PREDICTIVE MODEL

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import tensorflow as tf
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier


import os
os.environ['OMP_NUM_THREADS']='1'


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

     Mounted at /content/drive/
```

```
fileName = '/content/drive/MyDrive/LBG Data.csv'
```

```
data = pd.read_csv(fileName)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18324 entries, 0 to 18323
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   id                      18324 non-null  int64  
 1   addr_state              18324 non-null  object 
 2   annual_inc              18324 non-null  float64
 3   emp_length              17150 non-null  float64
 4   emp_title               17042 non-null  object 
 5   home_ownership          18324 non-null  object 
 6   installment             18324 non-null  float64
 7   loan_amnt               18324 non-null  int64  
 8   purpose                 18324 non-null  object 
 9   term                    18324 non-null  int64  
 10  int_rate                18324 non-null  float64
 11  avg_cur_bal             17758 non-null  float64
 12  inq_last_12m            9395 non-null   float64
 13  max_bal_bc              9395 non-null   float64
 14  mo_sin_old_il_acct      17192 non-null  float64
 15  mo_sin_old_rev_tl_op    17760 non-null  float64
 16  mo_sin_rcnt_rev_tl_op   17760 non-null  float64
 17  mo_sin_rcnt_tl          17760 non-null  float64
 18  mort_acc                17926 non-null  float64
 19  mths_since_last_delinq  9276 non-null   float64
 20  num_bc_tl               17760 non-null  float64
 21  num_il_tl               17760 non-null  float64
 22  num_op_rev_tl           17760 non-null  float64
 23  num_tl_90g_dpd_24m      17760 non-null  float64
```

```
 24  num_tl_op_past_12m      17760 non-null  float64
 25  open_acc                18324 non-null  int64
 26  percent_bc_gt_75        17714 non-null  float64
 27  pub_rec_bankruptcies    18324 non-null  int64
 28  total_acc               18324 non-null  int64
 29  total_bal_ex_mort       17926 non-null  float64
 30  loan_status             18324 non-null  object
dtypes: float64(20), int64(6), object(5)
memory usage: 4.3+ MB
```

```
data.describe()
```

	id	annual_inc	emp_length	installment	loan_amnt	term	int_rate	avg_cur_bal	in
count	1.832400e+04	1.832400e+04	17150.000000	18324.000000	18324.000000	18324.000000	18324.000000	17758.000000	9
mean	6.832645e+07	8.017611e+04	6.073178	467.543006	15522.661537	42.815979	13.850700	13466.600011	
std	4.245703e+07	6.487345e+04	3.639694	278.099801	9349.294243	10.822769	4.822253	16550.730832	
min	3.009180e+05	3.000000e+03	0.500000	30.650000	1000.000000	36.000000	5.310000	0.000000	
25%	3.491424e+07	4.700000e+04	2.000000	259.302500	8000.000000	36.000000	10.490000	3129.000000	
50%	6.838023e+07	6.500000e+04	6.000000	397.480000	14000.000000	36.000000	13.330000	7137.000000	
75%	9.730784e+07	9.500000e+04	10.000000	635.720000	21000.000000	60.000000	16.990000	18436.500000	
max	1.708249e+08	2.616000e+06	10.000000	1503.890000	40000.000000	60.000000	30.990000	341236.000000	

8 rows × 26 columns

```
data.isna()
#to see the number of  missing values
```

	id	addr_state	annual_inc	emp_length	emp_title	home_ownership	installment	loan_amnt	purpose	term	...
0	False	False	False	False	False	False	False	False	False	False	...
1	False	False	False	False	False	False	False	False	False	False	...
2	False	False	False	False	False	False	False	False	False	False	...
3	False	False	False	False	False	False	False	False	False	False	...
4	False	False	False	False	False	False	False	False	False	False	...
...	...	...	...	...	...	...	...	...	...	...	...
18319	False	False	False	False	False	False	False	False	False	False	...
18320	False	False	False	False	False	False	False	False	False	False	...
18321	False	False	False	False	False	False	False	False	False	False	...
18322	False	False	False	False	False	False	False	False	False	False	...
18323	False	False	False	False	False	False	False	False	False	False	...

18324 rows × 31 columns

```
data.isna().sum()
```

```
id                         0
addr_state                 0
annual_inc                 0
emp_length              1174
emp_title               1282
home_ownership             0
installment                0
loan_amnt                  0
purpose                    0
term                       0
int_rate                   0
avg_cur_bal              566
inq_last_12m            8929
max_bal_bc              8929
mo_sin_old_il_acct      1132
mo_sin_old_rev_tl_op     564
```

```
mo_sin_rcnt_rev_tl_op      564
mo_sin_rcnt_tl             564
mort_acc                   398
mths_since_last_delinq    9048
num_bc_tl                  564
num_il_tl                  564
num_op_rev_tl              564
num_tl_90g_dpd_24m         564
num_tl_op_past_12m         564
open_acc                     0
percent_bc_gt_75           610
pub_rec_bankruptcies         0
total_acc                    0
total_bal_ex_mort          398
loan_status                  0
dtype: int64
```

```
set(data.duplicated())
```

```
{False}
```

```
# we can pass the result of the df.duplicated into a set to see if there is any instance of True
# No duplicates in the data
set(data.duplicated())
```

```
{False}
```

```
data.shape
```

```
(18324, 31)
```

```
# Checking the class imbalance
data['loan_status'].value_counts(normalize=True)
```

```
Fully Paid     0.786837
Charged Off    0.213163
Name: loan_status, dtype: float64
```

```
df=data
```

```
df.head()
```

	id	addr_state	annual_inc	emp_length	emp_title	home_ownership	installment	loan_amnt	purpose
0	802173	CA	72000.0	3.0	CA. Dept. Of Corrections	MORTGAGE	395.66	12000	debt_consolidation
1	14518910	TX	97500.0	1.0	Curriculum & Implementation Manager	RENT	966.47	35000	debt_consolidation
2	54333324	NY	120000.0	1.0	Senior manager	RENT	806.57	25000	credit_card
3	62247022	CA	130000.0	10.0	Border Patrol Agent	RENT	846.17	25225	debt_consolidation
4	71986114	TX	58296.0	10.0	Account Manager	MORTGAGE	41.79	1200	other

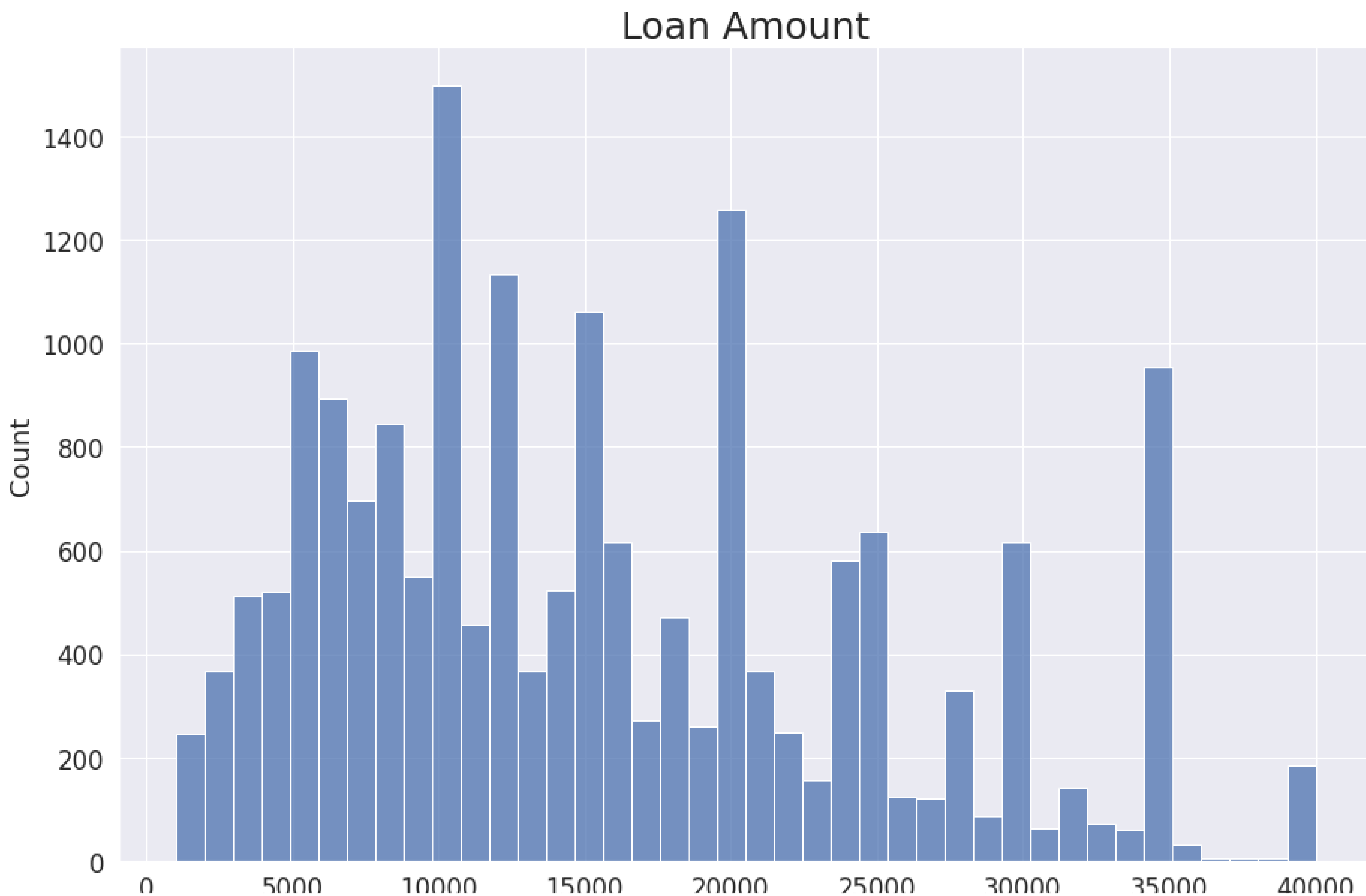
5 rows × 31 columns

```
df.info()
```

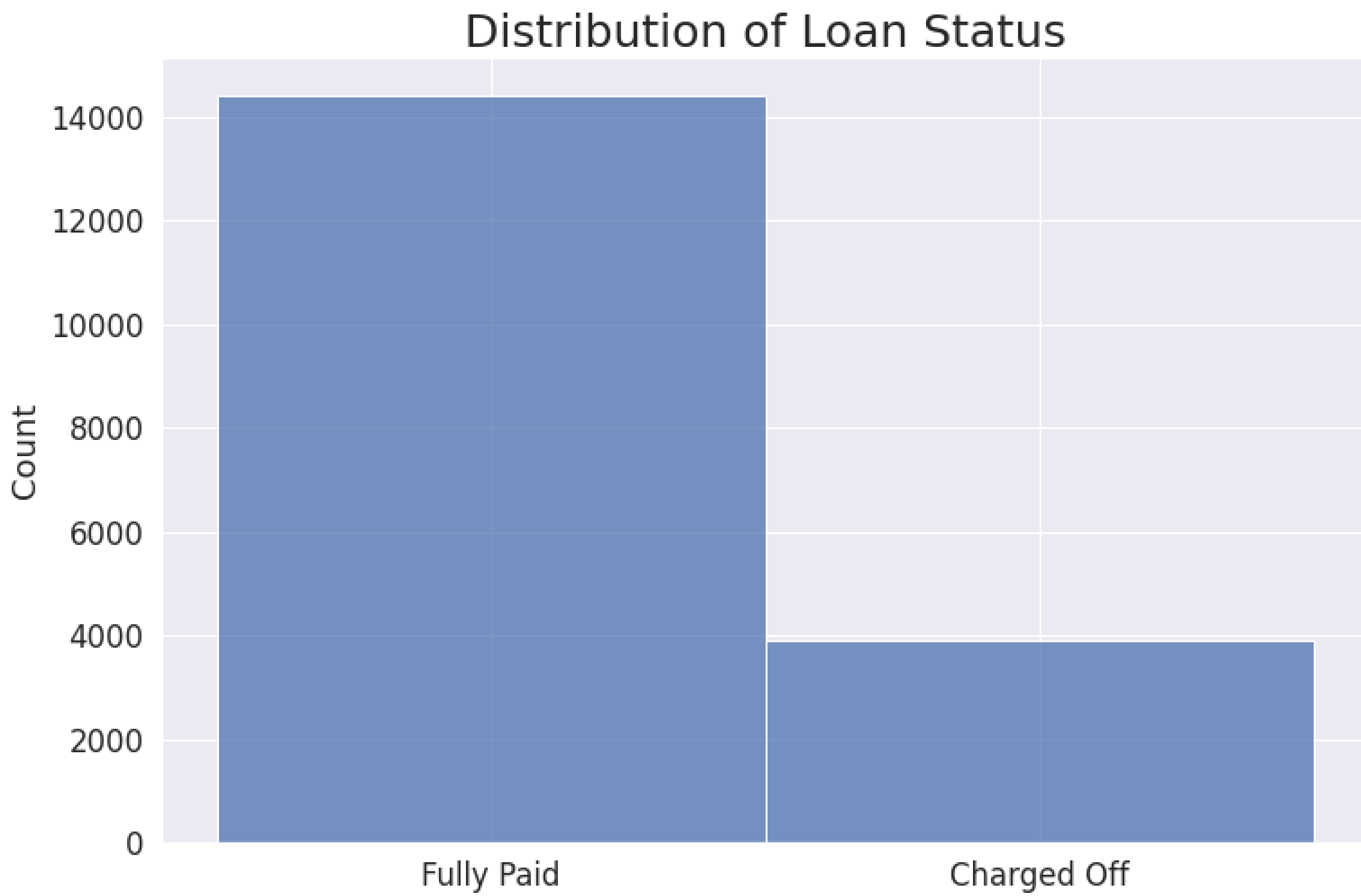
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18324 entries, 0 to 18323
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      18324 non-null  int64
```

```
 1   addr_state              18324 non-null  object
 2   annual_inc              18324 non-null  float64
 3   emp_length              17150 non-null  float64
 4   emp_title               17042 non-null  object
 5   home_ownership          18324 non-null  object
 6   installment             18324 non-null  float64
 7   loan_amnt               18324 non-null  int64
 8   purpose                 18324 non-null  object
 9   term                    18324 non-null  int64
 10  int_rate                18324 non-null  float64
 11  avg_cur_bal             17758 non-null  float64
 12  inq_last_12m            9395 non-null   float64
 13  max_bal_bc              9395 non-null   float64
 14  mo_sin_old_il_acct      17192 non-null  float64
 15  mo_sin_old_rev_tl_op    17760 non-null  float64
 16  mo_sin_rcnt_rev_tl_op   17760 non-null  float64
 17  mo_sin_rcnt_tl          17760 non-null  float64
 18  mort_acc                17926 non-null  float64
 19  mths_since_last_delinq  9276 non-null   float64
 20  num_bc_tl               17760 non-null  float64
 21  num_il_tl               17760 non-null  float64
 22  num_op_rev_tl           17760 non-null  float64
 23  num_tl_90g_dpd_24m      17760 non-null  float64
 24  num_tl_op_past_12m      17760 non-null  float64
 25  open_acc                18324 non-null  int64
 26  percent_bc_gt_75        17714 non-null  float64
 27  pub_rec_bankruptcies    18324 non-null  int64
 28  total_acc               18324 non-null  int64
 29  total_bal_ex_mort       17926 non-null  float64
 30  loan_status             18324 non-null  object
dtypes: float64(20), int64(6), object(5)
memory usage: 4.3+ MB
```

```
plt.figure(figsize=(15,10))
sns.histplot(df.loan_amnt)
plt.title('Loan Amount', fontsize = 25)
plt.show()
```




```
plt.figure(figsize=(12,8))
sns.histplot(df.loan_status)
plt.title('Distribution of Loan Status', fontsize = 25)
plt.show()
```




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18324 entries, 0 to 18323
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       18324 non-null  int64
 1   addr_state               18324 non-null  object
 2   annual_inc               18324 non-null  float64
 3   emp_length               17150 non-null  float64
 4   emp_title                17042 non-null  object
```

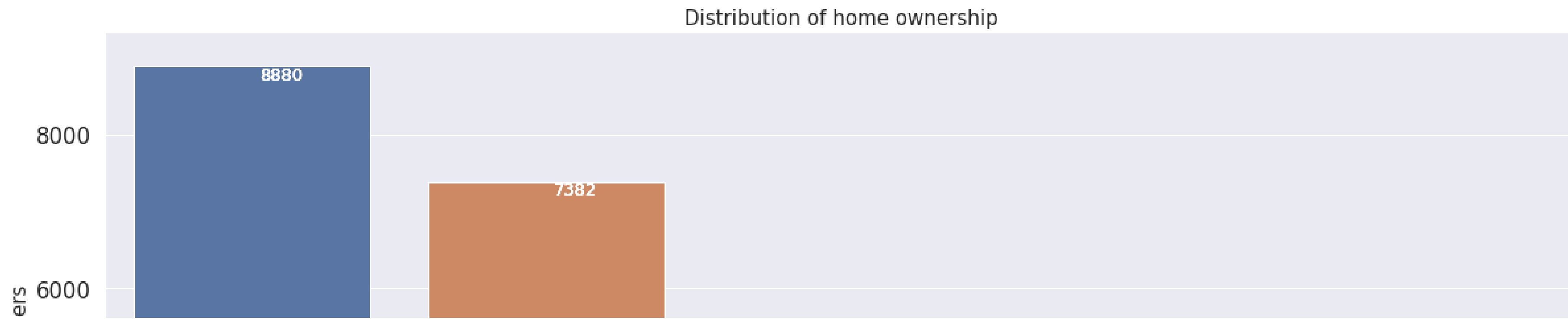
```
 5   home_ownership          18324 non-null  object 
 6   installment             18324 non-null  float64
 7   loan_amnt               18324 non-null  int64  
 8   purpose                 18324 non-null  object 
 9   term                    18324 non-null  int64  
 10  int_rate                18324 non-null  float64
 11  avg_cur_bal             17758 non-null  float64
 12  inq_last_12m            9395 non-null   float64
 13  max_bal_bc              9395 non-null   float64
 14  mo_sin_old_il_acct      17192 non-null  float64
 15  mo_sin_old_rev_tl_op    17760 non-null  float64
 16  mo_sin_rcnt_rev_tl_op   17760 non-null  float64
 17  mo_sin_rcnt_tl          17760 non-null  float64
 18  mort_acc                17926 non-null  float64
 19  mths_since_last_delinq  9276 non-null   float64
 20  num_bc_tl               17760 non-null  float64
 21  num_il_tl               17760 non-null  float64
 22  num_op_rev_tl           17760 non-null  float64
 23  num_tl_90g_dpd_24m      17760 non-null  float64
 24  num_tl_op_past_12m      17760 non-null  float64
 25  open_acc                18324 non-null  int64  
 26  percent_bc_gt_75        17714 non-null  float64
 27  pub_rec_bankruptcies    18324 non-null  int64  
 28  total_acc               18324 non-null  int64  
 29  total_bal_ex_mort       17926 non-null  float64
 30  loan_status             18324 non-null  object 
dtypes: float64(20), int64(6), object(5)
memory usage: 4.3+ MB
```

```
# Home ownership and loan status
```

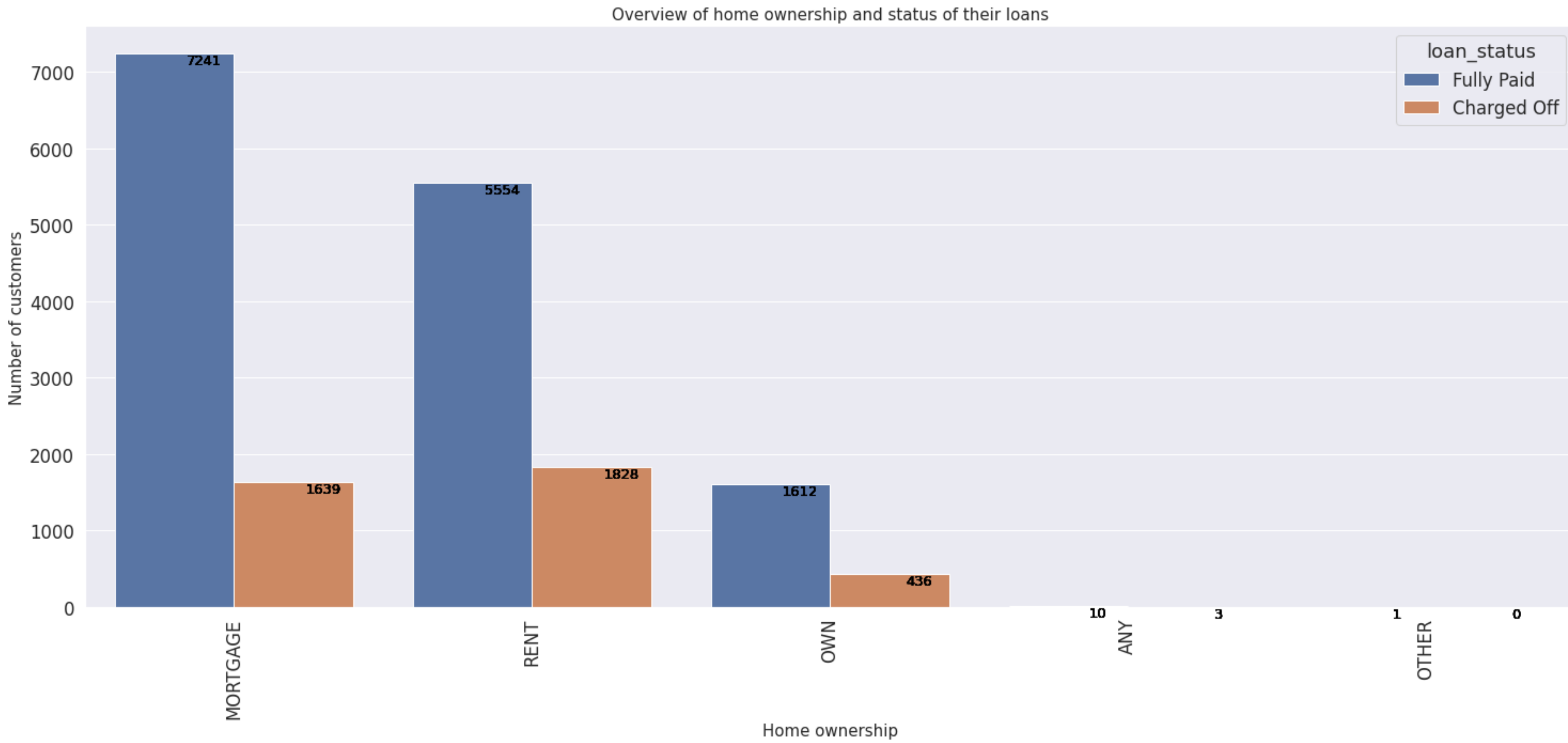
```
df['home_ownership'].value_counts()
```

```
MORTGAGE    8880
RENT        7382
OWN         2048
ANY           13
OTHER          1
Name: home_ownership, dtype: int64
```

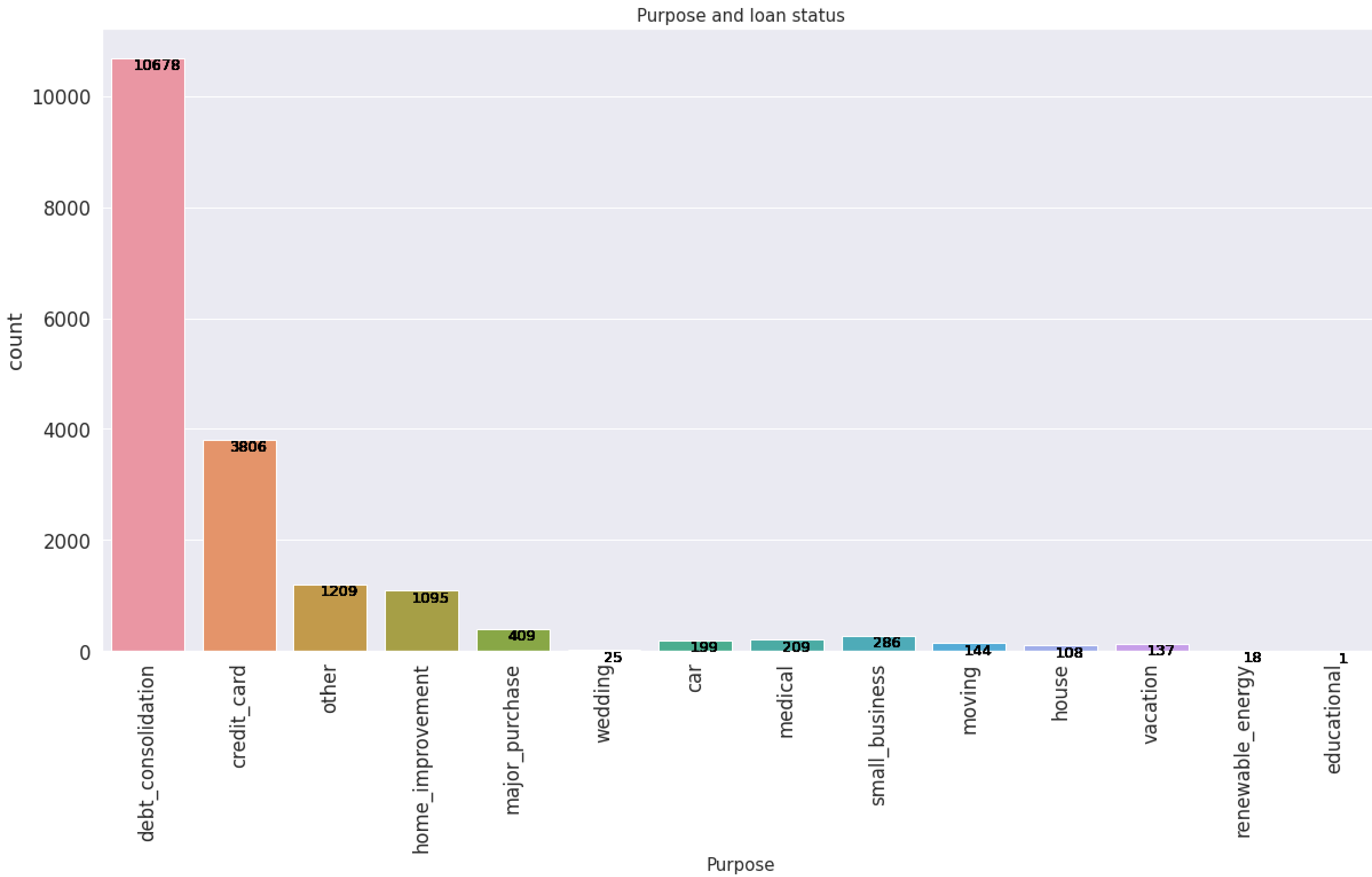
```python
plt.figure(figsize=(20,10))
ax=sns.countplot(x='home_ownership', data=df)
ax.set_title('Distribution of home ownership' , fontsize = 15)
plt.xlabel('Home ownership', fontsize=15)
plt.ylabel('Number of customers', fontsize=15)
plt.xticks(rotation='vertical')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.5, p.get_height()), ha='center', va='top', color='white', size=13)
```



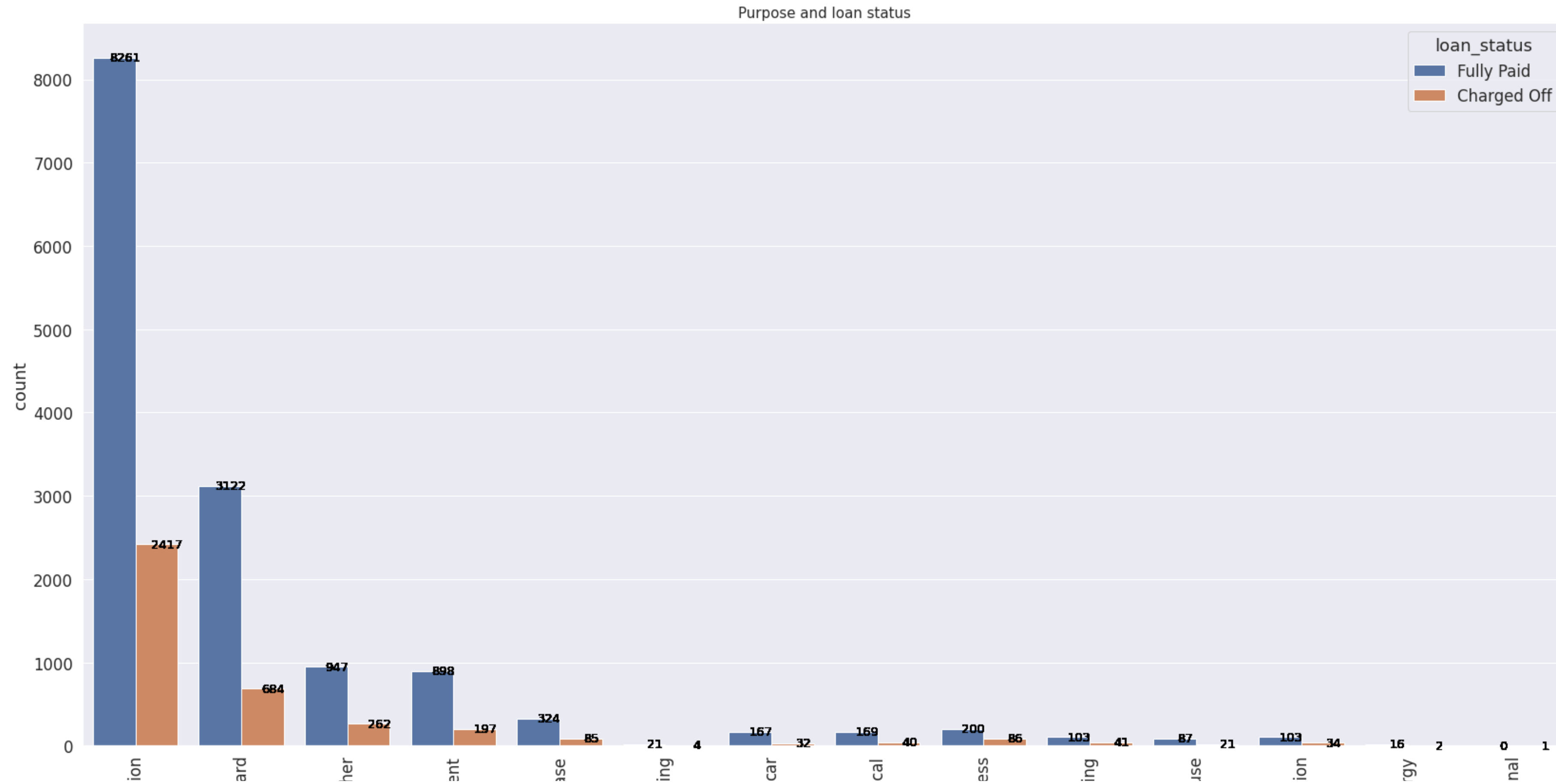

```
plt.figure(figsize=(25,10))
ax=sns.countplot(x='home_ownership',hue ='loan_status', data=df)
ax.set_title('Overview of home ownership and status of their loans' , fontsize = 15)
plt.xlabel('Home ownership', fontsize=15)
plt.ylabel('Number of customers', fontsize=15)
plt.xticks(rotation='vertical')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='top', color='black',  size=13)
```



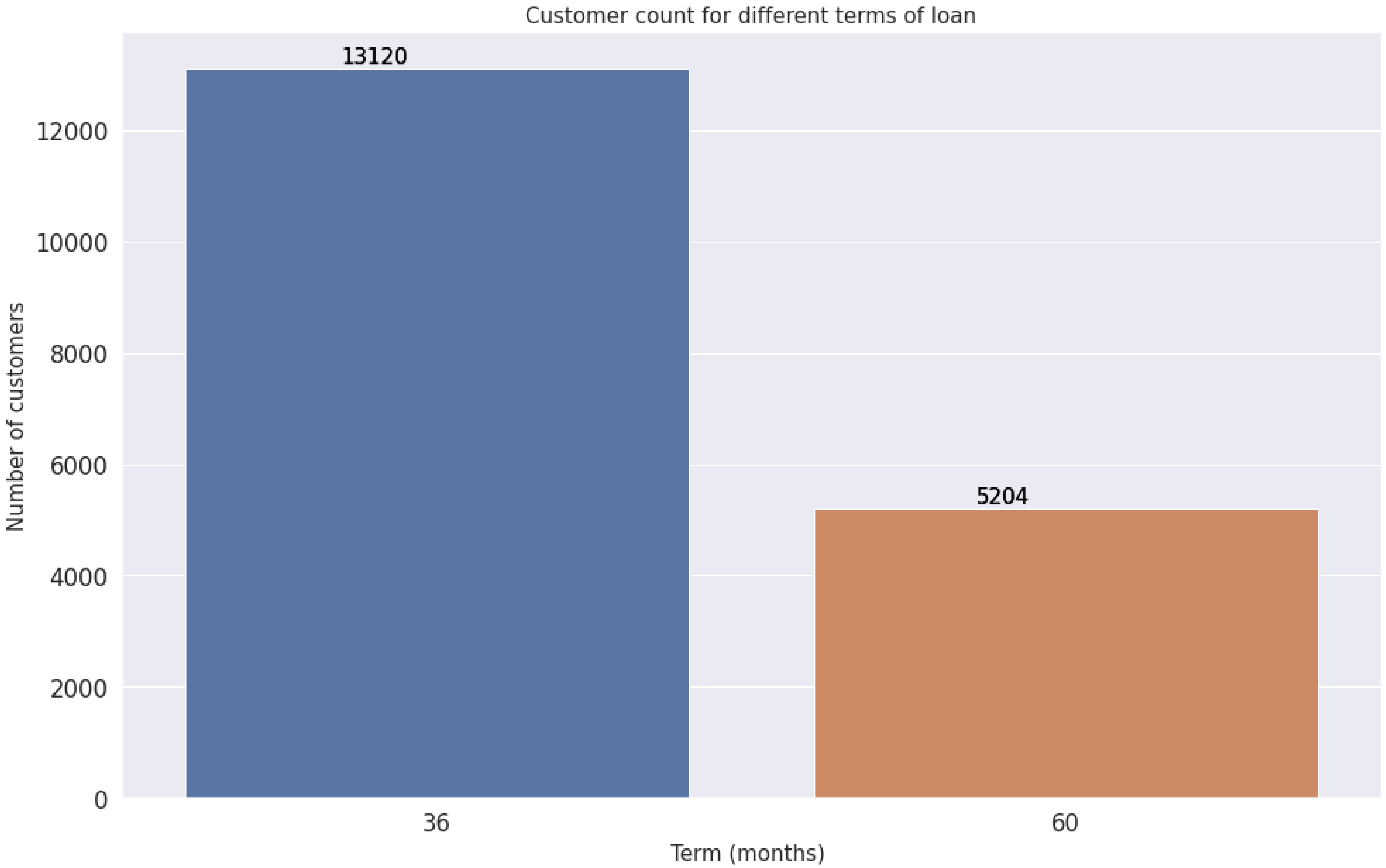

```
 # purpose and loan status

plt.figure(figsize=(20,10))
ax=sns.countplot(x='purpose', data=df)
ax.set_title('Purpose and loan status' , fontsize = 15)
plt.xlabel('Purpose', fontsize=15)
plt.xticks(rotation='vertical')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.5, p.get_height()), ha='center', va='top', color='black', size=13)
```


Purpose and loan status
count
10000
8000
6000
4000
2000
0
10678
3806
1209
1095
409
25
199
209
286
144
108
137
18
1
debt_consolidation
credit_card
other
home_improvement
major_purchase
wedding
car
medical
small_business
moving
house
vacation
renewable_energy
educational
Purpose

```
plt.figure(figsize=(28,14))
ax=sns.countplot(x='purpose', hue='loan_status', data=df)
ax.set_title('Purpose and loan status' , fontsize = 15)
plt.xlabel('Purpose', fontsize=15)
plt.xticks(rotation='vertical')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='center', color='black', size=13)
```

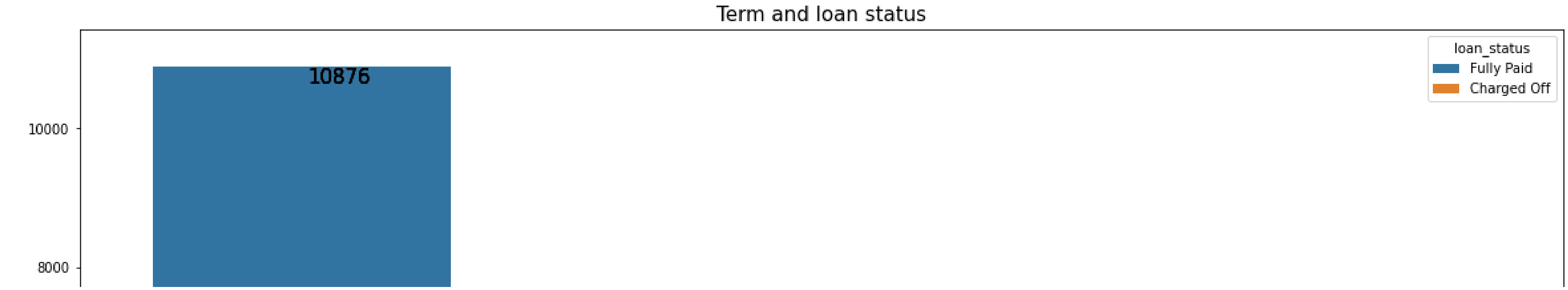



```
# term and loan status

plt.figure(figsize=(16,10))
ax=sns.countplot(x='term', data=df)
ax.set_title('Customer count for different terms of loan' , fontsize = 15)
plt.xlabel('Term (months) ', fontsize=15)
plt.ylabel('Number of customers', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```



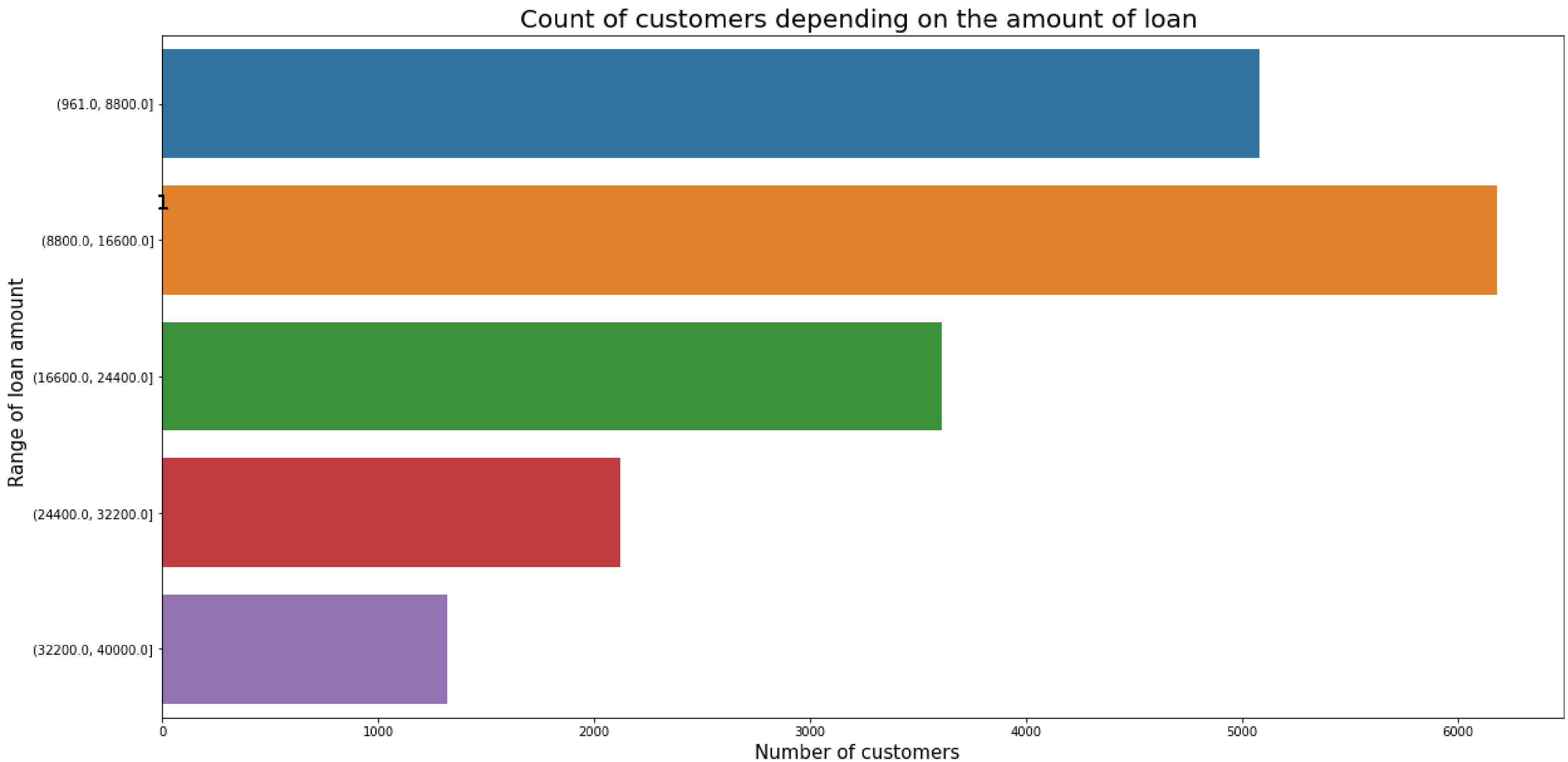

```
# term and loan status
```

```
plt.figure(figsize=(20,11))
ax=sns.countplot(x='term', hue='loan_status', data=df)
ax.set_title('Term and loan status' , fontsize = 15)
plt.xlabel('Term (months)', fontsize=15)
plt.ylabel('Number of customers', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.25, p.get_height()), ha='center', va='top', color='black', size=15)
```



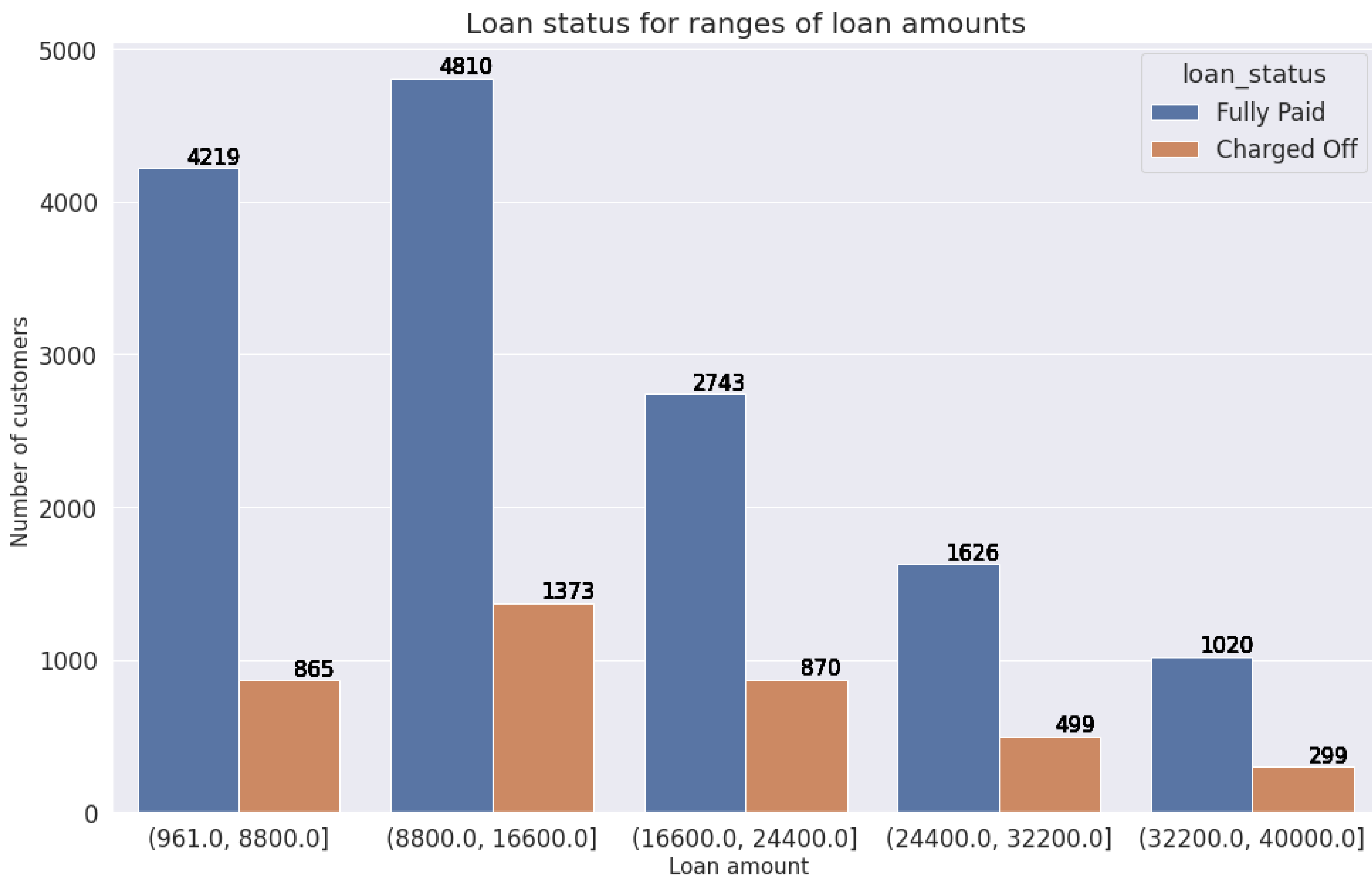
```
# To put the loan amount in bins
bins = pd.cut(df['loan_amnt'], 5)
```

```
 # loan status

plt.figure(figsize=(20,10))
ax=sns.countplot(y=bins)
sns.set(font_scale=1.5)
ax.set_title('Count of customers depending on the amount of loan ' , fontsize = 20)
plt.xlabel('Number of customers  ', fontsize=15)
plt.ylabel('Range of loan amount  ', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```

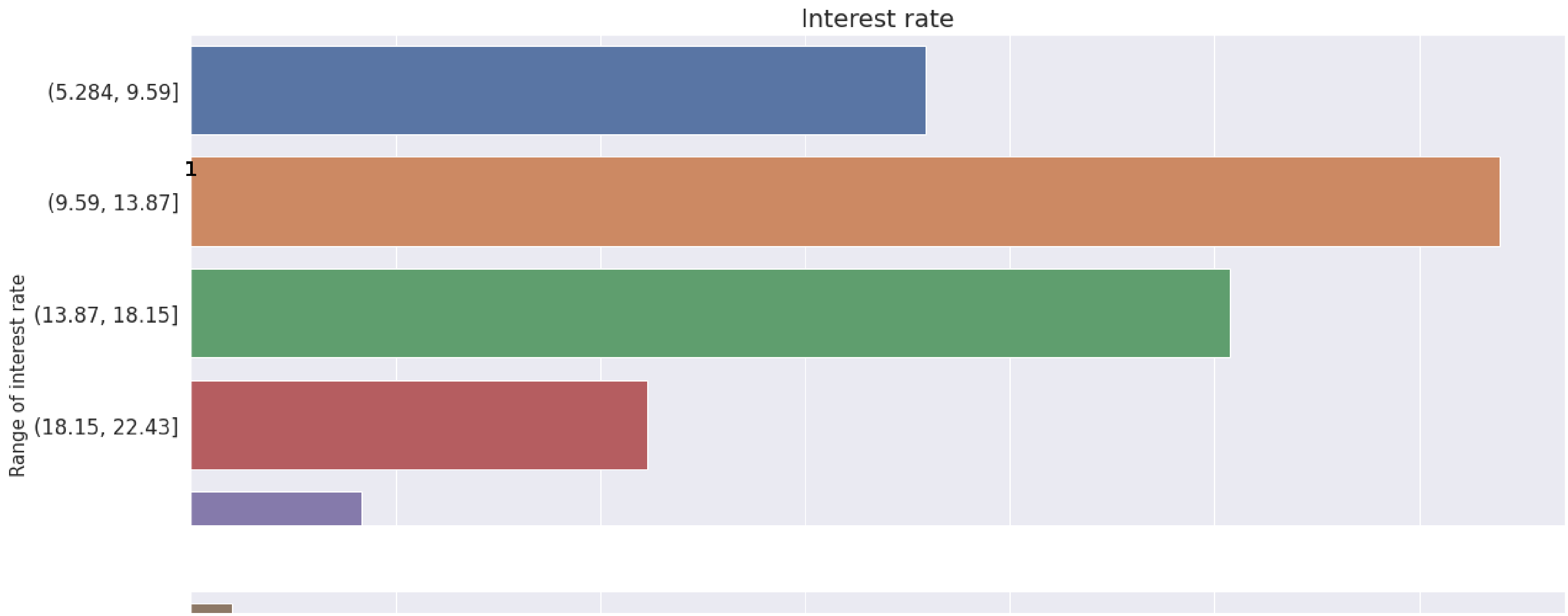



```
 # loan amount and loan status

plt.figure(figsize=(16,10))
sns.set(font_scale=1.5)
ax=sns.countplot(x=bins,  hue='loan_status', data=df)
ax.set_title('Loan status for ranges of loan amounts' , fontsize = 20)
plt.xlabel('Loan amount  ', fontsize=15)
plt.ylabel('Number of customers ', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
```

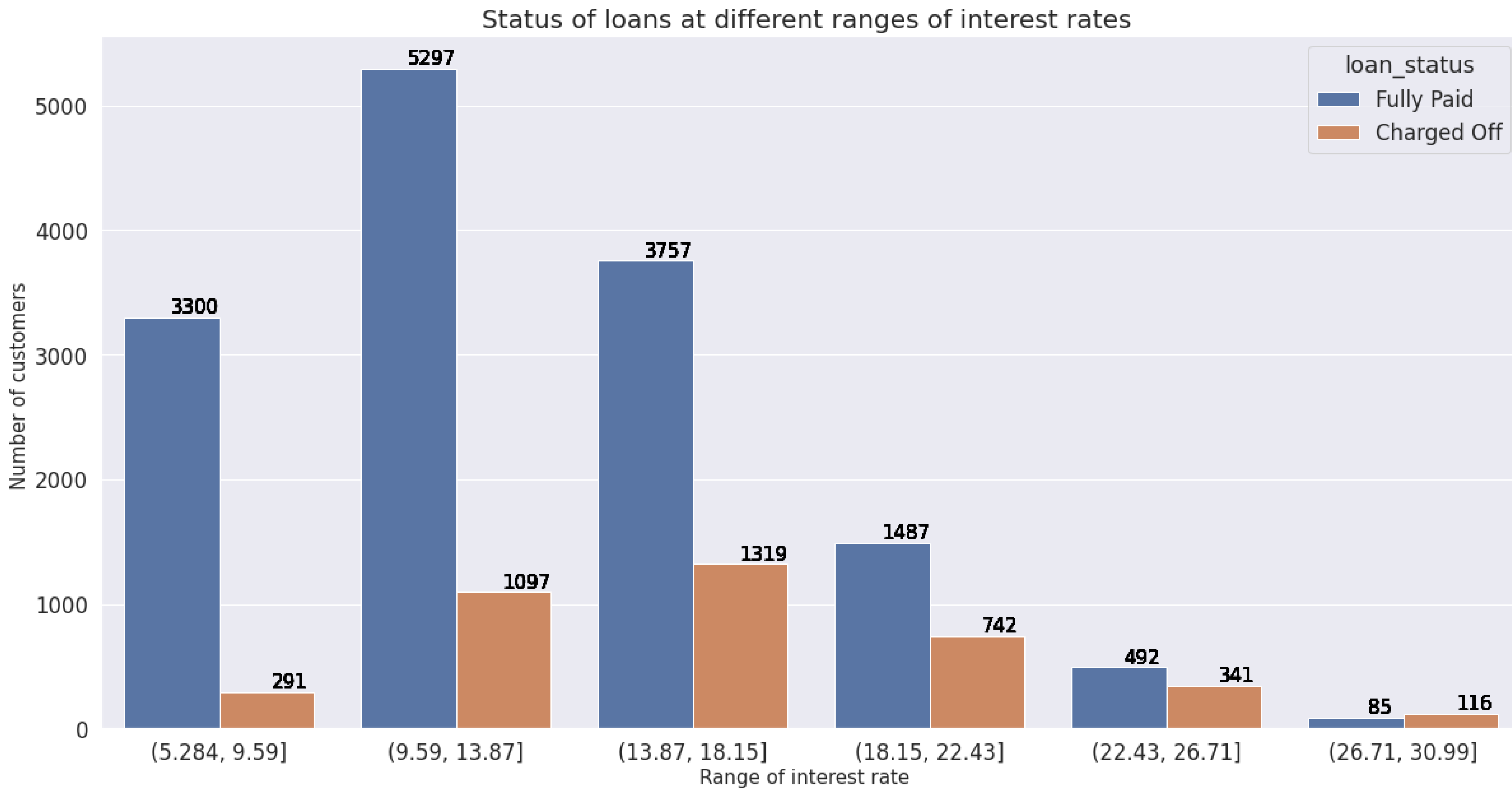
```
ax.annotate(format(p.get_height(), '.0f'),
        (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```



```
# Bin interest rate
# To put the loan amount in bins
bins2 = pd.cut(df['int_rate'], 6)


 # interest rate

plt.figure(figsize=(20,10))
ax=sns.countplot(y=bins2)
sns.set(font_scale=1.5)
ax.set_title('lnterest rate' , fontsize = 20)
plt.xlabel('Number of customers  ', fontsize=15)
plt.ylabel('Range of interest rate  ', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```

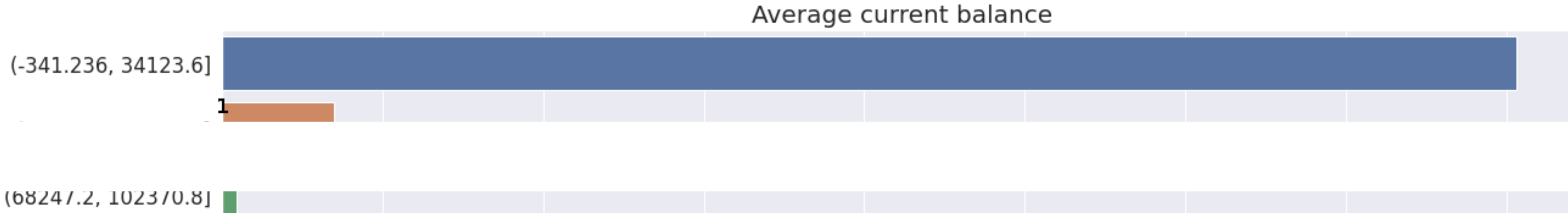



```
# interest rate and loan status

plt.figure(figsize=(20,10))
sns.set(font_scale=1.5)
ax=sns.countplot(x=bins2,  hue='loan_status', data=df)
ax.set_title('Status of loans at different ranges of interest rates ' , fontsize = 20)
plt.xlabel('Range of interest rate  ', fontsize=15)
plt.ylabel('Number of customers ', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```




```
# Bin average
# To put the average current balance in bins
bins3 = pd.cut(df['avg_cur_bal'], 10)


 # average balance

plt.figure(figsize=(20,10))
ax=sns.countplot(y=bins3)
sns.set(font_scale=1.5)
```

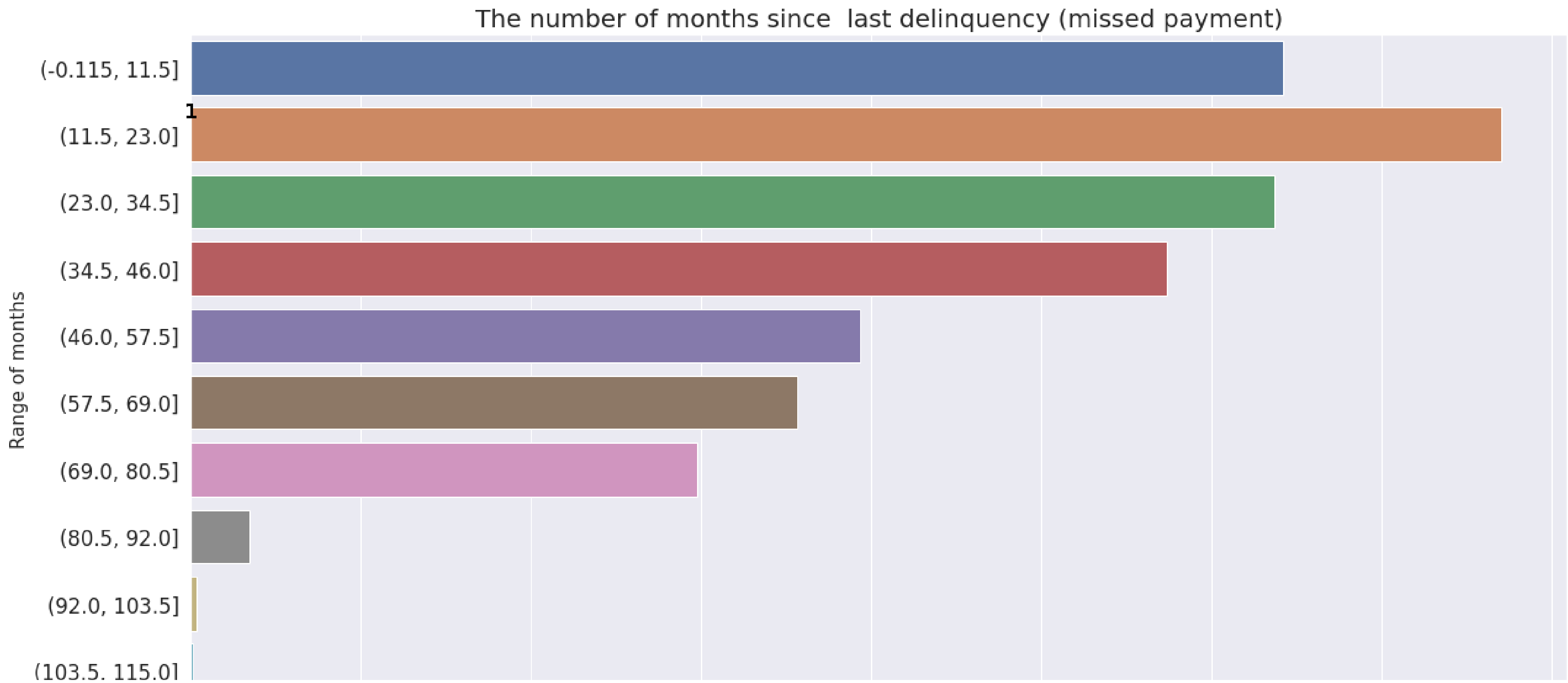
```
ax.set_title('Average current balance' , fontsize = 20)
plt.xlabel('Number of customers  ', fontsize=15)
plt.ylabel('Range of average balance ', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```



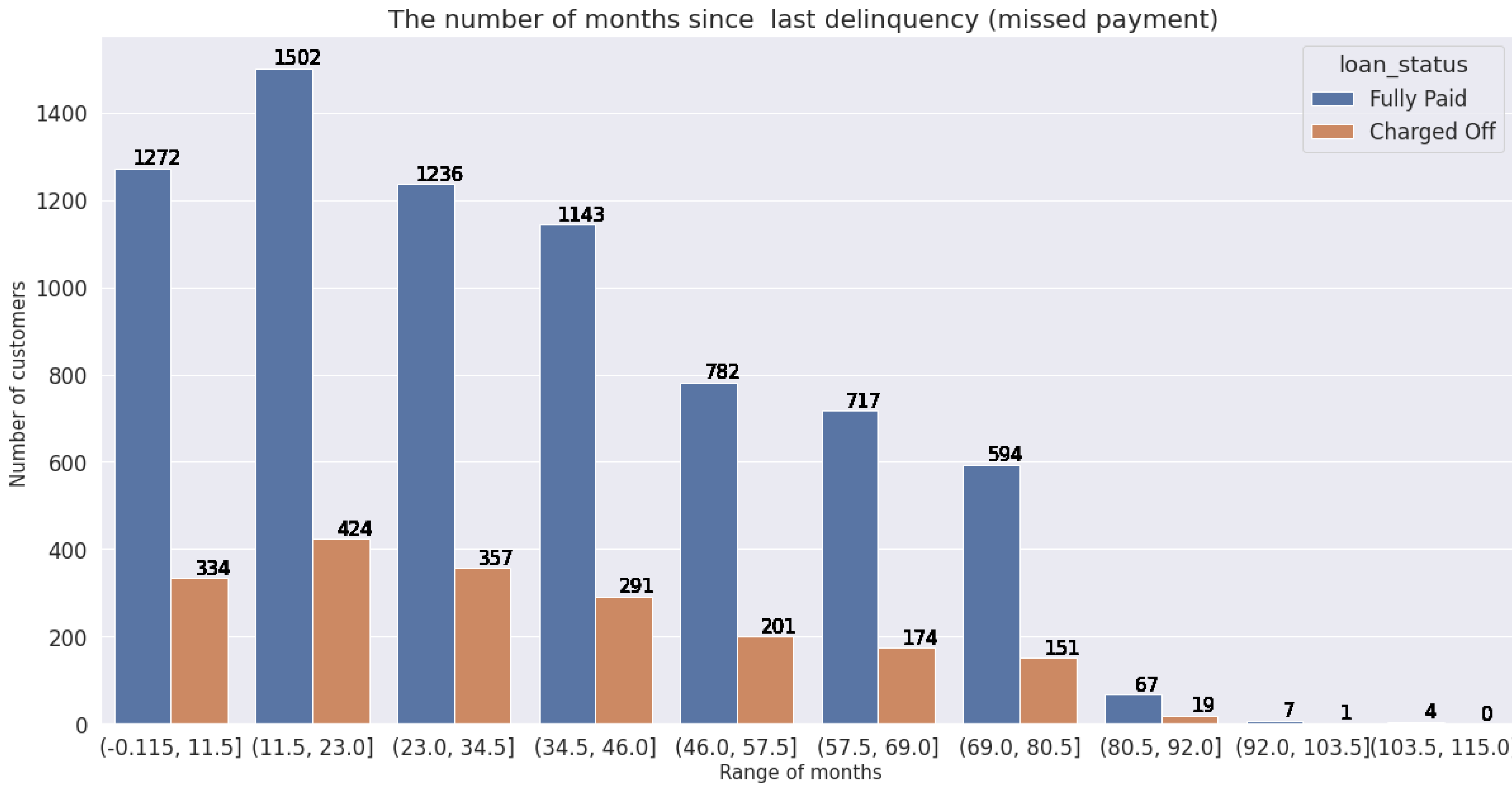

```
# Bin current balance
# To put the average current balance in bins
bins4 = pd.cut(df['mths_since_last_delinq'], 10)
```



```
 # average balance

plt.figure(figsize=(20,10))
ax=sns.countplot(y=bins4)
sns.set(font_scale=1.5)
ax.set_title('The number of months since  last delinquency (missed payment)' , fontsize = 20)
plt.xlabel('Number of customers  ', fontsize=15)
plt.ylabel('Range of months', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```




```
plt.figure(figsize=(20,10))
sns.set(font_scale=1.5)
ax=sns.countplot(x=bins4,  hue='loan_status', data=df)
ax.set_title('The number of months since  last delinquency (missed payment) ' , fontsize = 20)
plt.xlabel('Range of months  ', fontsize=15)
plt.ylabel('Number of customers ', fontsize=15)
plt.xticks(rotation='horizontal')
for p in ax.patches:
    for p in ax.patches:
        ax.annotate(format(p.get_height(), '.0f'),
                    (p.get_x()+0.3, p.get_height()), ha='center', va='bottom', color='black', size=15)
```




```
df.nunique()
```

```
id                    18324
addr_state               51
annual_inc             2434
emp_length               11
```

```
emp_title                 10040
home_ownership                5
installment               10246
loan_amnt                  1111
purpose                      14
term                          2
int_rate                    465
avg_cur_bal               12635
inq_last_12m                 25
max_bal_bc                 6440
mo_sin_old_il_acct          355
mo_sin_old_rev_tl_op        564
mo_sin_rcnt_rev_tl_op       150
mo_sin_rcnt_tl              100
mort_acc                     23
mths_since_last_delinq       98
num_bc_tl                    43
num_il_tl                    64
num_op_rev_tl                44
num_tl_90g_dpd_24m           11
num_tl_op_past_12m           20
open_acc                     49
percent_bc_gt_75            111
pub_rec_bankruptcies          7
total_acc                    93
total_bal_ex_mort         16450
loan_status                   2
dtype: int64
```

```
df.describe()
```

	id	annual_inc	emp_length	installment	loan_amnt	term	int_rate	avg_cur_bal	in
count	1.832400e+04	1.832400e+04	17150.000000	18324.000000	18324.000000	18324.000000	18324.000000	17758.000000	9
mean	6.832645e+07	8.017611e+04	6.073178	467.543006	15522.661537	42.815979	13.850700	13466.600011	
std	4.245703e+07	6.487345e+04	3.639694	278.099801	9349.294243	10.822769	4.822253	16550.730832	
min	3.009180e+05	3.000000e+03	0.500000	30.650000	1000.000000	36.000000	5.310000	0.000000	
50%	6.838023e+07	6.500000e+04	6.000000	397.480000	14000.000000	36.000000	13.330000	7137.000000	

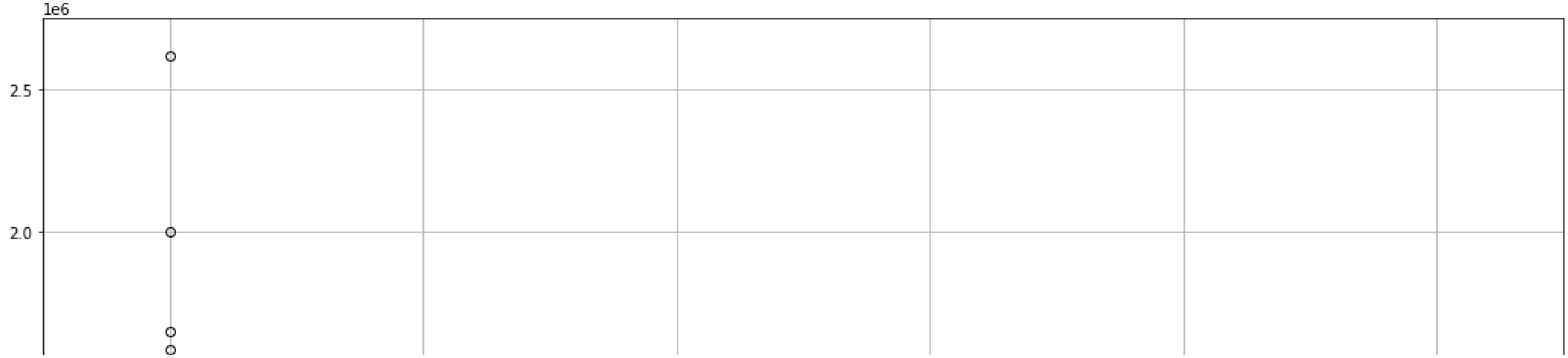
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18324 entries, 0 to 18323
Data columns (total 31 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     18324 non-null  int64  
 1   addr_state             18324 non-null  object 
 2   annual_inc             18324 non-null  float64
 3   emp_length             17150 non-null  float64
 4   emp_title              17042 non-null  object 
 5   home_ownership         18324 non-null  object 
 6   installment            18324 non-null  float64
 7   loan_amnt              18324 non-null  int64  
 8   purpose                18324 non-null  object 
 9   term                   18324 non-null  int64  
 10  int_rate               18324 non-null  float64
 11  avg_cur_bal            17758 non-null  float64
 12  inq_last_12m           9395 non-null   float64
 13  max_bal_bc             9395 non-null   float64
 14  mo_sin_old_il_acct     17192 non-null  float64
 15  mo_sin_old_rev_tl_op   17760 non-null  float64
 16  mo_sin_rcnt_rev_tl_op  17760 non-null  float64
 17  mo_sin_rcnt_tl         17760 non-null  float64
 18  mort_acc               17926 non-null  float64
 19  mths_since_last_delinq 9276 non-null   float64
 20  num_bc_tl              17760 non-null  float64
 21  num_il_tl              17760 non-null  float64
```

```
 22  num_op_rev_tl           17760 non-null  float64
 23  num_tl_90g_dpd_24m      17760 non-null  float64
 24  num_tl_op_past_12m      17760 non-null  float64
 25  open_acc                18324 non-null  int64
 26  percent_bc_gt_75        17714 non-null  float64
 27  pub_rec_bankruptcies    18324 non-null  int64
 28  total_acc               18324 non-null  int64
 29  total_bal_ex_mort       17926 non-null  float64
 30  loan_status             18324 non-null  object
dtypes: float64(20), int64(6), object(5)
memory usage: 4.3+ MB
```

```
plt.figure(figsize=(18,10))
df.boxplot(column=['annual_inc','emp_length','installment','term','avg_cur_bal','max_bal_bc'], return_type='axes')
```

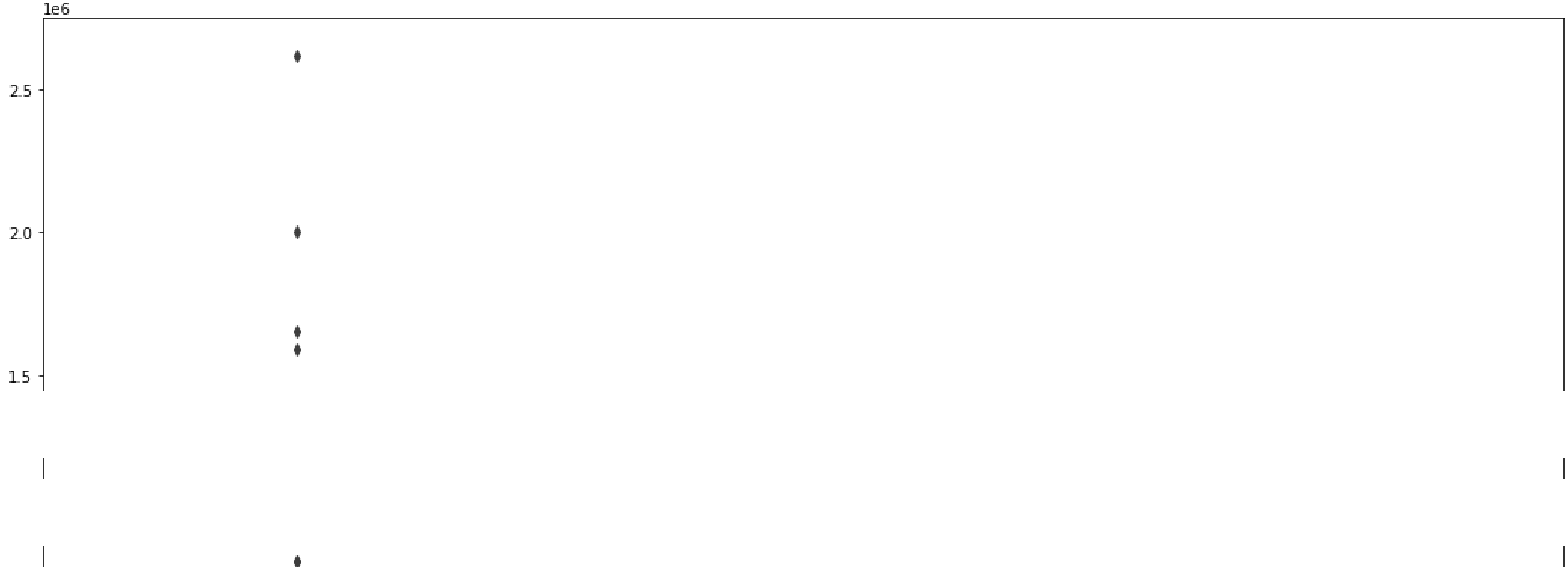
```
<Axes: >
```




```
# now our data has no missing values, we can ahead to scale our data and train our model.
# For this we will use KNN as the first model and Neural Networks for the second
```

```
plt.figure(figsize=(18,10))
sns.boxplot(data=df[['annual_inc','avg_cur_bal','max_bal_bc']])
```

```
<Axes: >
```




```
#ML
data.describe()
```

```
# To drop the 'Id' column
data = data.drop('id', axis=1)
```

```
data.head()
```

	addr_state	annual_inc	emp_length	emp_title	home_ownership	installment	loan_amnt	purpose	term	int
0	CA	72000.0	3.0	CA. Dept. Of Corrections	MORTGAGE	395.66	12000	debt_consolidation	36	
1	TX	97500.0	1.0	Curriculum & Implementation Manager	RENT	966.47	35000	debt_consolidation	60	
2	NY	120000.0	1.0	Senior manager	RENT	806.57	25000	credit_card	36	
3	CA	130000.0	10.0	Border Patrol Agent	RENT	846.17	25225	debt_consolidation	36	
4	TX	58296.0	10.0	Account Manager	MORTGAGE	41.79	1200	other	36	

5 rows × 30 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18324 entries, 0 to 18323
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   addr_state            18324 non-null  object
 1   annual_inc            18324 non-null  float64
```

```
 2   emp_length             17150 non-null  float64
 3   emp_title              17042 non-null  object
 4   home_ownership         18324 non-null  object
 5   installment            18324 non-null  float64
 6   loan_amnt              18324 non-null  int64
 7   purpose                18324 non-null  object
 8   term                   18324 non-null  int64
 9   int_rate               18324 non-null  float64
 10  avg_cur_bal            17758 non-null  float64
 11  inq_last_12m           9395 non-null   float64
 12  max_bal_bc             9395 non-null   float64
 13  mo_sin_old_il_acct     17192 non-null  float64
 14  mo_sin_old_rev_tl_op   17760 non-null  float64
 15  mo_sin_rcnt_rev_tl_op  17760 non-null  float64
 16  mo_sin_rcnt_tl         17760 non-null  float64
 17  mort_acc               17926 non-null  float64
 18  mths_since_last_delinq  9276 non-null   float64
 19  num_bc_tl              17760 non-null  float64
 20  num_il_tl              17760 non-null  float64
 21  num_op_rev_tl          17760 non-null  float64
 22  num_tl_90g_dpd_24m     17760 non-null  float64
 23  num_tl_op_past_12m     17760 non-null  float64
 24  open_acc               18324 non-null  int64
 25  percent_bc_gt_75       17714 non-null  float64
 26  pub_rec_bankruptcies   18324 non-null  int64
 27  total_acc              18324 non-null  int64
 28  total_bal_ex_mort      17926 non-null  float64
 29  loan_status            18324 non-null  object
dtypes: float64(20), int64(5), object(5)
memory usage: 4.2+ MB
```

```
data.isna().sum()
```

```
addr_state                 0
annual_inc                 0
emp_length              1174
emp_title               1282
home_ownership             0
installment                0
loan_amnt                  0
purpose                    0
```

```
term                          0
int_rate                      0
avg_cur_bal                 566
inq_last_12m               8929
max_bal_bc                 8929
mo_sin_old_il_acct         1132
mo_sin_old_rev_tl_op        564
mo_sin_rcnt_rev_tl_op       564
mo_sin_rcnt_tl              564
mort_acc                    398
mths_since_last_delinq     9048
num_bc_tl                   564
num_il_tl                   564
num_op_rev_tl               564
num_tl_90g_dpd_24m          564
num_tl_op_past_12m          564
open_acc                      0
percent_bc_gt_75            610
pub_rec_bankruptcies          0
total_acc                     0
total_bal_ex_mort           398
loan_status                   0
dtype: int64
```

```
df2=data
```

```
# Drop missing values only in the 'emp_title' column
df2.dropna(subset=['emp_title'], inplace=True)
```

```
df2.isna().sum()
```

```
addr_state                    0
annual_inc                    0
```

```
emp_length                  5
emp_title                   0
home_ownership              0
installment                 0
loan_amnt                   0
purpose                     0
term                        0
int_rate                    0
avg_cur_bal               528
inq_last_12m             8258
max_bal_bc               8258
mo_sin_old_il_acct        989
mo_sin_old_rev_tl_op      527
mo_sin_rcnt_rev_tl_op     527
mo_sin_rcnt_tl            527
mort_acc                  369
mths_since_last_delinq   8335
num_bc_tl                 527
num_il_tl                 527
num_op_rev_tl             527
num_tl_90g_dpd_24m        527
num_tl_op_past_12m        527
open_acc                    0
percent_bc_gt_75          552
pub_rec_bankruptcies        0
total_acc                   0
total_bal_ex_mort         369
loan_status                 0
dtype: int64
```

```
df2.head()
```

	addr_state	annual_inc	emp_length	emp_title	home_ownership	installment	loan_amnt	purpose	term	int
0	CA	72000.0	3.0	CA. Dept. Of Corrections	MORTGAGE	395.66	12000	debt_consolidation	36	
1	TX	97500.0	1.0	Curriculum & Implementation Manager	RENT	966.47	35000	debt_consolidation	60	
2	NY	120000.0	1.0	Senior manager	RENT	806.57	25000	credit_card	36	
3	CA	130000.0	10.0	Border Patrol Agent	RENT	846.17	25225	debt_consolidation	36	

```
# Identify columns with missing values
missing_cols = df2.columns[df2.isnull().any()]
```

```
missing_cols
```

```
Index(['emp_length', 'avg_cur_bal', 'inq_last_12m', 'max_bal_bc',
       'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op',
       'mo_sin_rcnt_tl', 'mort_acc', 'mths_since_last_delinq', 'num_bc_tl',
       'num_il_tl', 'num_op_rev_tl', 'num_tl_90g_dpd_24m',
       'num_tl_op_past_12m', 'percent_bc_gt_75', 'total_bal_ex_mort'],
      dtype='object')
```

```
li=list(missing_cols)
```

```
li
```

```
['emp_length',
 'avg_cur_bal',
```

```
       'inq_last_12m',
       'max_bal_bc',
       'mo_sin_old_il_acct',
       'mo_sin_old_rev_tl_op',
       'mo_sin_rcnt_rev_tl_op',
       'mo_sin_rcnt_tl',
       'mort_acc',
       'mths_since_last_delinq',
       'num_bc_tl',
       'num_il_tl',
       'num_op_rev_tl',
       'num_tl_90g_dpd_24m',
       'num_tl_op_past_12m',
       'percent_bc_gt_75',
       'total_bal_ex_mort']
```

```
# replace missing values using KNN imputation
# instantitating KNN imputer
imputer = KNNImputer(n_neighbors=5)
```

```
df2[li] = imputer.fit_transform(df2[li])
```

```
df2.isna().sum()
```

```
     addr_state                0
     annual_inc                0
     emp_length                0
     emp_title                 0
     home_ownership            0
     installment               0
     loan_amnt                 0
     purpose                   0
     term                      0
     int_rate                  0
     avg_cur_bal               0
     inq_last_12m              0
```

```
max_bal_bc                0
mo_sin_old_il_acct        0
mo_sin_old_rev_tl_op      0
mo_sin_rcnt_rev_tl_op     0
mo_sin_rcnt_tl            0
mort_acc                  0
mths_since_last_delinq    0
num_bc_tl                 0
num_il_tl                 0
num_op_rev_tl             0
num_tl_90g_dpd_24m        0
num_tl_op_past_12m        0
open_acc                  0
percent_bc_gt_75          0
pub_rec_bankruptcies      0
total_acc                 0
total_bal_ex_mort         0
loan_status               0
dtype: int64
```

```
df3=df2
```

```
df3.shape
```

```
(17042, 30)
```

```
df3['loan_status'].value_counts(normalize=True)
```

```
Fully Paid     0.789872
Charged Off    0.210128
Name: loan_status, dtype: float64
```

```
# handle class imbalance using oversampling
X = df3.drop('loan_status', axis=1)
y = df3['loan_status']
```

```
sampler = RandomOverSampler()
X_resampled, y_resampled = sampler.fit_resample(X, y)
```

```
X_resampled.shape
```

```
(26922, 29)
```

```
y_resampled.shape
```

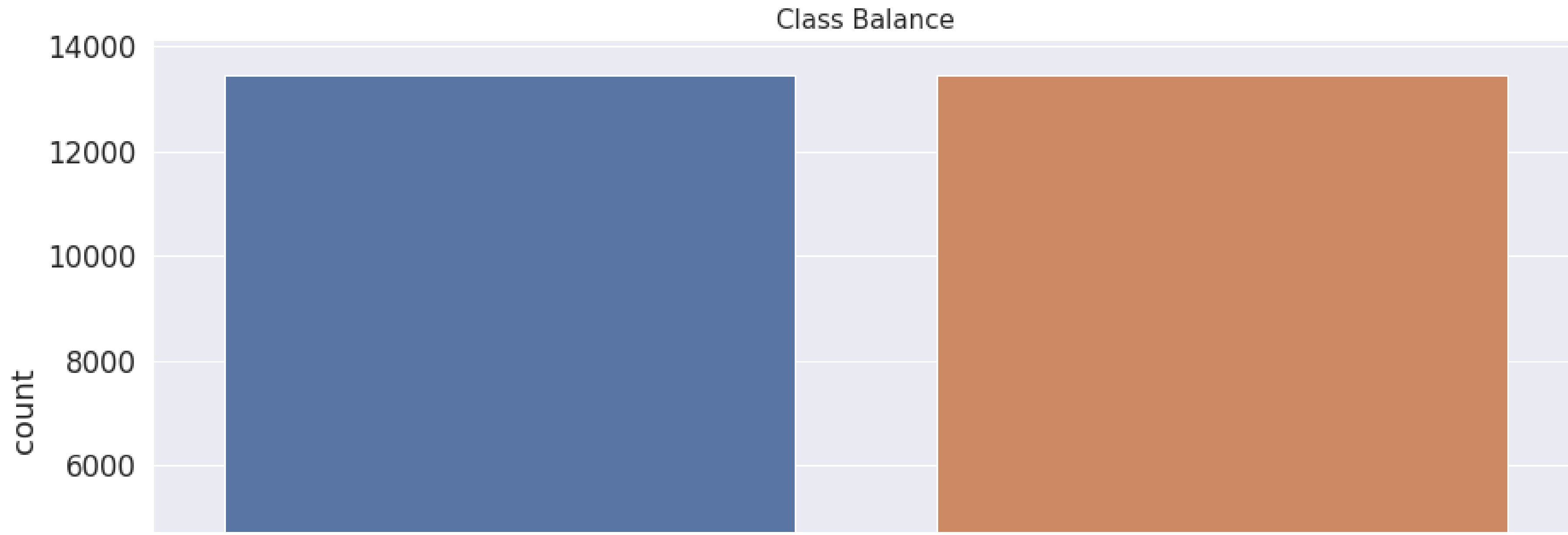
```
(26922,)
```

```
y_resampled.value_counts(normalize=True)
```

```
Fully Paid     0.5
Charged Off    0.5
Name: loan_status, dtype: float64
```

```
plt.figure(figsize=(15,8))
plt.title('Class Balance', fontsize=15)
sns.countplot(x= y_resampled)
```

```
<Axes: title={'center': 'Class Balance'}, xlabel='loan_status', ylabel='count'>
```




```
#Encoding
# encode categorical variables
le = LabelEncoder()
for col in X_resampled.select_dtypes(include='object'):
    X_resampled[col] = le.fit_transform(X_resampled[col])
```


```
X_resampled.head()
```

	addr_state	annual_inc	emp_length	emp_title	home_ownership	installment	loan_amnt	purpose	term	int_rate	...
0	4	72000.0	3.0	1063	1	395.66	12000	2	36	11.49	...
1	43	97500.0	1.0	2003	4	966.47	35000	2	60	21.99	

```
y_resampled.tail()
```

```
26917    Charged Off
26918    Charged Off
26919    Charged Off
26920    Charged Off
26921    Charged Off
Name: loan_status, dtype: object
```

```
# Replace categorical values in the outcome column
y_resampled = y_resampled.replace({'Fully Paid': 1, 'Charged Off': 0})

# Check the new values in the outcome column
print(y_resampled.unique())
```

```
[1 0]
```

```
y_resampled.tail()
```

```
26917    0
26918    0
26919    0
26920    0
26921    0
Name: loan_status, dtype: int64
```

```
X = X_resampled
```

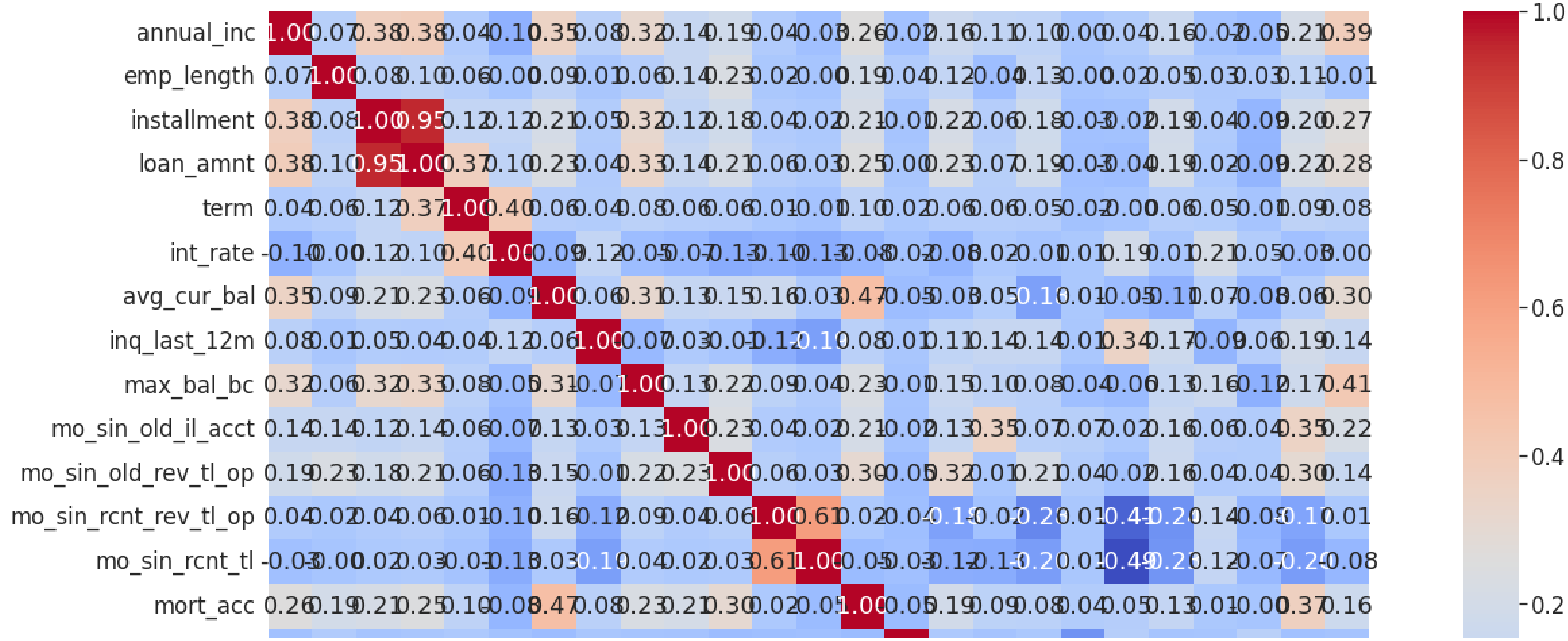
```
y = y_resampled
```

```
corr = df3.corr()
corr
```

	annual_inc	emp_length	installment	loan_amnt	term	int_rate	avg_cur_bal	inq_last_12m	ma
annual_inc	1.000000	0.068835	0.377717	0.384595	0.038419	-0.104523	0.352539	0.080676	
emp_length	0.068835	1.000000	0.080613	0.096398	0.062969	-0.002952	0.093442	0.005957	
installment	0.377717	0.080613	1.000000	0.948106	0.120482	0.120168	0.209443	0.045250	
loan_amnt	0.384595	0.096398	0.948106	1.000000	0.372737	0.104367	0.232020	0.039641	
term	0.038419	0.062969	0.120482	0.372737	1.000000	0.401595	0.055051	0.041989	
int_rate	-0.104523	-0.002952	0.120168	0.104367	0.401595	1.000000	-0.088488	0.119741	
avg_cur_bal	0.352539	0.093442	0.209443	0.232020	0.055051	-0.088488	1.000000	0.059283	

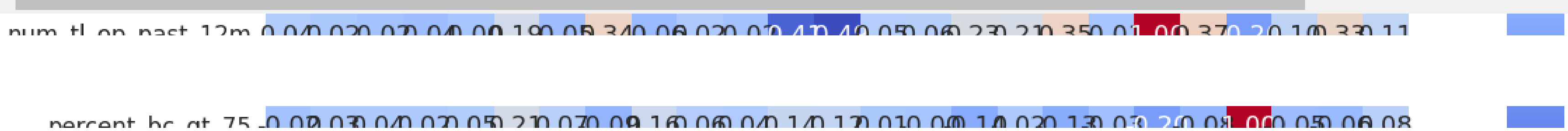
```
#to get variables that are highly correlated
plt.figure(figsize=(25,15))

# Calculate the correlation matrix
corr = df3.corr()

# Create a heatmap to visualize the correlation matrix
sns.heatmap(corr, cmap='coolwarm', annot=True, fmt='.2f', square=True)

# Set a threshold for correlation coefficient
threshold = 0.5

# Find the highly correlated features
highly_correlated = []
for i in range(len(corr.columns)):
    for j in range(i):
        if abs(corr.iloc[i, j]) > threshold:
            colname = corr.columns[i]
            highly_correlated.append(colname)

# Drop the highly correlated features
#df.drop(highly_correlated, axis=1, inplace=True)
print(highly_correlated)
```

```
['loan_amnt', 'mo_sin_rcnt_tl', 'num_op_rev_tl', 'open_acc', 'open_acc', 'total_acc', 'total_acc', 'total_acc', 'total_
```




```
print(highly_correlated)
```


```
['loan_amnt', 'mo_sin_rcnt_tl', 'num_op_rev_tl', 'open_acc', 'open_acc', 'total_acc', 'total_acc', 'total_acc', 'total_
```

```
# Instantiate SelectKBest with f_classif as the scoring function
selector = SelectKBest(score_func=f_classif, k=10)

# Fit the selector to the data
selector.fit(X_resampled, y_resampled)

# Get the indices of the selected features
selected_features_indices = selector.get_support(indices=True)
```

```
# Get the names of the selected features
selected_features_names = X_resampled.columns[selected_features_indices]

# Print the names of the selected features
print(selected_features_names)
```

```
Index(['annual_inc', 'home_ownership', 'loan_amnt', 'term', 'int_rate',
       'avg_cur_bal', 'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op',
       'mort_acc', 'num_tl_op_past_12m'],
      dtype='object')
```

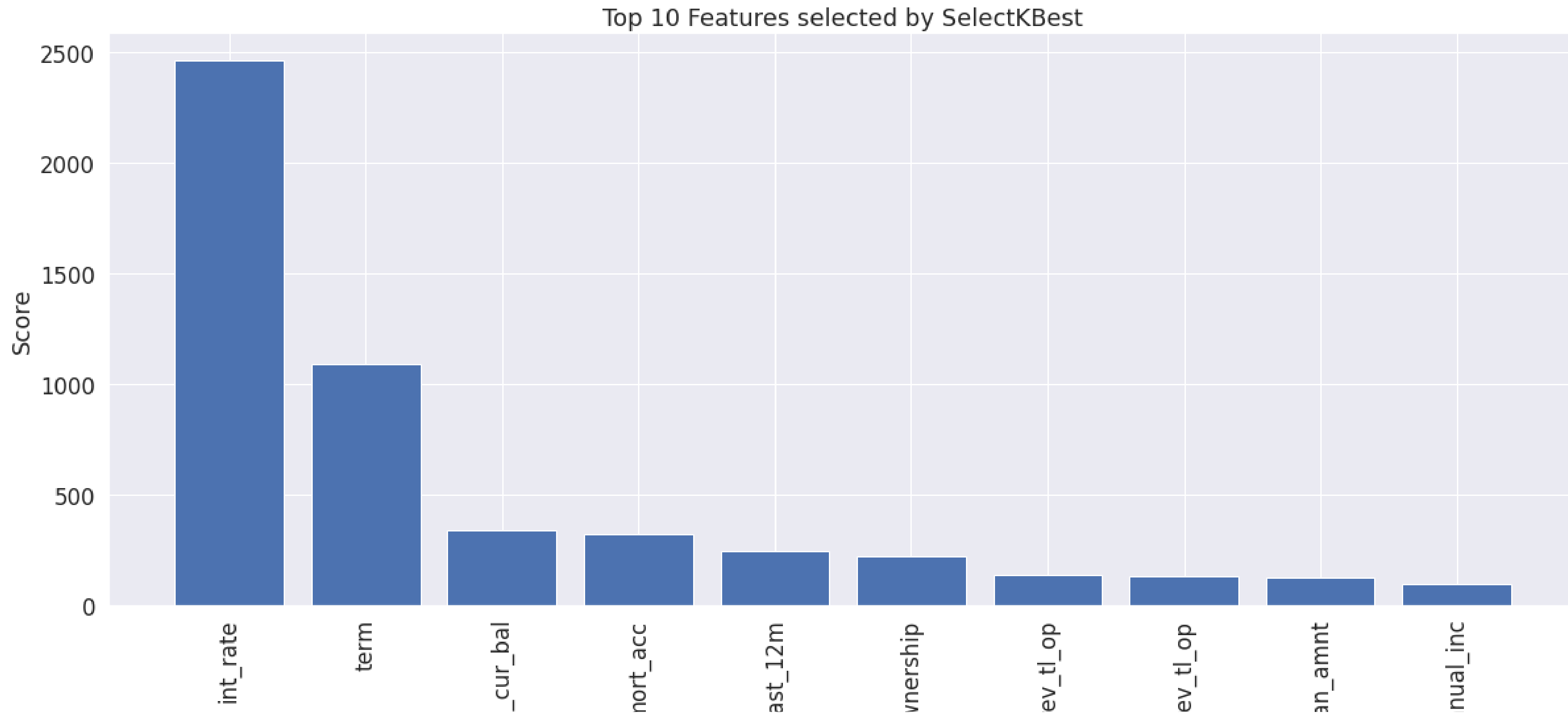
```
# Display the scores of the top 5 features
scores = selector.scores_
top_k_scores = sorted(scores, reverse=True)[:10]
top_k_indices = np.argsort(scores)[::-1][:10]

print("Top 5 feature scores:")
for i in range(len(top_k_scores)):
    print("Feature {}: Score = {:.2f}".format(top_k_indices[i], top_k_scores[i]))
```

```
Top 5 feature scores:
Feature 9: Score = 2463.85
Feature 8: Score = 1089.43
Feature 10: Score = 342.67
Feature 17: Score = 324.50
Feature 23: Score = 247.17
Feature 4: Score = 223.77
Feature 14: Score = 138.89
Feature 15: Score = 132.45
Feature 6: Score = 126.70
Feature 1: Score = 95.74
```

```
feature_names = df3.drop('loan_status', axis=1).columns
```

```python
# Get the names and scores of the top 10 features
feature_names = df3.drop('loan_status', axis=1).columns
top_scores = selector.scores_.argsort()[-10:][::-1]
top_features = feature_names[top_scores]

# Print the names and scores of the top 10 features
for i, feature in enumerate(top_features):
    print("{}. {} ({:.2f})".format(i+1, feature, selector.scores_[top_scores][i]))

# Create a bar plot of the top 10 features and their scores
plt.figure(figsize=(20,8))
sns.set(font_scale=1.5)
plt.bar(range(len(top_scores)), selector.scores_[top_scores])
plt.xticks(range(len(top_scores)), top_features, rotation='vertical')
plt.xlabel("Feature")
plt.ylabel("Score")
plt.title("Top 10 Features selected by SelectKBest")
plt.show()
```

```
1. int_rate (2463.85)
2. term (1089.43)
3. avg_cur_bal (342.67)
4. mort_acc (324.50)
5. num_tl_op_past_12m (247.17)
6. home_ownership (223.77)
7. mo_sin_old_rev_tl_op (138.89)
8. mo_sin_rcnt_rev_tl_op (132.45)
9. loan_amnt (126.70)
10. annual_inc (95.74)
```




```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17042 entries, 0 to 18323
Data columns (total 30 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
```

```
 0   addr_state              17042 non-null  object
 1   annual_inc              17042 non-null  float64
 2   emp_length              17042 non-null  float64
 3   emp_title               17042 non-null  object
 4   home_ownership          17042 non-null  object
 5   installment             17042 non-null  float64
 6   loan_amnt               17042 non-null  int64
 7   purpose                 17042 non-null  object
 8   term                    17042 non-null  int64
 9   int_rate                17042 non-null  float64
 10  avg_cur_bal             17042 non-null  float64
 11  inq_last_12m            17042 non-null  float64
 12  max_bal_bc              17042 non-null  float64
 13  mo_sin_old_il_acct      17042 non-null  float64
 14  mo_sin_old_rev_tl_op    17042 non-null  float64
 15  mo_sin_rcnt_rev_tl_op   17042 non-null  float64
 16  mo_sin_rcnt_tl          17042 non-null  float64
 17  mort_acc                17042 non-null  float64
 18  mths_since_last_delinq  17042 non-null  float64
 19  num_bc_tl               17042 non-null  float64
 20  num_il_tl               17042 non-null  float64
 21  num_op_rev_tl           17042 non-null  float64
 22  num_tl_90g_dpd_24m      17042 non-null  float64
 23  num_tl_op_past_12m      17042 non-null  float64
 24  open_acc                17042 non-null  int64
 25  percent_bc_gt_75        17042 non-null  float64
 26  pub_rec_bankruptcies    17042 non-null  int64
 27  total_acc               17042 non-null  int64
 28  total_bal_ex_mort       17042 non-null  float64
 29  loan_status             17042 non-null  object
dtypes: float64(20), int64(5), object(5)
memory usage: 4.0+ MB
```

```
# select the top K features using f_classic
kbest = SelectKBest(score_func=f_classif, k=10)
X_resampled = kbest.fit_transform(X_resampled, y_resampled)

# split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3,random_state = 40)
```

```
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform (X_test)
```

```
# train a decision tree classifier on the data
clf = DecisionTreeClassifier()

#clf = RandomForestClassifier()

clf.fit(X_train_sc, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test_sc)
print(classification_report(y_test, y_pred))
```

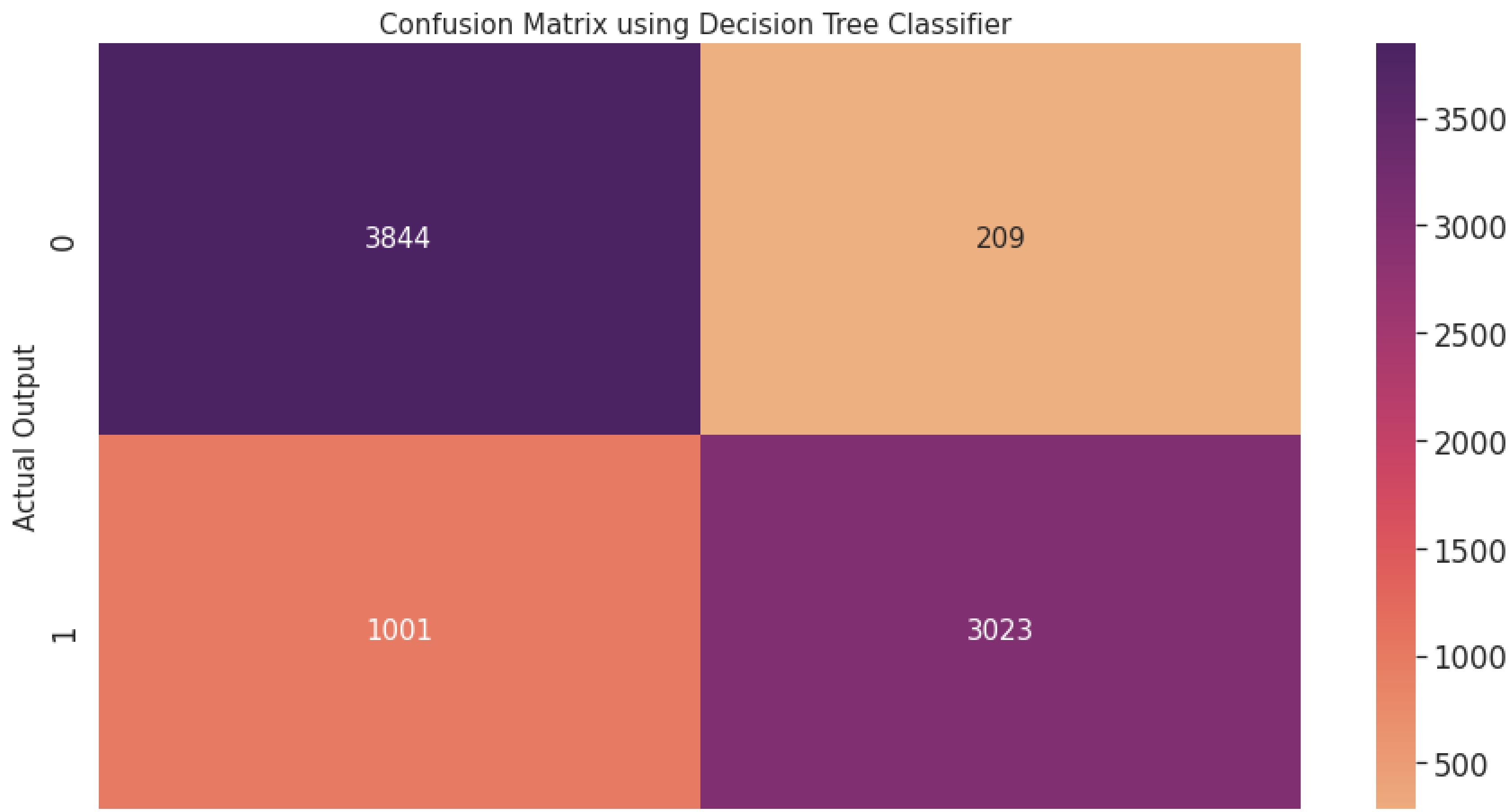
```
              precision    recall  f1-score   support

           0       0.79      0.95      0.86      4053
           1       0.94      0.75      0.83      4024

    accuracy                           0.85      8077
   macro avg       0.86      0.85      0.85      8077
weighted avg       0.86      0.85      0.85      8077
```

```
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
```

```
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('--------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.85


The confusion matrix is:
[[3844  209]
 [1001 3023]]


--------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.79      0.95      0.86      4053
           1       0.94      0.75      0.83      4024

    accuracy                           0.85      8077
   macro avg       0.86      0.85      0.85      8077
weighted avg       0.86      0.85      0.85      8077
```

```
plt.figure(figsize=(15,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 15}, annot= True, fmt = 'd')
plt.title('Confusion Matrix using Decision Tree Classifier ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```




```
#Tuning decision tree
#from sklearn.tree import DecisionTreeClassifier
#from sklearn.model_selection import GridSearchCV


# Define the Decision Tree classifier
dt = DecisionTreeClassifier()

# Define the hyperparameters to tune
param_grid = {'max_depth': [2, 4, 6, 8, 10],
              'min_samples_split': [2, 4, 6, 8, 10],
              'min_samples_leaf': [1, 2, 3, 4, 5]}

# Perform hyperparameter tuning using GridSearchCV
```

```
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, verbose=0)
grid_search.fit(X, y)

# Print the results
print("Best accuracy score: {:.2f}".format(grid_search.best_score_))
print("Best parameters: {}".format(grid_search.best_params_))
```

```
Best accuracy score: 0.70
Best parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 4}
```

```
#Using RandomForest

# train a decision tree classifier on the data
#clf = DecisionTreeClassifier()

clf = RandomForestClassifier()

clf.fit(X_train_sc, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test_sc)
print(classification_report(y_test, y_pred))
```

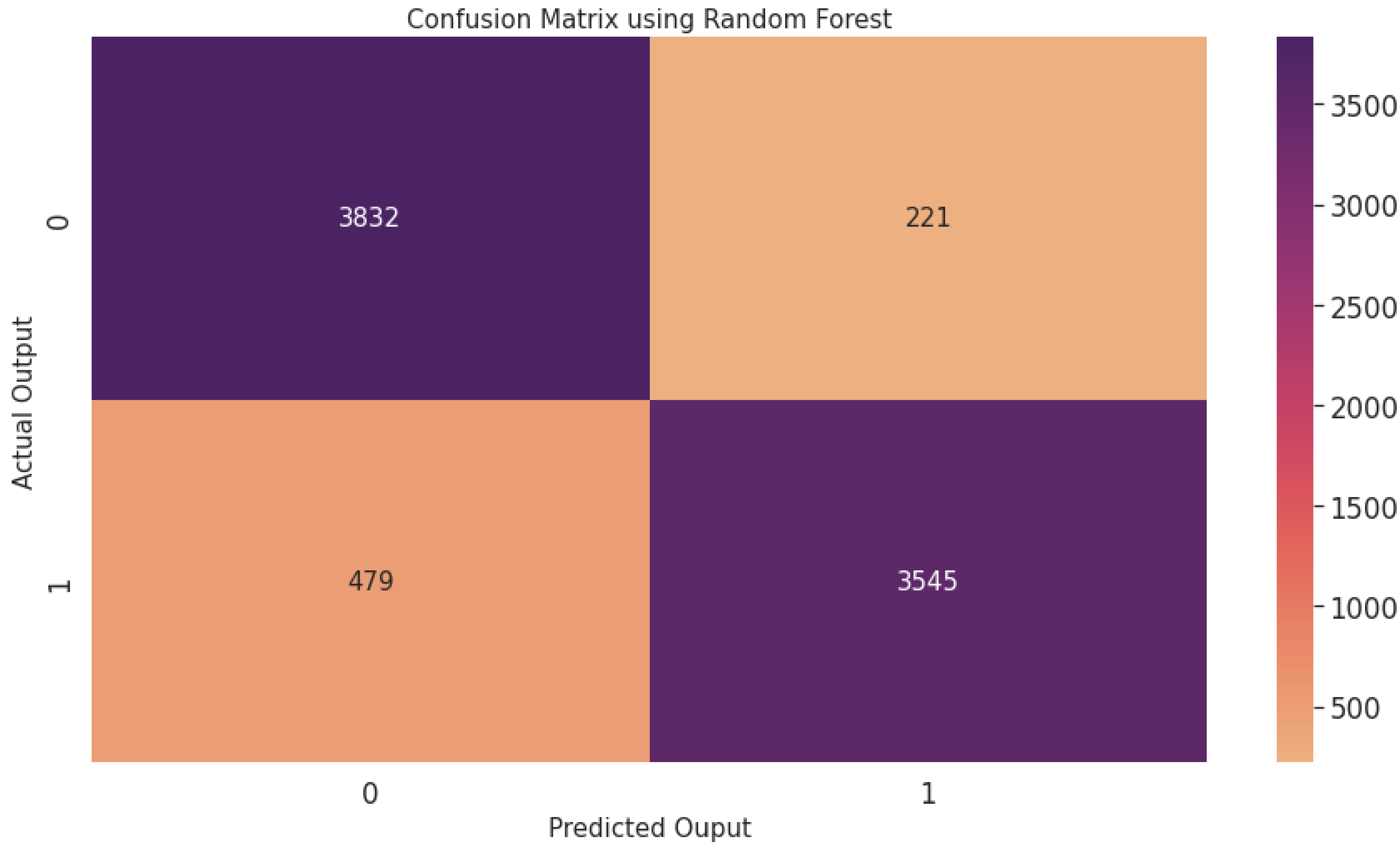
```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      4053
           1       0.94      0.88      0.91      4024

    accuracy                           0.91      8077
   macro avg       0.92      0.91      0.91      8077
```

```
weighted avg       0.92      0.91      0.91      8077
```

```
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('--------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.91


The confusion matrix is:
[[3832  221]
 [ 479 3545]]


--------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.89      0.95      0.92      4053
           1       0.94      0.88      0.91      4024

    accuracy                           0.91      8077
   macro avg       0.92      0.91      0.91      8077
weighted avg       0.92      0.91      0.91      8077
```

```
plt.figure(figsize=(15,8))
zx = sns. heatmap(conf_matrix, cmap ='flare',annot_kws={"size": 15}, annot= True, fmt = 'd')
plt.title('Confusion Matrix using Random Forest', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```



```
# GRADIENT BOOSTED DECISION TREE

#Using Gradient Boosted Decision Tree
# train a decision tree classifier on the data
#clf = DecisionTreeClassifier()
#clf = RandomForestClassifier()

clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.7, max_depth=16, random_state=42)
clf.fit(X_train_sc, y_train)

# test the classifier on the test set and print the classification report
y_pred = clf.predict(X_test_sc)
print(classification_report(y_test, y_pred))
```

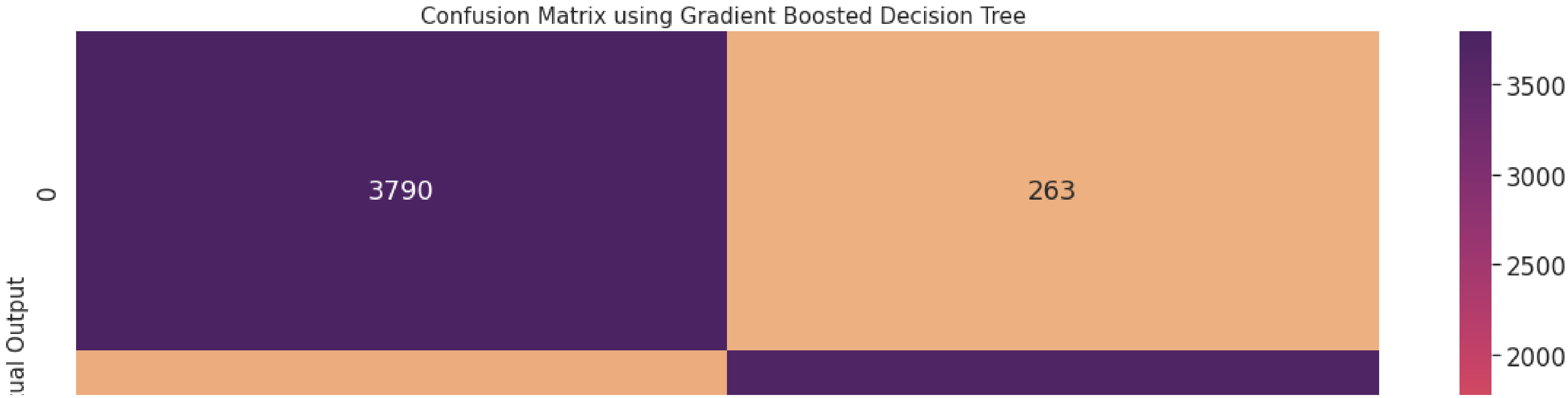
```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      4053
           1       0.93      0.92      0.93      4024

    accuracy                           0.93      8077
   macro avg       0.93      0.93      0.93      8077
weighted avg       0.93      0.93      0.93      8077
```

```
from sklearn import metrics
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('--------------------')
result = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(result)
```

```
Accuracy score:0.93


The confusion matrix is:
```

```
[[3790  263]
 [ 327 3697]]


---------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.92      0.94      0.93      4053
           1       0.93      0.92      0.93      4024

    accuracy                           0.93      8077
   macro avg       0.93      0.93      0.93      8077
weighted avg       0.93      0.93      0.93      8077
```

```
plt.figure(figsize=(20,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 18},annot= True, fmt = 'd')
plt.title('Confusion Matrix using Gradient Boosted Decision Tree ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```




```
# F1-score of 94%

# Identifying top 10 features driving the Gradient Boosted Decision model by allowing the model to select importance feature

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state = 40)



scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform (X_test)




# Step 4: Model Evaluation
from sklearn.metrics import f1_score
```

```
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.transform (X_test)


clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.7, max_depth=16, random_state=42)
clf.fit(X_train_scale, y_train)
y_pred = clf.predict(X_test_scale)
score = f1_score(y_test, y_pred)
print(f"F1-score: {score:.4f}")
```

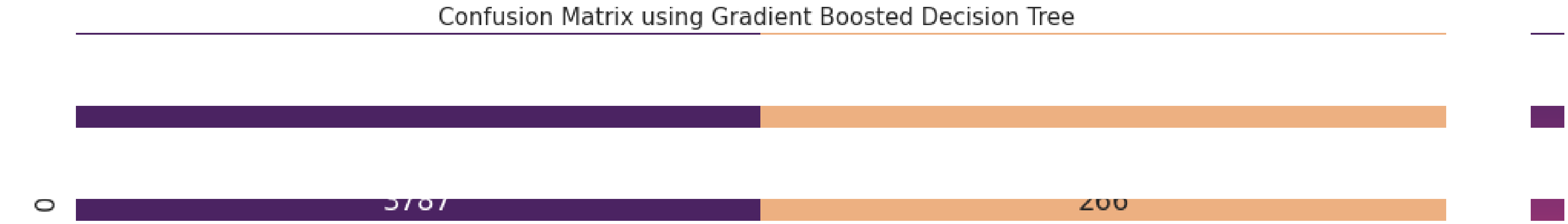
```
F1-score: 0.9350
```

```
from sklearn import metrics
Accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy score:%.2f\n\n'%(Accuracy))
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
print('The confusion matrix is:')
print(conf_matrix,'\n\n')
print('--------------------')
results = metrics.classification_report(y_test, y_pred)
print('Classification Report:\n')
print(results)
```

```
Accuracy score:0.94


The confusion matrix is:
[[3787  266]
 [ 258 3766]]


--------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.94      0.93      0.94      4053
           1       0.93      0.94      0.93      4024
```

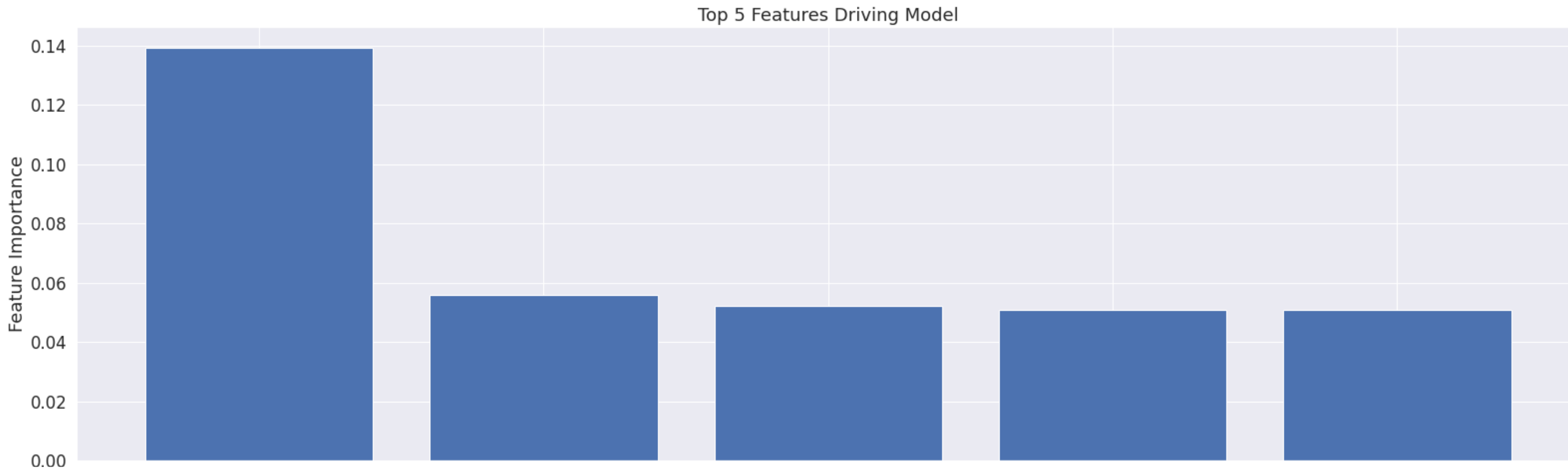
```
    accuracy                           0.94      8077
   macro avg       0.94      0.94      0.94      8077
weighted avg       0.94      0.94      0.94      8077
```

```
plt.figure(figsize=(20,8))
zx = sns. heatmap(conf_matrix, cmap ='flare', annot_kws={"size": 18},annot= True, fmt = 'd')
plt.title('Confusion Matrix using Gradient Boosted Decision Tree ', fontsize= 15)
plt.xlabel('Predicted Ouput', fontsize =15)
plt.ylabel('Actual Output', fontsize =15)
plt.show()
```




```
#Step 5: Feature Importance Analysis
feature_importances = clf.feature_importances_
feature_names = df3.drop('loan_status', axis=1).columns
```

```
# Step 6: Plot Feature Importance Graph


top_features = pd.Series(feature_importances, index=feature_names).sort_values(ascending=False)[:5]
plt.figure(figsize=(27,8))
plt.bar(top_features.index, top_features)
plt.title('Top 5 Features Driving Model')
plt.xlabel('Feature Name')
plt.ylabel('Feature Importance')
plt.show()
```


Top 5 Features Driving Model
Feature Importance
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00