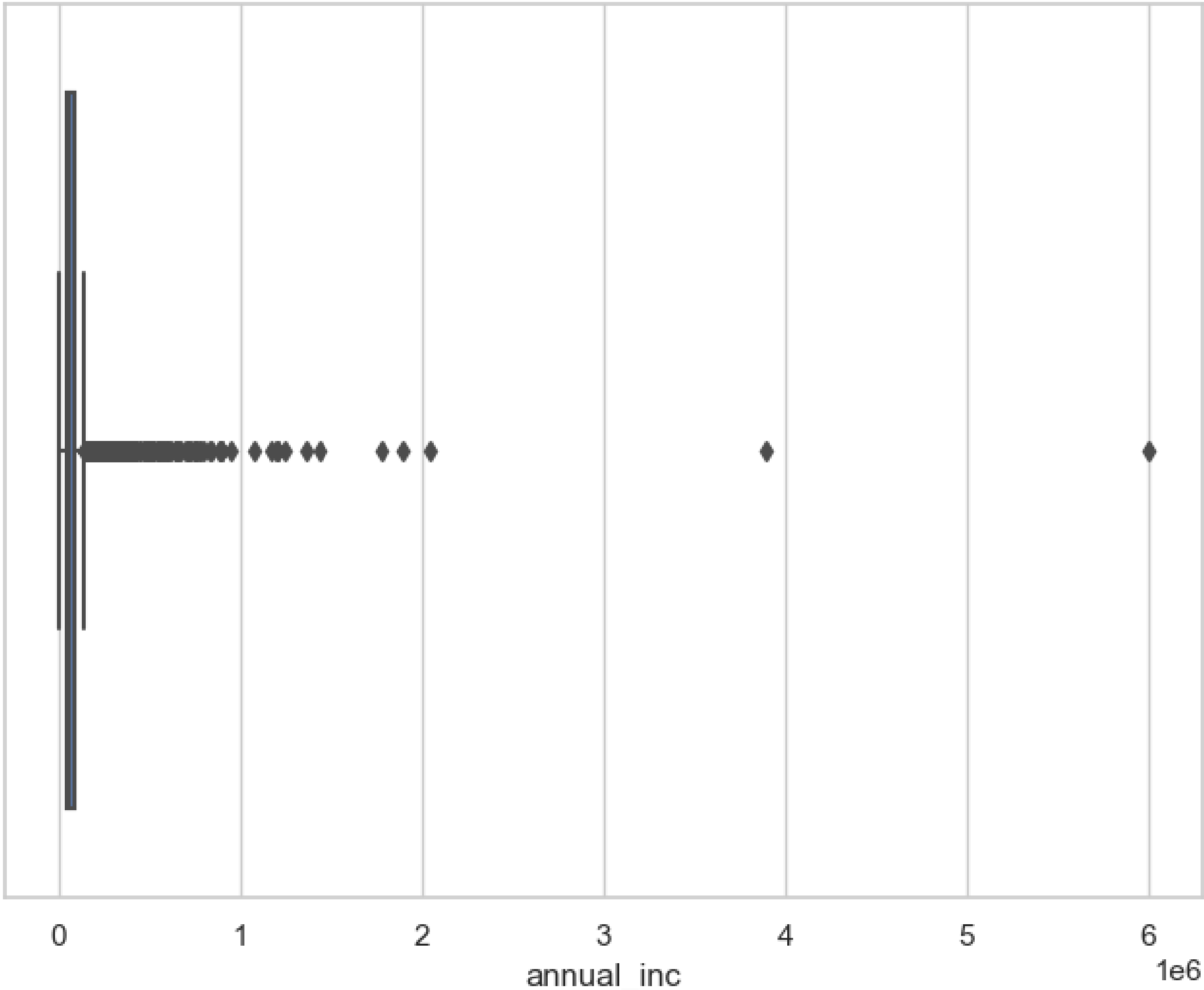


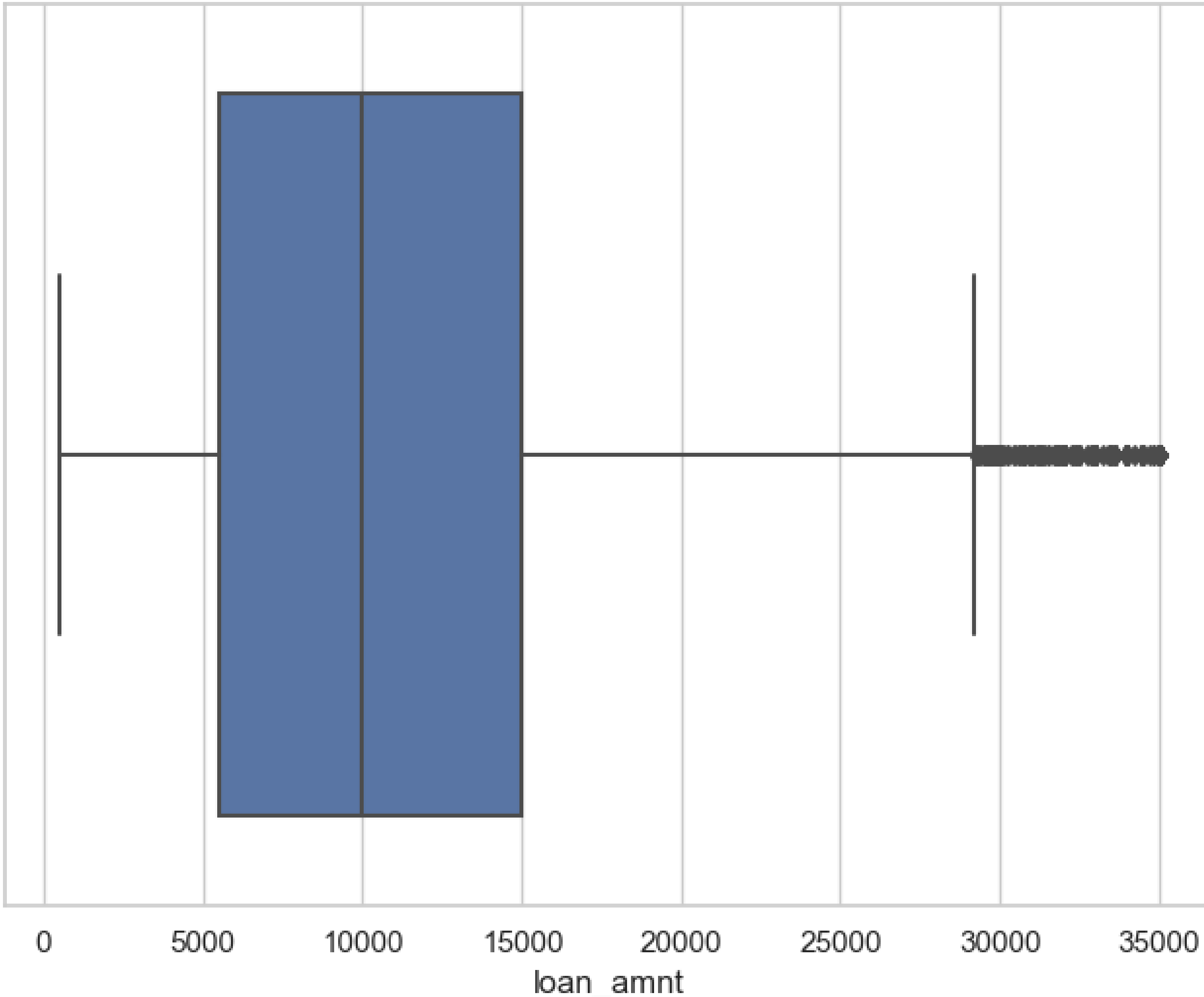
In [ ]:
```
def outliers(col_name):
    plt.figure(figsize=(8, 6))
    ax=sns.boxplot(x=loandf[col_name])
```

Check outliers for dti , annual_inc , loan_amnt

In [ ]:
```
outliers("dti")

outliers('annual_inc')

outliers("loan_amnt")
```


0
5
10
15
20
25
30
dti


0
1
2
3
4
5
6
1e6
annual_inc



Handling the Outliers

Create new dataframe loandf_New from the original loandf

In [ ]:

```
loandf_New = loandf
loandf_New.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 46 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     39717 non-null  int64
 1   member_id              39717 non-null  int64
 2   loan_amnt              39717 non-null  int64
 3   funded_amnt            39717 non-null  int64
 4   funded_amnt_inv        39717 non-null  float64
 5   installment            39717 non-null  float64
 6   grade                  39717 non-null  object
 7   sub_grade              39717 non-null  object
 8   emp_length             38642 non-null  object
 9   home_ownership         39717 non-null  object
 10  annual_inc             39717 non-null  float64
 11  verification_status    39717 non-null  object
 12  issue_d                39717 non-null  object
 13  loan_status            39717 non-null  object
 14  purpose                39717 non-null  object
 15  addr_state             39717 non-null  object
```

```
 16  dti                      39717 non-null  float64
 17  delinq_2yrs              39717 non-null  int64
 18  inq_last_6mths           39717 non-null  int64
 19  open_acc                 39717 non-null  int64
 20  pub_rec                  39717 non-null  int64
 21  revol_bal                39717 non-null  int64
 22  revol_util               39667 non-null  object
 23  total_acc                39717 non-null  object
 24  out_prncp                39717 non-null  float64
 25  out_prncp_inv            39717 non-null  float64
 26  total_pymnt              39717 non-null  float64
 27  total_pymnt_inv          39717 non-null  float64
 28  total_rec_prncp          39717 non-null  float64
 29  total_rec_int            39717 non-null  float64
 30  total_rec_late_fee       39717 non-null  float64
 31  recoveries               39717 non-null  float64
 32  collection_recovery_fee  39717 non-null  float64
 33  last_pymnt_d             39646 non-null  object
 34  last_pymnt_amnt          39717 non-null  float64
 35  last_credit_pull_d       39715 non-null  object
 36  pub_rec_bankruptcies     39020 non-null  float64
 37  loanPeriod               39717 non-null  int64
 38  zip_code_num             39717 non-null  int64
 39  dti_level                39717 non-null  object
 40  salary_range             39717 non-null  object
 41  int_rate%                39717 non-null  float64
 42  int_rate_range           39717 non-null  object
 43  loan_amt_range           39717 non-null  object
 44  loan_installment_range   39717 non-null  object
 45  int_rate_outlier         39717 non-null  float64
dtypes: float64(17), int64(11), object(18)
memory usage: 13.9+ MB
```

Remove these columns as we need not check for outliers in these columns

In [ ]:

```
#filt_df = df.loc[:, df.columns!=[('User_id','Col1')] ]

out_filt_df = loandf_New.drop([
                    'id',
                    'member_id',

                    'grade',
                    'sub_grade',
                    'emp_length',
                    'home_ownership',
                    'verification_status',
                    'issue_d',
                    'loan_status',
                    'purpose',
                    'addr_state',
                    'delinq_2yrs',
                    'inq_last_6mths',
                    'open_acc',
                    'pub_rec',
                    'revol_util',
                    'out_prncp','out_prncp_inv','total_rec_late_fee','recoveries','coll
                    'last_pymnt_d',
                    'last_credit_pull_d',

                    'pub_rec_bankruptcies',
                    'loanPeriod',
                    'zip_code_num',
```

```
                  'dti_level','salary_range','int_rate%','int_rate_range',
                  'loan_amt_range','loan_installment_range','int_rate_outlier',
                  'total_acc'
                ],axis=1)
```

In [ ]:

```
out_filt_df.head()
```

Out[ ]:

	loan_amnt	funded_amnt	funded_amnt_inv	installment	annual_inc	dti	revol_bal	total_pymnt	tc
0	5000	5000	4975.0	162.87	24000.0	27.65	13648	5863.155187	
1	2500	2500	2500.0	59.83	30000.0	1.00	1687	1008.710000	
2	2400	2400	2400.0	84.33	12252.0	8.72	2956	3005.666844	
3	10000	10000	10000.0	339.31	49200.0	20.00	5598	12231.890000	
4	3000	3000	3000.0	67.79	80000.0	17.94	27783	3513.330000	

Apply precentile 0.05 - 0.95 range to find outliers

In [ ]:

```
low = .05
high = .95
quant_df = out_filt_df.quantile([low, high])
print(quant_df)
```

```
      loan_amnt  funded_amnt  funded_amnt_inv  installment  annual_inc    dti  \
0.05     2400.0       2400.0      1873.658000       71.246     24000.0   2.13
0.95    25000.0      25000.0     24736.572264      762.996    142000.0  23.84

      revol_bal   total_pymnt  total_pymnt_inv  total_rec_prncp  \
0.05      321.8   1887.957036         1420.408         1339.842
0.95    41656.4  30245.118530        29627.236        24999.982

      total_rec_int  last_pymnt_amnt
0.05        186.168           43.340
0.95       7575.812        12183.944
```

Apply low and high percentiles into the dataframe and apply Null to outliers values

In [ ]:

```
out_filt_df = out_filt_df.apply(lambda x: x[(x>quant_df.loc[low,x.name]) &
                                    (x < quant_df.loc[high,x.name])], axis=0)
```

In [ ]:

```
out_filt_df.isnull().sum()*100/out_filt_df.shape[0]
```

Out[ ]:

```
loan_amnt          12.957647
funded_amnt        11.872388
funded_amnt_inv     9.993957
installment         9.993957
annual_inc         10.552954
dti                10.014101
revol_bal           9.993957
total_pymnt         9.993957
total_pymnt_inv     9.993957
total_rec_prncp     9.993957
total_rec_int       9.993957
```

```
last_pymnt_amnt     9.993957
dtype: float64
```

We will decide whether or not to remove the outliers in our further analysis or keep it seperately and analyse them to see if we find some meaningful insights from them

In [ ]:

```
out_filt_df.fillna(out_filt_df.mean(), inplace=True)

out_filt_df.isnull().sum()*100/out_filt_df.shape[0]  # check for missing values after i
```

Out[ ]:

```
loan_amnt          0.0
funded_amnt        0.0
funded_amnt_inv    0.0
installment        0.0
annual_inc         0.0
dti                0.0
revol_bal          0.0
total_pymnt        0.0
total_pymnt_inv    0.0
total_rec_prncp    0.0
total_rec_int      0.0
last_pymnt_amnt    0.0
dtype: float64
```

In [ ]:

```
loandf.update(out_filt_df)
loandf = loandf.dropna()
loandf.isnull().sum()*100/loandf.shape[0]
```

Out[ ]:

```
id                         0.0
member_id                  0.0
loan_amnt                  0.0
funded_amnt                0.0
funded_amnt_inv            0.0
installment                0.0
grade                      0.0
sub_grade                  0.0
emp_length                 0.0
home_ownership             0.0
annual_inc                 0.0
verification_status        0.0
issue_d                    0.0
loan_status                0.0
purpose                    0.0
addr_state                 0.0
dti                        0.0
delinq_2yrs                0.0
inq_last_6mths             0.0
open_acc                   0.0
pub_rec                    0.0
revol_bal                  0.0
revol_util                 0.0
total_acc                  0.0
out_prncp                  0.0
out_prncp_inv              0.0
total_pymnt                0.0
total_pymnt_inv            0.0
total_rec_prncp            0.0
total_rec_int              0.0
total_rec_late_fee         0.0
recoveries                 0.0
collection_recovery_fee    0.0
last_pymnt_d               0.0
```

```
last_pymnt_amnt          0.0
last_credit_pull_d       0.0
pub_rec_bankruptcies     0.0
loanPeriod               0.0
zip_code_num             0.0
dti_level                0.0
salary_range             0.0
int_rate%                0.0
int_rate_range           0.0
loan_amt_range           0.0
loan_installment_range   0.0
int_rate_outlier         0.0
dtype: float64
```

In [ ]:

```
for column in loandf.columns:
    unique_values = loandf[column].unique()
    print(f"Unique values in column '{column}':")
    print(unique_values)
```

```
Unique values in column 'id':
[1077501 1077430 1077175 ...  132892  119043  112496]
Unique values in column 'member_id':
[1296599 1314167 1313524 ...  132889  119040  112493]
Unique values in column 'loan_amnt':
[ 5000.          2500.         10299.70275978 10000.
  3000.          7000.          5600.           5375.
  6500.         12000.          9000.           3600.
  6000.          9200.         20250.          21000.
 15000.          4000.          8500.           4375.
 12400.         10800.         12500.           9600.
  4400.         14000.         11000.          16000.
  7100.         13000.         17500.          17675.
  8000.          3500.         16425.           8200.
 20975.          6400.         14400.           7250.
 18000.         11800.          4500.          10500.
 15300.         20000.          6200.           7200.
  9500.         18825.         24000.           5500.
 19750.         13650.         10625.           8850.
  6375.         11100.          4200.           8875.
 13500.         21600.          8450.          13475.
 22000.          7325.          7750.          13350.
 22475.          8400.         13250.           7350.
 11500.         11625.         15075.           5300.
  8650.          7400.         24250.          19600.
  4225.         16500.         15600.          14125.
 13200.         12300.          3200.          11875.
 23200.          4800.          7300.          10400.
  6600.          4475.          6300.           8250.
  9875.         21500.          7800.           9750.
 15550.         17000.          7500.           5800.
  8050.          5400.          4125.           9800.
 15700.          9900.          6250.          10200.
 23000.         21250.          8125.          18800.
 19200.         12875.          2625.          11300.
  4100.         18225.         18500.          16800.
 14050.         16100.         10525.          19775.
 14500.         11700.          4150.          12375.
 22250.         11200.         22500.          15900.
  3150.         18550.          7700.          24500.
 22200.         21400.          9400.          22400.
  5825.          7650.         20675.          20500.
 12800.          7600.          9575.          14575.
  7125.         10700.         10375.           3050.
 14100.         20050.         24925.          13600.
```

```
 7150.  15500.  17475.  17050.
 3250.  22750.   5900.  12600.
 6750.  17250.  19075.  17200.
13225.  11775.  16400.  10075.
 9350.   8075.  15625.  20125.
 8300.   2425.   6950.   5350.
 5875.   9450.  19000.  20400.
21650.  20300.  24575.   5850.
 4750.   5275.   9175.  10050.
19400.  18200.   8800.  19500.
 5200.  11900.   3300.  12200.
22575.   7175.  18250.  16750.
12950.   6350.  14750.   6625.
 6900.  18650.  22800.  12250.
 4350.  21200.   2700.   6025.
 3825.  14150.   2800.  18975.
 8575.   2575.   5450.   3800.
14650.  11250.   6075.   8475.
 9250.   3625.   4250.  12650.
13150.   4300.  10275.  23600.
 7875.  14550.   9925.  15850.
 6325.  15200.  15250.   6800.
11325.  13975.  13800.   3100.
 3975.   3575.  23700.   6475.
17375.  15800.  17625.  16675.
 5250.  22950.   4650.  10250.
 6100.   8325.   4850.   9425.
12700.  14850.  14300.   5150.
21625.   3775.  21575.  16250.
 8375.  18725.  11125.   3525.
19800.   9300.  19125.   5575.
12900.  10150.  20450.  23500.
16600.   6925.  14675.  11550.
17400.   3400.  12775.   5050.
12100.   6975.  23325.  11600.
 5100.  10175.  18400.  16550.
 5650.  16450.  18950.   3650.
10125.  16775.  20200.  10600.
 3725.  19425.   3125.  23800.
 4025.   2600.   8900.  10900.
17600.  14825.   7925.  14950.
 6700.   8600.   4900.  15575.
 5700.   3175.  14800.   5750.
14600.   6550.  22900.   6850.
 4600.  11425.   5325.  16950.
10675.   6650.  10775.  17325.
 3700.  20800.  13575.   4725.
24800.  15750.  17100.  15875.
10925.   4950.  10575.   2850.
21100.  11050.  20375.   9325.
 9375.   7475.  22125.  17750.
 8675.   7450.  24625.  17900.
12075.   6725.  24400.   5225.
14075.  17175.   9475.   9975.
20900.  12150.  17725.  15350.
 4925.   4550.  18750.  15125.
12475.   2750.   4625.  12175.
 7575.  23525.  12350.   9525.
 8975.  11975.  12850.  19850.
21850.   4425.   2550.  11400.
21725.  23100.  13700.   9950.
21750.  13750.  12025.  23400.
19700.   3900.  14725.  17800.
 5175.  15025.  23850.   9100.
23675.   9825.  16200.  11650.
```

```
18875.       3950.      19950.      12750.
 2875.      16275.      10300.      17450.
 3450.      13100.      23275.       8700.
 6450.       3675.       8150.      23975.
 3350.       7075.       8625.      11025.
 7850.      14175.       9150.      19925.
14275.      17825.      16875.      21800.
14475.      14225.      10225.      10650.
12725.      10950.      16300.      12550.
11725.      22600.       6225.       4450.
 3875.      13275.       6775.      19450.
 2900.       2450.      21300.       4700.
 7425.      19575.      24600.      15950.
13300.       2975.       8100.       6425.
 4050.      23450.      13675.      21350.
 9050.       2675.       5025.       5950.
12625.      10825.      24700.      13125.
 6125.       6825.      10975.      20950.
 3850.      11750.      15825.       7525.
 7950.      13400.       3375.      17850.
17875.       7550.       6175.      21125.
 3750.      10025.      14350.       7775.
18900.       8025.      13775.       3075.
11525.       5550.       5975.      22100.
14700.       3325.       5075.      14975.
 5625.      11575.      16325.      24200.
15050.       5425.      17700.      12450.
19725.      19550.      22875.      23075.
15450.      10750.       4325.       3275.
 8175.      20700.       4775.       8225.
 4575.       5125.      15775.      19475.
14200.      21225.      17225.      12425.
 7900.      14525.       2650.       8275.
 6275.       4075.      13075.      23750.
24650.      14250.       8825.       8350.
19150.       9725.      18575.       8725.
16050.      16075.       6150.       8750.
11075.      17950.      10875.      16350.
 3925.      11375.      18325.       9650.
 2725.      10425.       6575.      13175.
 9550.      12675.      15425.      18300.
18600.       5525.      10550.      22325.
15175.       3025.      12225.      15650.
11450.      23350.      13625.      20600.
 8550.      15975.       9775.      13425.
 2950.      12925.       9075.      21700.
15400.       4975.      11275.       7725.
 9225.      13725.       8775.      19250.
14900.      17300.       9700.      12325.
10100.      10350.       2825.      17975.
15275.       2925.       2525.       5725.
23425.       4875.       2475.       3425.
16700.       2775.      13050.       7975.
 5925.      16225.       9275.      11350.
21450.      10850.       7225.      13325.
 1200.       5475.      19300.       7050.
24375.       5775.      24175.      12050.
13850.      17075.      18275.       9125.
16525.      11850.      22300.       7675.
 8525.       7275.       4525.       7025.
14625.      13375.       4675.      24975.
12825.      18150.      18050.       9850.
14875.      17425.      16725.      13550.
 9625.      15150.      19875.      22650.
17150.       6875.       7375.       5675.
```

```
  4275.          7625.          6525.          3225.
  6675.         15675.         17275.         11475.
  1000.         12975.         15325.          8950.
 11675.         12275.          3475.         21425.
  3550.         18125.         23050.         11175.
 10450.         21825.         10475.         20150.
 24750.         13900.          4175.         24100.
 17925.         24150.         19975.         19900.
 13950.         12125.         11225.         23475.
 19650.         13450.         10725.         20475.
 17525.         23575.        ]
Unique values in column 'funded_amnt':
[ 5000.          2500.         10238.59253122 10000.
  3000.          7000.          5600.          5375.
  6500.         12000.          9000.          3600.
  6000.          9200.         20250.         21000.
 15000.          4000.          8500.          4375.
 12400.         10800.         12500.          9600.
  4400.         14000.         11000.         16000.
  7100.         13000.          8950.         17675.
  8000.          3500.          8925.         16425.
  8200.         13575.          6400.         14400.
  7200.         18000.         22075.         11800.
  4500.         10500.         15300.         12800.
 17500.          6200.          9500.         18825.
 24000.          5500.         19750.         13650.
 10625.          8850.          6375.         11100.
  4200.          8875.         13500.         21600.
  8450.         20000.         13475.         22000.
  7325.          7750.         13350.         22475.
  8400.         13250.          7350.         11500.
 11625.         15075.          5300.          8650.
  7400.         18100.         19600.          4225.
 16500.         15600.         14125.          8975.
 12300.          3200.         11875.         23200.
  4800.          7300.         10400.          6600.
 23250.          4475.          6300.         23150.
  8250.          9875.         16975.          7800.
  9750.         15050.         15550.         17000.
  7500.          5800.          8050.          5400.
  4125.          9800.         15700.          9900.
  6250.         19825.         10200.         23000.
 16925.         16475.         21250.         20675.
  8125.         18800.         19200.          8475.
  2625.         11300.          4100.         13450.
 16800.         17950.         13700.         19950.
 14050.         16100.         10525.         15775.
 19775.         14500.         11700.          4150.
 12375.         22250.         11200.         22500.
 15900.          3150.         11600.         18625.
  7700.         24500.         22200.         21400.
  9400.         17275.         19275.         22400.
  5825.          7650.         13200.         20500.
  7600.          9575.          8900.         14575.
  7125.         10700.         16050.         10375.
 21800.          3050.         21275.         14100.
 22050.         15325.         20050.         24925.
 21825.         13600.         15925.         21350.
  8175.         12325.         16725.         22875.
  8225.          7150.         15500.          5050.
 18550.         17475.         17350.         17050.
  3250.         22750.          9350.          5900.
 12600.          6750.         17250.         19075.
 17200.         12625.         13225.         11775.
 16400.         10075.         18275.         18225.
```

```
        8075.        15625.        10175.        17900.
       20125.        18375.         8300.         2425.
        6950.        18500.         5350.        22600.
        5875.         9450.        19000.        20400.
       21650.        15825.        20300.        13950.
       24575.         5850.        16175.         4750.
        5275.         9175.        10050.        19400.
       23350.        18200.         8800.        19500.
        5200.        11900.        10475.        21675.
       16300.         3300.        12250.        10350.
       13900.        13750.         7175.        18250.
       16750.        12950.         6350.        14750.
        6625.        12875.         6900.        18650.
       22800.         4350.        21200.         2700.
        6025.         3825.        14150.         2800.
       18975.         8575.         2575.         5450.
        3800.        14650.        11250.         6075.
        9250.         3625.         4250.        12650.
       13150.         4300.        10275.        24250.
       23600.         7875.        14550.         9925.
       15850.         6325.        15200.        15250.
        6800.        11325.        13975.        13800.
        3100.         3975.         3575.        23700.
        6475.        17375.        15800.        17625.
       16675.         5250.        22950.         4650.
       10250.         6100.         8325.         4850.
        9425.        12700.        14850.        14300.
        5150.        21625.         3775.        21575.
       16250.         8375.        18725.        11125.
        3525.        19800.         9300.        21500.
       19125.         5575.        12900.        10150.
       20450.        23500.        16600.         6925.
       14675.        11550.        17400.         3400.
       12775.        12100.         6975.        23325.
        5100.        18400.        16550.         5650.
       16450.        18950.         3650.        10125.
       16775.        20200.        10600.         3725.
       19425.         3125.        23800.         4025.
        2600.        10900.        17600.        14825.
        7925.        14950.         6700.         8600.
        4900.        15575.         5700.         3175.
       14800.         5750.        14600.         6550.
       22900.         6850.         4600.        11425.
        5325.        16950.        10675.         6650.
       10775.        17325.         3700.        20800.
        4725.        24800.        15750.        17100.
       15875.        10925.         4950.        10575.
        2850.        21100.        11050.        22575.
       20375.         9325.         9375.         7475.
       22125.        17750.         8675.         7450.
       24625.        12075.         6725.        24400.
        5225.        14075.        17175.         9475.
        9975.        20900.        12150.        17725.
       15350.         4925.         4550.        18750.
       15125.        12475.         2750.         4625.
       12175.         7575.        23525.        12350.
        9525.        11975.        12850.        19850.
       21850.         4425.         2550.        11400.
       21725.        23100.         9950.        21750.
       12025.        23400.        19700.         3900.
       14725.        17800.         5175.        15025.
       23850.         9100.        23675.         9825.
       16200.        11650.        18875.         3950.
       12750.         2875.        16275.        10300.
       17450.         3450.        13100.        23275.
```

```
 8700.   6450.   3675.   8150.
23975.   3350.   7075.   8625.
11025.   7850.  14175.  12200.
 9150.  19925.  14275.  17825.
16875.  14475.  14225.  10225.
10650.  12725.   7250.  10950.
12550.  11725.   6225.   4450.
 3875.  13275.   6775.  19450.
 2900.   2450.  21300.   4700.
 7425.  19575.  24600.  15950.
13300.   2975.   8100.   6425.
 4050.  23450.  13675.   9050.
14350.   2675.  23775.   5025.
22275.  22700.  21075.   5950.
13725.  24425.  24975.  12925.
20275.  10825.  24700.  13125.
 6125.   6825.  18075.  10975.
18600.   3850.  22550.  11750.
23225.  20850.  15100.  20950.
16075.  14775.   7525.   9650.
17575.  21025.   7950.  14425.
16025.  22975.  21775.  13400.
20975.   8750.  22375.   3375.
12575.  19150.  15150.  17975.
23375.  19325.  12050.  20225.
24375.  24350.  16375.   9125.
17850.  10850.  17875.  20350.
22425.  20575.  22525.   7550.
11375.  15400.  11175.  24300.
22925.   6175.  19975.  21125.
18450.   3750.  18925.  10025.
13925.  14525.  20625.  22100.
 7775.  20600.  14900.  18900.
18350.   8025.  13775.   3075.
 8525.  24475.  24825.  11525.
 5550.   5975.  14700.   3325.
 5075.  14975.   5625.  11575.
16325.  24200.   5425.  17700.
12450.  19725.  19550.  23075.
15450.  10750.   4325.   3275.
20700.   4775.   4575.   5125.
19475.  14200.  21225.  17225.
12425.   7900.   2650.   8275.
 6275.   4075.  13075.  20100.
23050.  23750.  12125.  16850.
14250.  19050.   8825.  10100.
10325.  16150.   4525.  23475.
22825.   8350.   7675.  23950.
19375.  17775.  18575.  14875.
 9725.  22850.  16225.  22325.
11450.  11225.   8725.  15175.
15425.  13550.  24725.  16900.
18475.  15725.  22625.   6150.
11075.  10875.  16350.  18050.
 3925.  18325.  18525.  19875.
 7225.   2725.  10425.   6575.
13175.   9550.  16700.  23650.
21175.  11150.  12675.   9025.
18300.   5525.  10550.   3025.
12225.  15650.  17300.  22025.
23625.  10450.  19625.  21925.
 9625.  24875.  13625.  21425.
 9775.  17150.  19900.   8550.
15975.  12525.  17425.  19100.
20550.  24125.  22725.  24450.
```

```
 15275.        18425.        11475.        13425.
  2950.         9075.        21700.         4975.
 11275.         7725.         9225.         8775.
 19250.        19025.        21525.        24850.
 15675.        20525.         9700.        20650.
 18850.        12825.        11675.        11825.
 15225.        15525.         2825.        20150.
 21875.        18175.        19675.        17125.
 16625.         2925.         2525.        21375.
 23300.        22675.        23725.        20325.
  5725.        23425.         4875.         2475.
  3425.         2775.        13050.         7975.
  5925.         9275.        11350.        21450.
 13325.         1200.         5475.        19300.
  7050.         5775.        24175.        13850.
 17075.        16525.        11850.        22300.
 20875.         7275.         7025.        14625.
 13375.         4675.        18150.         9850.
 14325.        22650.         6875.         7375.
  5675.         4275.         7625.         6525.
  3225.         6675.         8425.        14450.
 20725.         6050.        11925.        21150.
 17525.         1000.        17025.         3475.
 11950.         7825.        12975.        20775.
 18775.        14925.        12275.        10725.
 16650.        15475.        16125.        13825.
 23550.        18675.        17650.        17925.
 13525.        24275.        14025.         3550.
 22150.        21475.        18125.         9675.
 13025.        15375.        16575.        19650.
 24750.         4175.        24100.        24150.
 20475.         4825.       ]
Unique values in column 'funded_amnt_inv':
[4975.       2500.       2400.       ... 3738.488872 3110.87
 6425.004533]
Unique values in column 'installment':
[162.87       308.70975129  84.33       ... 155.52       507.46
  99.44      ]
Unique values in column 'grade':
['B' 'C' 'A' 'E' 'F' 'D' 'G']
Unique values in column 'sub_grade':
['B2' 'C4' 'C5' 'C1' 'B5' 'A4' 'E1' 'F2' 'C3' 'B1' 'D1' 'A1' 'B3' 'B4'
 'D2' 'A3' 'A5' 'D5' 'A2' 'E4' 'D3' 'C2' 'D4' 'F3' 'E3' 'F4' 'F1' 'E5'
 'G4' 'E2' 'G3' 'G2' 'G1' 'F5' 'G5']
Unique values in column 'emp_length':
['10+ years' '< 1 year' '1 year' '3 years' '8 years' '9 years' '4 years'
 '5 years' '6 years' '2 years' '7 years']
Unique values in column 'home_ownership':
['RENT' 'OWN' 'MORTGAGE' 'OTHER']
Unique values in column 'annual_inc':
[ 63658.2781747  30000.         49200.        ...  88068.
 100671.39       36153.       ]
Unique values in column 'verification_status':
['Verified' 'Source Verified' 'Not Verified']
Unique values in column 'issue_d':
['Dec-11' 'Nov-11' 'Oct-11' 'Sep-11' 'Aug-11' 'Jul-11' 'Jun-11' 'May-11'
 'Apr-11' 'Mar-11' 'Feb-11' 'Jan-11' 'Dec-10' 'Nov-10' 'Oct-10' 'Sep-10'
 'Aug-10' 'Jul-10' 'Jun-10' 'May-10' 'Apr-10' 'Mar-10' 'Feb-10' 'Jan-10'
 'Dec-09' 'Nov-09' 'Oct-09' 'Sep-09' 'Aug-09' 'Jul-09' 'Jun-09' 'May-09'
 'Apr-09' 'Mar-09' 'Feb-09' 'Jan-09' 'Dec-08' 'Nov-08' 'Oct-08' 'Sep-08'
 'Aug-08' 'Jul-08' 'Jun-08' 'May-08' 'Apr-08' 'Mar-08' 'Feb-08' 'Jan-08'
 'Dec-07' 'Nov-07' 'Oct-07' 'Aug-07']
Unique values in column 'loan_status':
['Fully Paid' 'Charged Off' 'Current']
Unique values in column 'purpose':
```

```
['credit_card' 'car' 'small_business' 'other' 'wedding'
 'debt_consolidation' 'home_improvement' 'major_purchase' 'medical'
 'moving' 'vacation' 'house' 'renewable_energy' 'educational']
Unique values in column 'addr_state':
['AZ' 'GA' 'IL' 'CA' 'OR' 'NC' 'TX' 'VA' 'MO' 'CT' 'UT' 'FL' 'PA' 'MN'
 'NY' 'NJ' 'KY' 'OH' 'SC' 'RI' 'LA' 'MA' 'WA' 'WI' 'AL' 'CO' 'KS' 'NV'
 'AK' 'MD' 'WV' 'VT' 'MI' 'DC' 'SD' 'NH' 'AR' 'NM' 'MT' 'HI' 'WY' 'OK'
 'DE' 'MS' 'TN' 'IA' 'NE' 'ID' 'IN']
Unique values in column 'dti':
[13.33021434  8.72       20.         ...  2.34        2.24
  4.48      ]
Unique values in column 'delinq_2yrs':
[ 0  2  3  1  4  6  5  8  7  9 11]
Unique values in column 'inq_last_6mths':
[1 5 2 0 3 4 6 7 8]
Unique values in column 'open_acc':
[ 3  2 10 15  9  7  4 11 14 12 20  8  6 17  5 13 16 30 21 18 19 27 23 34
 25 22 24 26 32 28 29 33 31 39 35 36 38 44]
Unique values in column 'pub_rec':
[0 1 2 3 4]
Unique values in column 'revol_bal':
[13648.  1687.  2956. ... 13126. 14930. 26233.]
Unique values in column 'revol_util':
['83.70%' '9.40%' '98.50%' ... '49.63%' '0.04%' '7.28%']
Unique values in column 'total_acc':
[9 4 '10+' 3 7 6 8 5 2]
Unique values in column 'out_prncp':
[   0.    524.06 1849.1  ...   19.12   13.28   79.24]
Unique values in column 'out_prncp_inv':
[   0.    524.06 1844.43 ...   19.09   13.28   79.24]
Unique values in column 'total_pymnt':
[ 5863.155187   11374.41537756  3005.666844   ...  4015.96
 11652.75        3579.662273  ]
Unique values in column 'total_pymnt_inv':
[ 5833.84       10785.42002882  3005.67       ...  1624.17
  2122.53        1825.35      ]
Unique values in column 'total_rec_prncp':
[ 5000.          9260.50913275  2400.         ... 10463.04
  1496.83        8688.59      ]
Unique values in column 'total_rec_int':
[ 863.16  435.17  605.67 ...  609.26 2659.96  579.66]
Unique values in column 'total_rec_late_fee':
[ 0.         16.97       15.00000003 ... 18.98999996 24.01000007
 52.26222671]
Unique values in column 'recoveries':
[   0.    117.08  189.06 ...  151.2  1909.87  304.2 ]
Unique values in column 'collection_recovery_fee':
[  0.       1.11     2.09   ... 512.49    28.7262 668.36  ]
Unique values in column 'last_pymnt_d':
['Jan-15' 'Apr-13' 'Jun-14' 'May-16' 'Apr-12' 'Nov-12' 'Jun-13' 'Sep-13'
 'Jul-12' 'Oct-13' 'May-13' 'Feb-15' 'Aug-15' 'Oct-12' 'Sep-12' 'Dec-12'
 'Dec-14' 'Aug-13' 'Nov-13' 'Jan-14' 'Apr-14' 'Aug-14' 'Oct-14' 'Aug-12'
 'Jul-14' 'Jul-13' 'Jan-16' 'Feb-16' 'Apr-15' 'Feb-14' 'Sep-14' 'Jun-12'
 'Feb-13' 'Mar-13' 'May-14' 'Mar-15' 'Jan-13' 'Dec-13' 'Feb-12' 'Mar-14'
 'Sep-15' 'Nov-15' 'Mar-16' 'Jan-12' 'Oct-15' 'Nov-14' 'Mar-12' 'May-12'
 'Apr-16' 'Dec-15' 'Jun-15' 'May-15' 'Jul-15' 'Dec-11' 'Nov-11' 'Oct-11'
 'Sep-11' 'Aug-11' 'Jul-11' 'Jun-11' 'May-11' 'Apr-11' 'Mar-11' 'Feb-11'
 'Jan-11' 'Dec-10' 'Nov-10' 'Oct-10' 'Sep-10' 'Aug-10' 'Jul-10' 'Jun-10'
 'May-10' 'Apr-10' 'Mar-10' 'Feb-10' 'Jan-10' 'Dec-09' 'Nov-09' 'Oct-09'
 'Sep-09' 'Aug-09' 'Jul-09' 'Jun-09' 'May-09' 'Apr-09' 'Mar-09' 'Feb-09'
 'Jan-09' 'Dec-08' 'Oct-08' 'Aug-08' 'Jul-08' 'Sep-08' 'Jun-08' 'May-08'
 'Nov-08']
Unique values in column 'last_pymnt_amnt':
[ 171.62  119.66  649.91 ... 3891.08 1571.29 1016.15]
Unique values in column 'last_credit_pull_d':
```

```
['May-16' 'Sep-13' 'Apr-16' 'Jan-16' 'Dec-14' 'Aug-12' 'Mar-13' 'Dec-15'
 'Aug-13' 'Nov-12' 'Mar-14' 'Apr-15' 'May-14' 'Jul-15' 'Feb-16' 'Mar-16'
 'Sep-12' 'May-13' 'Jan-15' 'Jun-12' 'Mar-15' 'Dec-12' 'Sep-14' 'Feb-14'
 'Jun-15' 'Oct-13' 'Apr-14' 'Oct-14' 'Feb-13' 'Nov-15' 'Jul-14' 'Sep-15'
 'Oct-12' 'Nov-13' 'Nov-14' 'Feb-12' 'Oct-15' 'Apr-12' 'Aug-15' 'Jun-14'
 'Jan-12' 'Aug-14' 'Jun-13' 'Dec-13' 'May-12' 'Jul-12' 'Jan-14' 'Jul-13'
 'Apr-13' 'May-15' 'Feb-15' 'Mar-12' 'Nov-11' 'Dec-11' 'Jan-13' 'Oct-11'
 'Sep-11' 'Aug-11' 'Jul-11' 'Jun-11' 'May-11' 'Apr-11' 'Mar-11' 'Feb-11'
 'Jan-11' 'Dec-10' 'Nov-10' 'Oct-10' 'Sep-10' 'Aug-10' 'Jul-10' 'Jun-10'
 'May-10' 'Apr-10' 'Feb-10' 'Mar-10' 'Aug-07' 'Jan-10' 'Dec-09' 'Nov-09'
 'Oct-09' 'Sep-09' 'Jul-09' 'Aug-09' 'May-09' 'Jun-09' 'Apr-09' 'Mar-09'
 'Feb-09' 'Jan-09' 'Dec-08' 'Jun-08' 'Sep-08' 'May-08' 'Aug-08' 'Mar-08'
 'Oct-08']
Unique values in column 'pub_rec_bankruptcies':
[0. 1. 2.]
Unique values in column 'loanPeriod':
[36 60]
Unique values in column 'zip_code_num':
[860 309 606 917 972 852 280 900 958 774 853 913 245 951 641 921  67 890
 770 335 799 605 150 326 564 141  80 330 974 934 405 946 445 850 604 292
  88 180  29 700  10 441 104  61 616 947 914 765 980  17 752 787  77 540
 225 440 437 559 912 325 300 923 352  13 146  74 786 937 331 115 191 114
 908 902 992 750 950 329 226 614 802 672  83 100 926 931 712  60 707 342
 895 430 919 996 891 935 801 928 233 927 970 211 303  70 194 263 403 301
 553 993 312 432 602 216 151 971 305 334  50 129 925 483 760 200  85 981
 103 601 117  63 920 543 775 570  38 221 985 113 275 236 148  28 450 532
 729 321 959 941 955 217 880 660  62 193 761 857 306 271 142 956 983 945
 109 112 187 630 435 488 287 705 592 318 549 212 347 274 265 785  27  89
 813  69 260 201 349 322  75 124 940 967 111 773 997  76 538  21 304 234
 308 809  71 296 240 830  11 622 207 140 336 619 208 618  14 644 283 276
 631 243 960 181 922 224 975 105 986 218 652 782 410 480 719 982  65  81
 954 346 442  25 122 173 282 120  82 766 229 840 744 933 451 907 728 159
 333 293 701 984 811 597 957 165 720 119 359 195  84 969 924 531 716 337
 841 323 740 179 285 551 658 944 232 905 600 327 711 906 444 856 777  72
 554 145 537 152 847 295 829 320 131 939 572 281  64 550  78 452 778 313
 851 784 804 571 210 988 400 995 805  23 158 657  16  19 290 190 366  66
 991 968 721 439 640 546  24 751 431 741 904 156 316 299  87 739 949 261
  73 222 244 617  18 286 759 952 930 911 220 731 730 262 160  31  54 223
 272 882 557 797 725 130  30 206 324 170 291 161 647 916 665 209 915 110
  86 484 844  20 354 448 978 757 363 953 577 315 664 186 182 574 800 197
 137 314 755 973 603 481 780 894 341 178  68 565 611 288 443 662 874 560
 535 756 168 827 541 615 989  37 339 338 367 273  52 623 416 648 918 436
 898 674 496 294 762 128 903 328 932 650 246 633 666 228  15 302 573 118
 998 767 490 350 254 596 637  32 763 494 402 545 184 239 977 297 284 144
 748 310 147 153 544 948 576 976 107 846 344 351 754 910 656 357 791 493
 855 278 125 566 175 530 171 703 620 438 626 307 636 319 116 645 708 816
 625 133 612 961 238 166 361 231 241 826 783 793 646 188 108 653 871  57
 796 990 219 724 456 214 237 737 121 199 548 453 704 368 828 598 136 610
 722 743 810 706 235 139 613 454 317 746 446 486 863  33 279 407 794 457
 189 196 539 424 492 482 667 845 401 362 627 717 356 607 198 936 713 227
 883 563 893 806 360 172 422 768  34  12 594 215 628 749 101 814 255 745
 495 183 106 663 943  94 177 365 132 897 776 803 843 458 864 421 253 795
 727 528 270 277 735 447  79 358 815 250 230 790 884 242 534 404 397 870
 434 671 591 675  53 859 126 102 256 489 258 423 497 788 127 176 380  58
 635 498 599 822 638 723 449 420 726 185 963 298 257 575 624 134 877 499
 781 718 670 138  26 678 398 411 149 247 881 875 651 364 203 427 629 355
 174 547 567 558 135 157 999 808 634 455 143 154 562 779 561 734 655 812
 268  51 865 406 661 758 676 491 267 609 595 259 163 264  35 409 376 471
 820 375 747 123 714 590 639 412 425  22 608 369 164 433 825 266  96 251
 593 487 169 413 155 764 710 408 668  56 669 167 542 679 462 824 249 798
 370 485 654 289 807 252 556 353 677 769  90 371 831 527 736   7 332 468
 461  93 248 463 391 381 415 378 792 673 789 414 396 836  44 392 772 374
 823 395 394 965 838 390 388 386  40 385 379 681 837 373 753 834 479]
Unique values in column 'dti_level':
['E(>20)' 'A(<5)' 'B(5-10)' 'D(15-20)' 'C(10-15)']
```

```
Unique values in column 'salary_range':
['C(20K-50K)' 'B(10K-20K)' 'D(50K-75K)' 'A(<10K)' 'F(>100K)']
Unique values in column 'int_rate%':
[10.65 15.27 15.96 13.49 12.69  7.9  18.64 21.28 14.65  9.91 16.29  6.03
 11.71 12.42 16.77  7.51  8.9  18.25  6.62 19.91 17.27 14.27 17.58 21.67
 19.42 22.06 20.89 20.3  23.91 19.03 23.52 23.13 22.74 22.35 24.11  6.
 22.11  7.49 11.99  5.99 10.99  9.99 18.79 11.49 15.99 16.49  6.99 12.99
 15.23 14.79  5.42  8.49 10.59 17.49 15.62 21.36 19.29 13.99 18.39 16.89
 17.99 20.62 20.99 22.85 19.69 20.25 23.22 21.74 22.48 23.59 12.62 18.07
 11.63  7.91  7.42 11.14 20.2  12.12 19.39 16.11 17.54 22.64 16.59 17.19
 12.87 20.69  9.67 21.82 19.79 18.49 13.84 22.94 24.59 24.4  21.48 14.82
  7.29 17.88 20.11 16.02 17.51 13.43 14.91 13.06 15.28 15.65 17.14 11.11
 10.37 14.17 16.4   7.66 10.   10.74  5.79  6.92  9.63 14.54 12.68 18.62
 19.36 13.8  18.99 21.59 20.85 21.22 19.74 20.48  6.91 12.23 12.61 10.36
  6.17  6.54  9.25 16.69 15.95  8.88 13.35  9.62 16.32 12.98 14.83 13.72
 14.09 14.46 20.03 17.8  15.2  15.57 18.54 19.66 17.06 18.17 17.43 20.4
 20.77 18.91 21.14 17.44 13.23 11.12  7.88 13.61 10.38 17.56 17.93 15.58
 13.98 14.84 15.21  6.76  6.39 11.86  7.14 14.35 16.82 10.75 14.72 16.45
 20.53 19.41 20.16 21.27 18.3  18.67 19.04 20.9  21.64 12.73 10.25 13.11
 10.62 13.48 14.59 16.07 15.7   9.88 11.36 15.33 13.85 14.96 14.22  7.74
 13.22 13.57  8.59 17.04 14.61  8.94 12.18 11.83 11.48 16.35 13.92 15.31
 14.26 19.13 12.53 16.7  16.   17.39 18.09  7.4  18.43 17.74  7.05 20.52
 20.86 19.47 18.78 21.21 19.82 20.17 13.16  8.   13.47 12.21 16.63  9.32
 12.84 11.26 15.68 15.37 10.95 11.89 14.11 13.79  7.68 11.58  7.37 16.95
 15.05 18.53 14.74 14.42 18.21 17.26 18.84 17.9  19.16 13.67  9.38 12.72
 13.36 11.46 10.51  9.07 13.04 11.78 12.41 10.83 12.09 17.46 14.3  17.15
 15.25 10.2  15.88 14.93 16.2  18.72 14.62  8.32 14.12 10.96 10.33 10.01
 12.86 11.28 11.59  8.63 12.54 12.22 11.91 15.38 16.96  9.7  16.33 14.75
 13.17 15.07 16.01 10.71 10.64  9.76 11.34 10.39 13.87 11.03 11.66 13.24
 10.08  9.45 13.55 12.29 11.97 12.92 15.45 14.5  14.18 15.13 16.08 15.76
 17.03 10.46 13.93 10.78  9.51 12.36 13.3   9.83  9.01 10.91 10.28 12.49
 11.22]
Unique values in column 'int_rate_range':
['B(10%-15%)' 'C(15%-20%)' 'A(<10%)' 'D(>20%)']
Unique values in column 'loan_amt_range':
['C(5K-10K)' 'B(1K-5K)' 'D(10K-15K)' 'F(20K-30K)' 'E(15K-20K)' 'G(>30K)'
 'A(<1K)']
Unique values in column 'loan_installment_range':
['C(100-200)' 'B(50-100)' 'D(200-500)' 'A(<50)' 'E(500-750)' 'F(750-1000)'
 'G(>1000)']
Unique values in column 'int_rate_outlier':
[10.65 15.27 15.96 13.49 12.69  7.9  18.64 21.28 14.65  9.91 16.29  6.03
 11.71 12.42 16.77  7.51  8.9  18.25  6.62 19.91 17.27 14.27 17.58 21.67
 19.42 22.06 20.89 20.3  23.91 19.03 23.52 23.13 22.74 22.35 24.11  6.
 22.11  7.49 11.99  5.99 10.99  9.99 18.79 11.49 15.99 16.49  6.99 12.99
 15.23 14.79  5.42  8.49 10.59 17.49 15.62 21.36 19.29 13.99 18.39 16.89
 17.99 20.62 20.99 22.85 19.69 20.25 23.22 21.74 22.48 23.59 12.62 18.07
 11.63  7.91  7.42 11.14 20.2  12.12 19.39 16.11 17.54 22.64 16.59 17.19
 12.87 20.69  9.67 21.82 19.79 18.49 13.84 22.94 24.59 24.4  21.48 14.82
  7.29 17.88 20.11 16.02 17.51 13.43 14.91 13.06 15.28 15.65 17.14 11.11
 10.37 14.17 16.4   7.66 10.   10.74  5.79  6.92  9.63 14.54 12.68 18.62
 19.36 13.8  18.99 21.59 20.85 21.22 19.74 20.48  6.91 12.23 12.61 10.36
  6.17  6.54  9.25 16.69 15.95  8.88 13.35  9.62 16.32 12.98 14.83 13.72
 14.09 14.46 20.03 17.8  15.2  15.57 18.54 19.66 17.06 18.17 17.43 20.4
 20.77 18.91 21.14 17.44 13.23 11.12  7.88 13.61 10.38 17.56 17.93 15.58
 13.98 14.84 15.21  6.76  6.39 11.86  7.14 14.35 16.82 10.75 14.72 16.45
 20.53 19.41 20.16 21.27 18.3  18.67 19.04 20.9  21.64 12.73 10.25 13.11
 10.62 13.48 14.59 16.07 15.7   9.88 11.36 15.33 13.85 14.96 14.22  7.74
 13.22 13.57  8.59 17.04 14.61  8.94 12.18 11.83 11.48 16.35 13.92 15.31
 14.26 19.13 12.53 16.7  16.   17.39 18.09  7.4  18.43 17.74  7.05 20.52
 20.86 19.47 18.78 21.21 19.82 20.17 13.16  8.   13.47 12.21 16.63  9.32
 12.84 11.26 15.68 15.37 10.95 11.89 14.11 13.79  7.68 11.58  7.37 16.95
 15.05 18.53 14.74 14.42 18.21 17.26 18.84 17.9  19.16 13.67  9.38 12.72
 13.36 11.46 10.51  9.07 13.04 11.78 12.41 10.83 12.09 17.46 14.3  17.15
 15.25 10.2  15.88 14.93 16.2  18.72 14.62  8.32 14.12 10.96 10.33 10.01
```

```
12.86 11.28 11.59  8.63 12.54 12.22 11.91 15.38 16.96  9.7  16.33 14.75
13.17 15.07 16.01 10.71 10.64  9.76 11.34 10.39 13.87 11.03 11.66 13.24
10.08  9.45 13.55 12.29 11.97 12.92 15.45 14.5  14.18 15.13 16.08 15.76
17.03 10.46 13.93 10.78  9.51 12.36 13.3   9.83  9.01 10.91 10.28 12.49
11.22]
```

Data Preparation

In [ ]:

```
loandf.describe()
```

Out[ ]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	installment	ann
count	3.783500e+04	3.783500e+04	37835.000000	37835.000000	37835.000000	37835.000000	37835.(
mean	6.899869e+05	8.597532e+05	10351.972555	10290.237422	9973.276689	310.186449	63976.:
std	2.029235e+05	2.542853e+05	5096.186360	5064.622375	5299.651093	153.526530	25183.:
min	5.473400e+04	8.036400e+04	1000.000000	1000.000000	750.000000	32.440000	19200.(
25%	5.210765e+05	6.731990e+05	6000.000000	6000.000000	5914.107605	188.020000	45000.(
50%	6.693350e+05	8.555920e+05	10000.000000	10000.000000	9871.252594	308.709751	62322.(
75%	8.392890e+05	1.049062e+06	13000.000000	13000.000000	12902.582030	391.510000	76800.(
max	1.077501e+06	1.314167e+06	24975.000000	24975.000000	24736.560330	762.950000	141996.(

8 rows × 28 columns

Checking for the class imbalance problem in loan status prediction

In [ ]:

```
# Filter out rows with 'Current' in the 'Loan_Status' column
loandf = loandf[loandf['loan_status'] != 'Current']

# Reset the index if needed
loandf.reset_index(drop=True, inplace=True)
```

In [ ]:

```
loandf['loan_status'].value_counts()
```

Out[ ]:

```
Fully Paid     31534
Charged Off     5203
Name: loan_status, dtype: int64
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
loandf['loan_status'] = label_encoder.fit_transform(loandf['loan_status'])
```

In [ ]:

```
loandf['loan_status'].value_counts()
```

Out[ ]:

```
1    31534
0     5203
```

```
Name: loan_status, dtype: int64
```

In [ ]:

```
loandf.head(20)

loandf['emp_length'].value_counts()
```

Out[ ]:

```
10+ years    8359
< 1 year     4322
2 years      4196
3 years      3940
4 years      3283
5 years      3147
1 year       3062
6 years      2132
7 years      1685
8 years      1405
9 years      1206
Name: emp_length, dtype: int64
```

In [ ]:

```
ordinal_mapping = {
    'emp_length': {'10+ years': 10, '< 1 year': 1, '2 years': 2, '3 years': 3, '4 years
}

# Use map to replace categorical values with integers based on the defined order
loandf.replace(ordinal_mapping, inplace=True)
```

In [ ]:

```
loandf['emp_length'].value_counts()
```

Out[ ]:

```
10    8359
1     7384
2     4196
3     3940
4     3283
5     3147
6     2132
7     1685
8     1405
9     1206
Name: emp_length, dtype: int64
```

In [ ]:

```
loandf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36737 entries, 0 to 36736
Data columns (total 46 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   id                      36737 non-null  int64  
 1   member_id               36737 non-null  int64  
 2   loan_amnt               36737 non-null  float64
 3   funded_amnt             36737 non-null  float64
 4   funded_amnt_inv         36737 non-null  float64
 5   installment             36737 non-null  float64
 6   grade                   36737 non-null  object 
 7   sub_grade               36737 non-null  object 
 8   emp_length              36737 non-null  int64  
 9   home_ownership          36737 non-null  object 
 10  annual_inc              36737 non-null  float64
 11  verification_status     36737 non-null  object 
```

```
 12  issue_d                  36737 non-null  object
 13  loan_status              36737 non-null  int64
 14  purpose                  36737 non-null  object
 15  addr_state               36737 non-null  object
 16  dti                      36737 non-null  float64
 17  delinq_2yrs              36737 non-null  int64
 18  inq_last_6mths           36737 non-null  int64
 19  open_acc                 36737 non-null  int64
 20  pub_rec                  36737 non-null  int64
 21  revol_bal                36737 non-null  float64
 22  revol_util               36737 non-null  object
 23  total_acc                36737 non-null  object
 24  out_prncp                36737 non-null  float64
 25  out_prncp_inv            36737 non-null  float64
 26  total_pymnt              36737 non-null  float64
 27  total_pymnt_inv          36737 non-null  float64
 28  total_rec_prncp          36737 non-null  float64
 29  total_rec_int            36737 non-null  float64
 30  total_rec_late_fee       36737 non-null  float64
 31  recoveries               36737 non-null  float64
 32  collection_recovery_fee  36737 non-null  float64
 33  last_pymnt_d             36737 non-null  object
 34  last_pymnt_amnt          36737 non-null  float64
 35  last_credit_pull_d       36737 non-null  object
 36  pub_rec_bankruptcies     36737 non-null  float64
 37  loanPeriod               36737 non-null  int64
 38  zip_code_num             36737 non-null  int64
 39  dti_level                36737 non-null  object
 40  salary_range             36737 non-null  object
 41  int_rate%                36737 non-null  float64
 42  int_rate_range           36737 non-null  object
 43  loan_amt_range           36737 non-null  object
 44  loan_installment_range   36737 non-null  object
 45  int_rate_outlier         36737 non-null  float64
dtypes: float64(20), int64(10), object(16)
memory usage: 12.9+ MB
```

Data Mapping for large catagorical veriable to numerical values for better understading in application development.

In [ ]:

```
check_data = loandf[['loan_amnt', 'installment', 'loan_status',
                     'annual_inc', 'dti', 'home_ownership', 'purpose',
                     'loanPeriod', 'addr_state', 'emp_length',
                      'int_rate%']]
```

In [ ]:

```
check_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36737 entries, 0 to 36736
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   loan_amnt       36737 non-null  float64
 1   installment     36737 non-null  float64
 2   loan_status     36737 non-null  int64
 3   annual_inc      36737 non-null  float64
 4   dti             36737 non-null  float64
 5   home_ownership  36737 non-null  object
 6   purpose         36737 non-null  object
 7   loanPeriod      36737 non-null  int64
 8   addr_state      36737 non-null  object
 9   emp_length      36737 non-null  int64
```

```
 10  int_rate%       36737 non-null  float64
dtypes: float64(5), int64(3), object(3)
memory usage: 3.1+ MB
```

In [ ]:

```
check_data.nunique()
```

Out[ ]:

```
loan_amnt           715
installment       12678
loan_status           2
annual_inc         4246
dti                2173
home_ownership        4
purpose              14
loanPeriod            2
addr_state           49
emp_length           10
int_rate%           336
dtype: int64
```

In [ ]:

```
unique_p = check_data['home_ownership'].unique().tolist()
print(unique_p)
```

```
['RENT', 'OWN', 'MORTGAGE', 'OTHER']
```

In [ ]:

```
ownership_mapping = {
    "RENT": 1,
    "OWN": 2,
    "MORTGAGE": 3,
    "OTHER": 4
}
```

In [ ]:

```
purpose_mapping = {
    "credit_card": 1,
    "car": 2,
    "small_business": 3,
    "other": 4,
    "wedding": 5,
    "debt_consolidation": 6,
    "home_improvement": 7,
    "major_purchase": 8,
    "medical": 9,
    "moving": 10,
    "vacation": 11,
    "house": 12,
    "renewable_energy": 13,
    "educational": 14,
}
```

In [ ]:

```
# Dictionary to map state abbreviations to integer values
state_mapping = {
    "AZ": 1,
    "GA": 2,
    "IL": 3,
    "CA": 4,
    "NC": 5,
    "TX": 6,
```

```
    "VA": 7,
    "MO": 8,
    "CT": 9,
    "UT": 10,
    "FL": 11,
    "PA": 12,
    "MN": 13,
    "NY": 14,
    "NJ": 15,
    "OR": 16,
    "KY": 17,
    "OH": 18,
    "SC": 19,
    "RI": 20,
    "LA": 21,
    "MA": 22,
    "WA": 23,
    "WI": 24,
    "AL": 25,
    "NV": 26,
    "AK": 27,
    "CO": 28,
    "MD": 29,
    "WV": 30,
    "VT": 31,
    "MI": 32,
    "DC": 33,
    "SD": 34,
    "NH": 35,
    "AR": 36,
    "NM": 37,
    "KS": 38,
    "HI": 39,
    "OK": 40,
    "MT": 41,
    "WY": 42,
    "DE": 43,
    "MS": 44,
    "TN": 45,
    "IA": 46,
    "NE": 47,
    "ID": 48,
    "IN": 49
}


import pandas as pd

# Create new columns with mapped values
check_data['home_ownership'] = check_data['home_ownership'].map(ownership_mapping)
check_data['purpose'] = check_data['purpose'].map(purpose_mapping)
check_data['state'] = check_data['addr_state'].map(state_mapping)
```

In [ ]:

```
check_data.describe()
```

Out[ ]:

	loan_amnt	installment	loan_status	annual_inc	dti	home_ownership	pu
count	36737.000000	36737.000000	36737.000000	36737.000000	36737.000000	36737.000000	36737.0

	loan_amnt	installment	loan_status	annual_inc	dti	home_ownership	pu
mean	10271.473727	308.479979	0.858372	63884.925931	13.343163	1.969867	5.3
std	5072.002077	153.148373	0.348673	25181.066709	5.442177	0.966158	2.4
min	1000.000000	32.440000	0.000000	19200.000000	0.000000	1.000000	1.0
25%	6000.000000	187.080000	1.000000	45000.000000	9.300000	1.000000	4.0
50%	10000.000000	308.410000	1.000000	62000.000000	13.330214	2.000000	6.0
75%	13000.000000	389.300000	1.000000	76160.000000	17.490000	3.000000	6.0
max	24975.000000	762.950000	1.000000	141996.000000	25.500000	4.000000	14.0

SMOTE

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

data = check_data[['loan_amnt', 'installment', 'loan_status',
                   'annual_inc', 'dti', 'home_ownership', 'purpose',
                   'loanPeriod', 'state',
                    'int_rate%']]


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

X['home_ownership'] = pd.to_numeric(X['home_ownership'], errors='coerce')
X['purpose'] = pd.to_numeric(X['purpose'], errors='coerce')
X['loanPeriod'] = pd.to_numeric(X['loanPeriod'], errors='coerce')
X['state'] = pd.to_numeric(X['state'], errors='coerce')
# Split the data into training and testing sets for Loan_Status prediction
X_train_loan_status, X_test_loan_status, y_train_loan_status, y_test_loan_status = trai

# Apply SMOTE to balance the class distribution for Loan_Status prediction
smote_loan_status = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled_loan_status, y_train_resampled_loan_status = smote_loan_status.fit_re
```

Loan Status

Random Forest

In [ ]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
RFC_loan_status = RandomForestClassifier(random_state=42)
RFC_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the test set for Loan_Status prediction
y_pred_loan_status = RFC_loan_status.predict(X_test_loan_status)
```

```
# Evaluate the classifier's performance for Loan_Status prediction
accuracy_loan_status = accuracy_score(y_test_loan_status, y_pred_loan_status)
report_loan_status = classification_report(y_test_loan_status, y_pred_loan_status)

print("Loan_Status Prediction Accuracy:", accuracy_loan_status)


mse = mean_squared_error(y_test_loan_status, y_pred_loan_status)
r2 = r2_score(y_test_loan_status, y_pred_loan_status)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')

print("Loan_Status Classification Report:\n", report_loan_status)
```

```
Loan_Status Prediction Accuracy: 0.7857920522591181
MSE: 0.2142
R^2: -0.7904
Loan_Status Classification Report:
               precision    recall  f1-score   support

           0       0.23      0.23      0.23      1021
           1       0.88      0.88      0.88      6327

    accuracy                           0.79      7348
   macro avg       0.55      0.55      0.55      7348
weighted avg       0.79      0.79      0.79      7348
```

Decision Tree

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier


DTC_loan_status = DecisionTreeClassifier(random_state=42)
DTC_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the test set for Loan_Status prediction
y_pred_loan_status = DTC_loan_status.predict(X_test_loan_status)

# Evaluate the classifier's performance for Loan_Status prediction
accuracy_loan_status = accuracy_score(y_test_loan_status, y_pred_loan_status)
report_loan_status = classification_report(y_test_loan_status, y_pred_loan_status)

print("Loan_Status Prediction Accuracy:", accuracy_loan_status)


mse = mean_squared_error(y_test_loan_status, y_pred_loan_status)
r2 = r2_score(y_test_loan_status, y_pred_loan_status)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')

print("Loan_Status Classification Report:\n", report_loan_status)
```

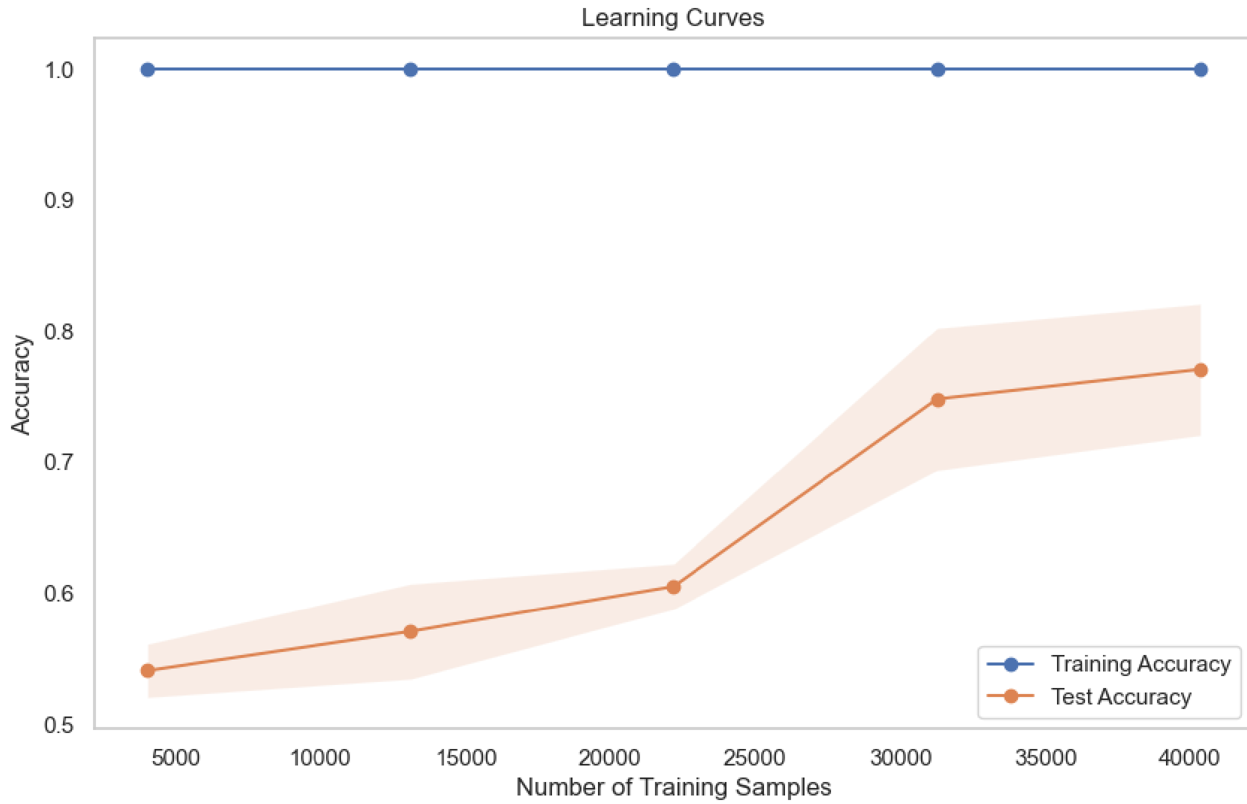
```
Loan_Status Prediction Accuracy: 0.7129831246597713
MSE: 0.2870
R^2: -1.3990
Loan_Status Classification Report:
               precision    recall  f1-score   support
```

```
           0       0.18      0.31      0.23      1021
           1       0.87      0.78      0.82      6327

    accuracy                           0.71      7348
   macro avg       0.53      0.54      0.53      7348
weighted avg       0.78      0.71      0.74      7348
```

In [ ]:

```
from sklearn.model_selection import learning_curve

# Initialize the Decision Tree Classifier
DTC_loan_status = DecisionTreeClassifier(random_state=42)

# Fit the classifier on the training data
DTC_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the training and test sets
y_pred_train = DTC_loan_status.predict(X_train_resampled_loan_status)
y_pred_test = DTC_loan_status.predict(X_test_loan_status)

# Evaluate the classifier's performance on both sets
accuracy_train = accuracy_score(y_train_resampled_loan_status, y_pred_train)
accuracy_test = accuracy_score(y_test_loan_status, y_pred_test)

# Print the accuracy on both sets
print("Training Accuracy:", accuracy_train)
print("Test Accuracy:", accuracy_test)

# Check for overfitting using learning curves
train_sizes, train_scores, test_scores = learning_curve(DTC_loan_status, X_train_resamp

# Calculate the mean and standard deviation of training and test scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot learning curves
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label="Training Accuracy", marker='o', linestyle='-')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1
plt.plot(train_sizes, test_mean, label="Test Accuracy", marker='o', linestyle='-')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15)
plt.xlabel("Number of Training Samples")
plt.ylabel("Accuracy")
plt.title("Learning Curves")
plt.legend(loc="best")
plt.grid()
plt.show()
```

```
Training Accuracy: 1.0
Test Accuracy: 0.7129831246597713
```



XGBoost

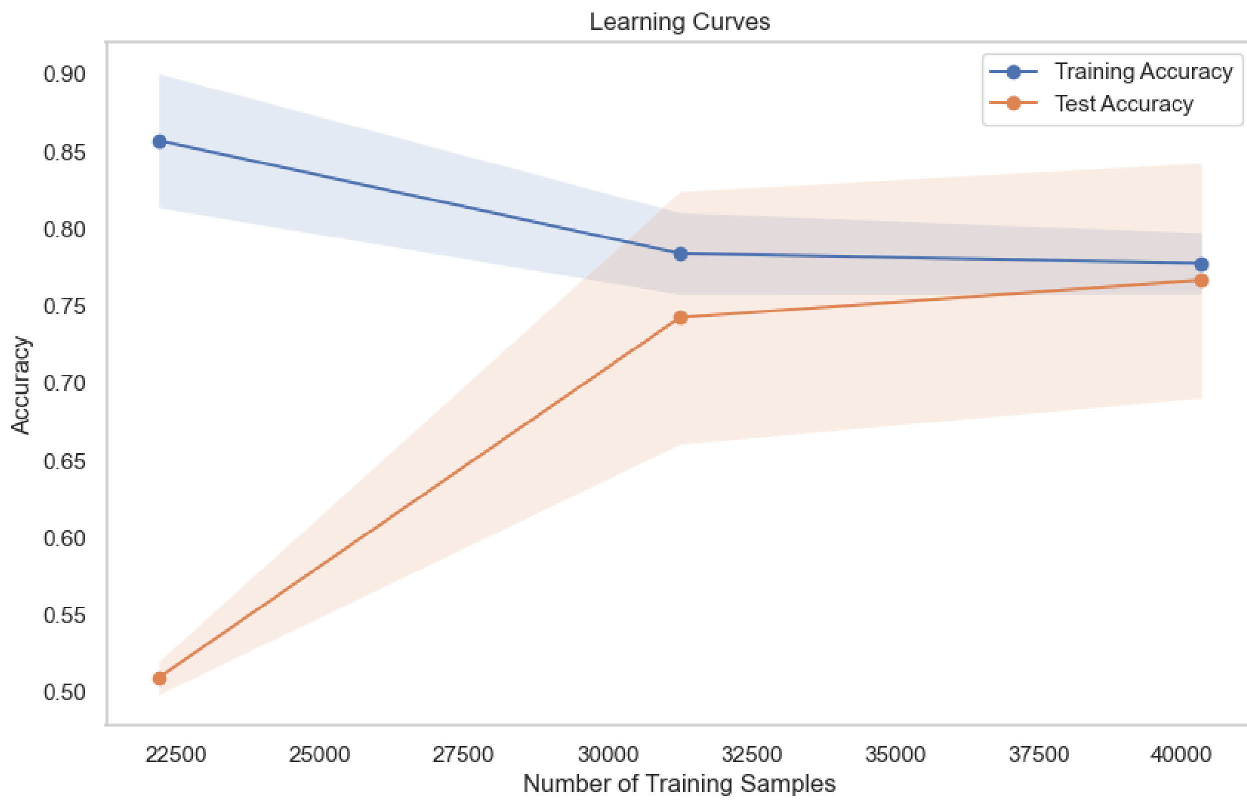
In [ ]:
```
import xgboost as xgb

xgb_loan_status = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, r

# Fit the classifier on the resampled data for Loan_Status prediction
xgb_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the test set for Loan_Status prediction
y_pred_loan_status = xgb_loan_status.predict(X_test_loan_status)

# Evaluate the classifier's performance for Loan_Status prediction
accuracy_loan_status = accuracy_score(y_test_loan_status, y_pred_loan_status)
report_loan_status = classification_report(y_test_loan_status, y_pred_loan_status)

print("Loan_Status Prediction Accuracy:", accuracy_loan_status)


mse = mean_squared_error(y_test_loan_status, y_pred_loan_status)
r2 = r2_score(y_test_loan_status, y_pred_loan_status)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')
```

```
Loan_Status Prediction Accuracy: 0.7431954273271638
MSE: 0.2568
R^2: -1.1464
```

In [ ]:
```
# Initialize the Decision Tree Classifier
clf_loan_status = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, r

# Fit the classifier on the training data
```

```
clf_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the training and test sets
y_pred_train = clf_loan_status.predict(X_train_resampled_loan_status)
y_pred_test = clf_loan_status.predict(X_test_loan_status)

# Evaluate the classifier's performance on both sets
accuracy_train = accuracy_score(y_train_resampled_loan_status, y_pred_train)
accuracy_test = accuracy_score(y_test_loan_status, y_pred_test)

# Print the accuracy on both sets
print("Training Accuracy:", accuracy_train)
print("Test Accuracy:", accuracy_test)

# Check for overfitting using learning curves
train_sizes, train_scores, test_scores = learning_curve(clf_loan_status, X_train_resamp

# Calculate the mean and standard deviation of training and test scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot learning curves
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label="Training Accuracy", marker='o', linestyle='-')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1
plt.plot(train_sizes, test_mean, label="Test Accuracy", marker='o', linestyle='-')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.15)
plt.xlabel("Number of Training Samples")
plt.ylabel("Accuracy")
plt.title("Learning Curves")
plt.legend(loc="best")
plt.grid()
plt.show()
```

```
Training Accuracy: 0.7755782124013171
Test Accuracy: 0.7431954273271638
```



Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression

# Create an XGBoost classifier
xgb_loan_status = LogisticRegression(random_state=42)

# Fit the classifier on the resampled data for Loan_Status prediction
xgb_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the test set for Loan_Status prediction
y_pred_loan_status = xgb_loan_status.predict(X_test_loan_status)

# Evaluate the classifier's performance for Loan_Status prediction
accuracy_loan_status = accuracy_score(y_test_loan_status, y_pred_loan_status)
report_loan_status = classification_report(y_test_loan_status, y_pred_loan_status)

print("Loan_Status Prediction Accuracy:", accuracy_loan_status)


mse = mean_squared_error(y_test_loan_status, y_pred_loan_status)
r2 = r2_score(y_test_loan_status, y_pred_loan_status)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')

print("Loan_Status Classification Report:\n", report_loan_status)
```

```
Loan_Status Prediction Accuracy: 0.5748502994011976
MSE: 0.4251
R^2: -2.5535
Loan_Status Classification Report:
               precision    recall  f1-score   support
```

```
           0       0.19      0.62      0.29      1021
           1       0.90      0.57      0.70      6327

    accuracy                           0.57      7348
   macro avg       0.55      0.60      0.49      7348
weighted avg       0.80      0.57      0.64      7348
```

Intrest Rate

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Feature Selection
selected_features = ['loan_amnt', 'installment',
                     'annual_inc', 'dti', 'home_ownership', 'purpose',
                     'loanPeriod', 'state'
                     ]

# Target variable
target = 'int_rate%'

# Splitting the dataset into features (X) and target (y)
X = data[selected_features]
y = data[target]


# Splitting data into training and testing sets
X_train_int, X_test_int, y_train_int, y_test_int = train_test_split(X, y, test_size=0.2

# Identifying categorical columns
categorical_features = X.select_dtypes(include=['object', 'bool']).columns.tolist()

# Creating a Column Transformer for Preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), [col for col in selected_features if col not in categ
        ('cat', OneHotEncoder(), categorical_features)
    ])
```

In [ ]:

```
print(X_train_int.shape)
print(X_test_int.shape)
print(y_train_int.shape)
print(y_test_int.shape)
```

```
(29389, 8)
(7348, 8)
(29389,)
(7348,)
```

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

rf_model = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', RandomForestRegressor(random_state=42))])
```

```
# Training the model
rf_model.fit(X_train_int, y_train_int)

# Predicting on test data
y_pred_int = rf_model.predict(X_test_int)

# Evaluating the model
mse = mean_squared_error(y_test_int, y_pred_int)
r2 = r2_score(y_test_int, y_pred_int)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")
```

```
Mean Squared Error (MSE): 1.963013150546976
R² Score: 0.8577321548047496
```

XGBOOST

In [ ]:

```
from xgboost import XGBRegressor

# XGBoost Regressor model
xgboost_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', xgb.XGBRegressor(objective ='reg:squarederror', random_state=42))
])
# Fit the regressor on the training data
xgboost_model.fit(X_train_int, y_train_int)

# Make predictions on the test set
y_pred_int = xgboost_model.predict(X_test_int)

# Evaluate the regressor's performance
# For regression, consider using metrics like Mean Squared Error (MSE), Mean Absolute E
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test_int, y_pred_int)
r2 = r2_score(y_test_int, y_pred_int)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')
```

```
MSE: 2.1126
R^2: 0.8469
```

Elactic Net Regression

In [ ]:

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.compose import ColumnTransformer  # Assuming you have a preprocessor

# Define your preprocessor here (example placeholder, customize as needed)
# preprocessor = ColumnTransformer(transformers=[...])

# Define alpha and l1_ratio hyperparameters
alpha = 0.1  # Adjust as needed
l1_ratio = 0.5  # Adjust as needed

# ElasticNet Regressor model
```

```
elastic_net_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', ElasticNet(alpha=alpha, l1_ratio=l1_ratio))
])

# Fit the regressor on the training data
elastic_net_model.fit(X_train_int, y_train_int)

# Make predictions on the test set
y_pred_int = elastic_net_model.predict(X_test_int)

# Evaluate the regressor's performance
mse = mean_squared_error(y_test_int, y_pred_int)
r2 = r2_score(y_test_int, y_pred_int)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')
```

```
MSE: 10.2876
R^2: 0.2544
```

In [ ]:

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.compose import ColumnTransformer  # Assuming you have a preprocessor

# Assuming the preprocessor is defined somewhere above
# preprocessor = ColumnTransformer(transformers=[...])

# Define the pipeline with a placeholder for ElasticNet
elastic_net_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', ElasticNet())
])

# Define the parameter grid, note the use of 'regressor__' prefix to specify ElasticNet
param_grid = {
    'regressor__alpha': [0.01, 0.1, 1],  # Adjust these values and ranges as needed
    'regressor__l1_ratio': [0.2, 0.5, 0.8]  # Adjust these values and ranges as needed
}

# Setup GridSearchCV
cv = GridSearchCV(elastic_net_pipeline, param_grid, cv=5, scoring='r2')  # Or use anoth

# Fit GridSearchCV
cv.fit(X_train_int, y_train_int)

# Print best parameters and best score
print("Best parameters:", cv.best_params_)
print("Best score:", cv.best_score_)

# Optionally, you can use the best estimator to make predictions
y_pred_int = cv.predict(X_test_int)

# Evaluate the model's performance with the best found parameters
mse = mean_squared_error(y_test_int, y_pred_int)
r2 = r2_score(y_test_int, y_pred_int)
```

```
print(f'MSE with Best Parameters: {mse:.4f}')
print(f'R^2 with Best Parameters: {r2:.4f}')
```

```
Best parameters: {'regressor__alpha': 0.01, 'regressor__l1_ratio': 0.5}
Best score: 0.26023339285104047
MSE with Best Parameters: 10.1749
R^2 with Best Parameters: 0.2626
```

Final Model Preparations

Loan Status

In [ ]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
RFC_loan_status = RandomForestClassifier(random_state=42)
RFC_loan_status.fit(X_train_resampled_loan_status, y_train_resampled_loan_status)

# Make predictions on the test set for Loan_Status prediction
y_pred_loan_status = RFC_loan_status.predict(X_test_loan_status)

# Evaluate the classifier's performance for Loan_Status prediction
accuracy_loan_status = accuracy_score(y_test_loan_status, y_pred_loan_status)
report_loan_status = classification_report(y_test_loan_status, y_pred_loan_status)

print("Loan_Status Prediction Accuracy:", accuracy_loan_status)


mse = mean_squared_error(y_test_loan_status, y_pred_loan_status)
r2 = r2_score(y_test_loan_status, y_pred_loan_status)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')

print("Loan_Status Classification Report:\n", report_loan_status)
```

```
Loan_Status Prediction Accuracy: 0.7857920522591181
MSE: 0.2142
R^2: -0.7904
Loan_Status Classification Report:
               precision    recall  f1-score   support

           0       0.23      0.23      0.23      1021
           1       0.88      0.88      0.88      6327

    accuracy                           0.79      7348
   macro avg       0.55      0.55      0.55      7348
weighted avg       0.79      0.79      0.79      7348
```

In [ ]:

```
X_test_loan_status.head(10)
```

Out[ ]:

	loan_amnt	installment	annual_inc	dti	home_ownership	purpose	loanPeriod	state
28833	10000.00000	329.120000	63658.278175	13.330214	2	6	36	14
4009	10000.00000	312.910000	57000.000000	13.090000	1	6	36	13
17990	7200.00000	218.360000	31500.000000	6.630000	1	6	36	8
263	18000.00000	571.560000	78000.000000	7.000000	1	1	36	14

	loan_amnt	installment	annual_inc	dti	home_ownership	purpose	loanPeriod	state
34921	17000.00000	574.100000	51996.000000	16.870000	3	6	36	2
15557	10299.70276	308.709751	85000.000000	13.330214	3	3	60	6
9731	10299.70276	308.709751	86000.000000	6.000000	1	6	36	7
32720	5175.00000	172.330000	63658.278175	13.120000	1	6	36	11
29669	12000.00000	407.090000	60000.000000	8.420000	3	7	36	9
8993	18000.00000	666.110000	58240.000000	7.110000	1	6	36	22

In [ ]:

```
test = RFC_loan_status.predict([[17000.00000, 574.100000, 51996.000000, 16.870000, 3, 6
```

In [ ]:

```
y_pred_prob_loan_status = RFC_loan_status.predict_proba([[17000.00000, 574.100000, 5199
```

In [ ]:

```
print(test)

print(y_pred_prob_loan_status)
```

```
[1]
[[0.27 0.73]]
```

In [ ]:

```
import joblib

# Assuming 'rf_model' is your optimized Random Forest pipeline
joblib.dump(RFC_loan_status, 'rf_model1.joblib')
```

Out[ ]: ['rf_model1.joblib']

Interest Rate

In [ ]:

```
from xgboost import XGBRegressor

# XGBoost Regressor model
xgboost_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', xgb.XGBRegressor(objective ='reg:squarederror', random_state=42))
])
# Fit the regressor on the training data
xgboost_model.fit(X_train_int, y_train_int)

# Make predictions on the test set
y_pred_int = xgboost_model.predict(X_test_int)



# Evaluate the regressor's performance
# For regression, consider using metrics like Mean Squared Error (MSE), Mean Absolute Er
from sklearn.metrics import mean_squared_error, r2_score
```

```python
mse = mean_squared_error(y_test_int, y_pred_int)
r2 = r2_score(y_test_int, y_pred_int)

print(f'MSE: {mse:.4f}')
print(f'R^2: {r2:.4f}')
```

```
MSE: 2.1126
R^2: 0.8469
```

In [ ]:
```python
X_test_int.head(10)
```

Out[ ]:

	loan_amnt	installment	annual_inc	dti	home_ownership	purpose	loanPeriod	state
28833	10000.00000	329.120000	63658.278175	13.330214	2	6	36	14
4009	10000.00000	312.910000	57000.000000	13.090000	1	6	36	13
17990	7200.00000	218.360000	31500.000000	6.630000	1	6	36	8
263	18000.00000	571.560000	78000.000000	7.000000	1	1	36	14
34921	17000.00000	574.100000	51996.000000	16.870000	3	6	36	2
15557	10299.70276	308.709751	85000.000000	13.330214	3	3	60	6
9731	10299.70276	308.709751	86000.000000	6.000000	1	6	36	7
32720	5175.00000	172.330000	63658.278175	13.120000	1	6	36	11
29669	12000.00000	407.090000	60000.000000	8.420000	3	7	36	9
8993	18000.00000	666.110000	58240.000000	7.110000	1	6	36	22

In [ ]:
```python
# Assuming these are the correct column names based on your model's training data
column_names = ['loan_amnt', 'installment',
                    'annual_inc', 'dti', 'home_ownership', 'purpose',
                    'loanPeriod', 'state']

# Your input data for prediction
input_data = [[180000.00000, 400.110000, 58240.000000, 7.110000, 1, 4, 36, 22]]

# Convert input data to a pandas DataFrame
input_df = pd.DataFrame(input_data, columns=column_names)

# Make predictions using the dataframe
y_pred = xgboost_model.predict(input_df)

print(y_pred)
```

```
[10.090835]
```

In [ ]:
```python
import joblib

# Assuming 'rf_model' is your optimized Random Forest pipeline
joblib.dump(xgboost_model, 'XGBModel1.joblib')
```

Out[ ]: ['XGBModel1.joblib']