loan-tap-logistic-regression

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report,␣
↪roc_auc_score
from sklearn.metrics import precision_recall_curve, auc, roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
```

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving LoanTapData.csv to LoanTapData.csv
```

[ ]:
```
df = pd.read_csv("LoanTapData.csv")
df.sample(5)
```

[ ]:
```
        loan_amnt        term  int_rate  installment grade sub_grade  \
55376       25000   36 months     15.31       870.44     C        C2
176997      12000   60 months     17.86       303.81     D        D5
242595      20000   60 months     18.99       518.71     E        E1
330244      14125   36 months     18.75       515.99     D        D3
378456       7000   36 months     15.61       244.76     C        C5

                     emp_title emp_length home_ownership  annual_inc  …  \
55376                      CSC   < 1 year       MORTGAGE    120000.0  …
176997    Sales Representative  10+ years           RENT    120000.0  …
242595                 carrier  10+ years       MORTGAGE     65000.0  …
330244      Cox Communications    4 years       MORTGAGE    115000.0  …
378456                 Manager  10+ years       MORTGAGE    109000.0  …

       open_acc pub_rec revol_bal revol_util total_acc  initial_list_status  \
55376        10       0     20359       57.8        27                    f
```

```
176997        9        0      7423       24.1        17                      w
242595       10        0     25305       88.5        20                      w
330244       13        0      4012       57.3        23                      w
378456       11        0      8614       46.8        19                      w

       application_type  mort_acc  pub_rec_bankruptcies  \
55376        INDIVIDUAL       1.0                   0.0
176997       INDIVIDUAL       1.0                   0.0
242595       INDIVIDUAL       3.0                   0.0
330244       INDIVIDUAL       4.0                   0.0
378456       INDIVIDUAL       2.0                   0.0

                                                  address
55376   263 Susan Stream Apt. 499\r\nKimberlytown, NM …
176997                 PSC 8941, Box 5836\r\nAPO AP 00813
242595                         USCGC Cook\r\nFPO AP 70466
330244  15435 Brown Mountains Apt. 016\r\nDanieltown, …
378456  67034 William Islands Apt. 134\r\nSouth Samant…

[5 rows x 27 columns]
```

0.0.1 Data Exploration and Wrangling

[ ]:
```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  int64
 1   term                  396030 non-null  object
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object
 5   sub_grade             396030 non-null  object
 6   emp_title             373103 non-null  object
 7   emp_length            377729 non-null  object
 8   home_ownership        396030 non-null  object
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object
 11  issue_d               396030 non-null  object
 12  loan_status           396030 non-null  object
 13  purpose               396030 non-null  object
 14  title                 394274 non-null  object
 15  dti                   396030 non-null  float64
```

```
 16  earliest_cr_line      396030 non-null  object
 17  open_acc              396030 non-null  int64
 18  pub_rec               396030 non-null  int64
 19  revol_bal             396030 non-null  int64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  int64
 22  initial_list_status   396030 non-null  object
 23  application_type      396030 non-null  object
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object
dtypes: float64(7), int64(5), object(15)
memory usage: 81.6+ MB
```

[ ]:

```
           loan_amnt       int_rate    installment    annual_inc  \
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean    14113.888089      13.639400     431.849698  7.420318e+04
std      8357.441341       4.472157     250.727790  6.163762e+04
min       500.000000       5.320000      16.080000  0.000000e+00
25%      8000.000000      10.490000     250.330000  4.500000e+04
50%     12000.000000      13.330000     375.430000  6.400000e+04
75%     20000.000000      16.490000     567.300000  9.000000e+04
max     40000.000000      30.990000    1533.810000  8.706582e+06

                 dti       open_acc        pub_rec     revol_bal  \
count  396030.000000  396030.000000  396030.000000  3.960300e+05
mean       17.379514      11.311153       0.178191  1.584454e+04
std        18.019092       5.137649       0.530671  2.059184e+04
min         0.000000       0.000000       0.000000  0.000000e+00
25%        11.280000       8.000000       0.000000  6.025000e+03
50%        16.910000      10.000000       0.000000  1.118100e+04
75%        22.980000      14.000000       0.000000  1.962000e+04
max      9999.000000      90.000000      86.000000  1.743266e+06

          revol_util      total_acc       mort_acc  pub_rec_bankruptcies
count  395754.000000  396030.000000  358235.000000         395495.000000
mean       53.791749      25.414744       1.813991              0.121648
std        24.452193      11.886991       2.147930              0.356174
min         0.000000       2.000000       0.000000              0.000000
25%        35.800000      17.000000       0.000000              0.000000
50%        54.800000      24.000000       1.000000              0.000000
75%        72.900000      32.000000       3.000000              0.000000
max       892.300000     151.000000      34.000000              8.000000
```

[ ]:

```python
# No.of Rows and Columns
print("No. of Rows:", df.shape[0])
print("No. of Columns:", df.shape[1])
```

```
No. of Rows: 396030
No. of Columns: 27
```

[ ]:
```
df.isna().sum().sum()
# we can drop the null values, but we are not going to drop it, instead we are
↪going to impute it
```

[ ]:
```
81590
```

[ ]:
```
# Distribution of Target labels
df["loan_status"].value_counts(normalize=True)
```

[ ]:
```
loan_status
Fully Paid     0.803871
Charged Off    0.196129
Name: proportion, dtype: float64
```

- There is a good amount of imabalance in the target outcomes, this implies accuracy metric will not be a good option to evaluate our model.
- First let I will fit a model with the same imbalanced data and check for evaulation metrics then after that I will fit model on the dataset to which I am going to oversample it for reducing the imbalance of the minority class.

[ ]:
```
# correlation Matrix
df.select_dtypes(include="number").corr(method="spearman")
```

[ ]:
```
                      loan_amnt  int_rate  installment  annual_inc       dti  \
loan_amnt              1.000000  0.131432     0.968334    0.488566  0.053118
int_rate               0.131432  1.000000     0.137293   -0.096648  0.172123
installment            0.968334  0.137293     1.000000    0.470464  0.055538
annual_inc             0.488566 -0.096648     0.470464    1.000000 -0.202856
dti                    0.053118  0.172123     0.055538   -0.202856  1.000000
open_acc               0.215244  0.004181     0.207828    0.240468  0.322745
pub_rec               -0.100435  0.072204    -0.093357   -0.046322 -0.042220
revol_bal              0.469646  0.005948     0.460147    0.393232  0.249720
revol_util             0.104708  0.303990     0.131628    0.060172  0.185130
total_acc              0.237115 -0.050880     0.216667    0.334453  0.236925
mort_acc               0.231240 -0.102962     0.201861    0.378772 -0.048033
pub_rec_bankruptcies  -0.108661  0.060848    -0.102922   -0.071770 -0.032790

                      open_acc   pub_rec  revol_bal  revol_util  total_acc  \
loan_amnt             0.215244 -0.100435   0.469646    0.104708   0.237115
int_rate              0.004181  0.072204   0.005948    0.303990  -0.050880
installment           0.207828 -0.093357   0.460147    0.131628   0.216667
annual_inc            0.240468 -0.046322   0.393232    0.060172   0.334453
dti                   0.322745 -0.042220   0.249720    0.185130   0.236925
open_acc              1.000000 -0.019111   0.364325   -0.139233   0.672232
pub_rec              -0.019111  1.000000  -0.209249   -0.095391   0.033317
```

```
revol_bal             0.364325 -0.209249   1.000000    0.419506   0.294369
revol_util           -0.139233 -0.095391   0.419506    1.000000  -0.104718
total_acc             0.672232  0.033317   0.294369   -0.104718   1.000000
mort_acc              0.142336  0.031729   0.239253    0.008440   0.404738
pub_rec_bankruptcies -0.025244  0.862245  -0.205400   -0.091208   0.041582

                      mort_acc  pub_rec_bankruptcies
loan_amnt             0.231240             -0.108661
int_rate             -0.102962              0.060848
installment           0.201861             -0.102922
annual_inc            0.378772             -0.071770
dti                  -0.048033             -0.032790
open_acc              0.142336             -0.025244
pub_rec               0.031729              0.862245
revol_bal             0.239253             -0.205400
revol_util            0.008440             -0.091208
total_acc             0.404738              0.041582
mort_acc              1.000000              0.039623
pub_rec_bankruptcies  0.039623              1.000000
```

[ ]:
```python
df.select_dtypes(include="number").corr(method="pearson")
```

[ ]:
```
                      loan_amnt  int_rate  installment  annual_inc       dti  \
loan_amnt              1.000000  0.168921     0.953929    0.336887  0.016636
int_rate               0.168921  1.000000     0.162758   -0.056771  0.079038
installment            0.953929  0.162758     1.000000    0.330381  0.015786
annual_inc             0.336887 -0.056771     0.330381    1.000000 -0.081685
dti                    0.016636  0.079038     0.015786   -0.081685  1.000000
open_acc               0.198556  0.011649     0.188973    0.136150  0.136181
pub_rec               -0.077779  0.060986    -0.067892   -0.013720 -0.017639
revol_bal              0.328320 -0.011280     0.316455    0.299773  0.063571
revol_util             0.099911  0.293659     0.123915    0.027871  0.088375
total_acc              0.223886 -0.036404     0.202430    0.193023  0.102128
mort_acc               0.222315 -0.082583     0.193694    0.236320 -0.025439
pub_rec_bankruptcies  -0.106539  0.057450    -0.098628   -0.050162 -0.014558

                      open_acc   pub_rec  revol_bal  revol_util  total_acc  \
loan_amnt             0.198556 -0.077779   0.328320    0.099911   0.223886
int_rate              0.011649  0.060986  -0.011280    0.293659  -0.036404
installment           0.188973 -0.067892   0.316455    0.123915   0.202430
annual_inc            0.136150 -0.013720   0.299773    0.027871   0.193023
dti                   0.136181 -0.017639   0.063571    0.088375   0.102128
open_acc              1.000000 -0.018392   0.221192   -0.131420   0.680728
pub_rec              -0.018392  1.000000  -0.101664   -0.075910   0.019723
revol_bal             0.221192 -0.101664   1.000000    0.226346   0.191616
revol_util           -0.131420 -0.075910   0.226346    1.000000  -0.104273
total_acc             0.680728  0.019723   0.191616   -0.104273   1.000000
```

```
mort_acc              0.109205  0.011552   0.194925    0.007514   0.381072
pub_rec_bankruptcies -0.027732  0.699408  -0.124532   -0.086751   0.042035

                      mort_acc  pub_rec_bankruptcies
loan_amnt             0.222315             -0.106539
int_rate             -0.082583              0.057450
installment           0.193694             -0.098628
annual_inc            0.236320             -0.050162
dti                  -0.025439             -0.014558
open_acc              0.109205             -0.027732
pub_rec               0.011552              0.699408
revol_bal             0.194925             -0.124532
revol_util            0.007514             -0.086751
total_acc             0.381072              0.042035
mort_acc              1.000000              0.027239
pub_rec_bankruptcies  0.027239              1.000000
```

- As you can see both spearman and pearson correlation values are almost equal which says that there might be a presence of slight to no non-linearity in the features.
- The installment amount and loan amount are highly correlated which has correlation of over 0.95 so we can remove any one of the 2 features.

[ ]:
```
# Dropping installemnt column
df.drop(columns=["installment"],inplace=True)
```

[ ]:
```
df["term"].unique()
```

[ ]:
```
array([' 36 months', ' 60 months'], dtype=object)
```

[ ]:
```
# Cleaning individual columns
# term
df["term"]=df["term"].str.strip()
map = {"36 months":36,"60 months":60}
df["term"]=df["term"].map(map)
```

[ ]:
```
# grade
df["grade"].value_counts()
```

[ ]:
```
grade
B    116018
C    105987
A     64187
D     63524
E     31488
F     11772
G      3054
Name: count, dtype: int64
```

[ ]:
```
df["emp_length"].value_counts()
map = {"10+ years":10,"2 years":2,"< 1 year":1,"3 years":3,"5 years":5,"6␣
↪years":6,
      "4 years":4,"1 year":1,"7 years":7,"8 years":8,"9 years":9}
df["emp_length"] = df["emp_length"].map(map)
```

[ ]:
```
df["emp_length"].value_counts()
```

[ ]:
```
emp_length
10.0    126041
1.0      57607
2.0      35827
3.0      31665
5.0      26495
4.0      23952
6.0      20841
7.0      20819
8.0      19168
9.0      15314
Name: count, dtype: int64
```

[ ]:
```
df["home_ownership"].value_counts()
```

[ ]:
```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: count, dtype: int64
```

[ ]:
```
df.loc[(df["home_ownership"]=="NONE") | (df["home_ownership"]=="ANY"),␣
↪"home_ownership"] = "OTHER"
```

[ ]:
```
df["home_ownership"].value_counts()
```

[ ]:
```
home_ownership
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: count, dtype: int64
```

[ ]:
```
df["verification_status"].value_counts()
```

```
[ ]: verification_status
Verified           139563
Source Verified    131385
Not Verified       125082
Name: count, dtype: int64
```

```
[ ]: # since the full address is not of use
df["zipcode"]=df["address"].apply(lambda x:x[-5:])
```

```
[ ]: # We extracted zip code from address column, now address is of no use, that is␣
   ↪why we are dropping it
df.drop(columns=["address"],inplace=True)
```

0.0.2 Converting Data Types of columns

```
[ ]: df["grade"] = df["grade"].astype("category")
df["sub_grade"] = df["sub_grade"].astype("category")
df["verification_status"]=df["verification_status"].astype("category")
```

```
[ ]: df["issue_d"] = pd.to_datetime(df["issue_d"], format='%b-%y')
df["earliest_cr_line"] = pd.to_datetime(df["earliest_cr_line"], format='%b-%y')
```

```
[ ]: df["title"].value_counts()
```

```
[ ]: title
Debt consolidation                        152472
Credit card refinancing                    51487
Home improvement                           15264
Other                                      12930
Debt Consolidation                         11608
                                           …
buisness startup                               1
Medical-Lap Band                               1
Debt consolidation to improve my life!         1
one debt only                                  1
Toxic Debt Payoff                              1
Name: count, Length: 48787, dtype: int64
```

```
[ ]: df["purpose"].value_counts()
```

```
[ ]: purpose
debt_consolidation    234507
credit_card            83019
home_improvement       24030
other                  21185
major_purchase          8790
small_business          5701
```

```
car                     4697
medical                 4196
moving                  2854
vacation                2452
house                   2201
wedding                 1812
renewable_energy         329
educational              257
Name: count, dtype: int64
```

[ ]:
```
# As you can see Title and purpose column are giving same information, so we␣
↪can drop one of that
```

[ ]:
```
df.drop(columns=["title"],inplace=True)
```

[ ]:
```
# Initial list status
df["initial_list_status"].value_counts()
df["initial_list_status"].astype("category")
```

[ ]:
```
0         w
1         f
2         f
3         f
4         f
         ..
396025    w
396026    f
396027    f
396028    f
396029    f
Name: initial_list_status, Length: 396030, dtype: category
Categories (2, object): ['f', 'w']
```

[ ]:
```
df["application_type"].value_counts()
df["application_type"].astype("category")
```

[ ]:
```
0         INDIVIDUAL
1         INDIVIDUAL
2         INDIVIDUAL
3         INDIVIDUAL
4         INDIVIDUAL
             …
396025    INDIVIDUAL
396026    INDIVIDUAL
396027    INDIVIDUAL
396028    INDIVIDUAL
396029    INDIVIDUAL
```

```
Name: application_type, Length: 396030, dtype: category
Categories (3, object): ['DIRECT_PAY', 'INDIVIDUAL', 'JOINT']
```

[ ]:
```
# By converting columns into categorical I have reduced the memory usage from␣
↪82 Mb to 68 Mb
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 25 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  int64
 1   term                  396030 non-null  int64
 2   int_rate              396030 non-null  float64
 3   grade                 396030 non-null  category
 4   sub_grade             396030 non-null  category
 5   emp_title             373103 non-null  object
 6   emp_length            377729 non-null  float64
 7   home_ownership        396030 non-null  object
 8   annual_inc            396030 non-null  float64
 9   verification_status   396030 non-null  category
 10  issue_d               396030 non-null  datetime64[ns]
 11  loan_status           396030 non-null  object
 12  purpose               396030 non-null  object
 13  dti                   396030 non-null  float64
 14  earliest_cr_line      396030 non-null  datetime64[ns]
 15  open_acc              396030 non-null  int64
 16  pub_rec               396030 non-null  int64
 17  revol_bal             396030 non-null  int64
 18  revol_util            395754 non-null  float64
 19  total_acc             396030 non-null  int64
 20  initial_list_status   396030 non-null  object
 21  application_type      396030 non-null  object
 22  mort_acc              358235 non-null  float64
 23  pub_rec_bankruptcies  395495 non-null  float64
 24  zipcode               396030 non-null  object
dtypes: category(3), datetime64[ns](2), float64(7), int64(6), object(7)
memory usage: 67.6+ MB
```

[ ]:
```
df["issue_year"]=df["issue_d"].dt.year
df["issue_month"]=df["issue_d"].dt.month
df["earliest_cr_year"]=df["earliest_cr_line"].dt.year
df["earliest_cr_month"]=df["earliest_cr_line"].dt.month
```

[ ]:
```
df.drop(columns=["issue_d","earliest_cr_line"], axis=1, inplace=True)
```

Handling Outlier and Null values

[ ]:
```
df_copy = df.copy()
```

[ ]:
```
num_cols = ["loan_amnt","int_rate","annual_inc","dti","revol_bal","revol_util",␣
↪"total_acc","pub_rec","open_acc","mort_acc","pub_rec_bankruptcies"]

def remove_outliers(df, cols):
  for col in cols:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    IQR = q3-q1
    lower_bound = q1-(1.5*IQR)
    upper_bound = q3+(1.5*IQR)
    df = df[(df[col]>=lower_bound) & (df[col]<=upper_bound)]
  return df
```

[ ]:
```
df_copy = remove_outliers(df_copy, num_cols)
```

[ ]:
```
# After removing outliers, There is a lot of information loss, almost 36% data␣
↪loss, which is very significant
# But still we have to remove outliers, but now we discard less important␣
↪columns from the list
df_copy.shape
```

[ ]:
```
(253377, 27)
```

[ ]:
```
# creating one more copy of DataFrame
df_new = df.copy()
```

[ ]:
```
num_cols = ["loan_amnt","int_rate","annual_inc","dti","revol_bal","revol_util"]

def remove_outliers(df, cols):
  for col in cols:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    IQR = q3-q1
    lower_bound = q1-(1.5*IQR)
    upper_bound = q3+(1.5*IQR)
    df = df[(df[col]>=lower_bound) & (df[col]<=upper_bound)]
  return df
```

[ ]:
```
df_new = remove_outliers(df_new, num_cols)
```

[ ]:
```
df_new.isnull().sum()
```

[ ]:
```
loan_amnt                    0
term                         0
int_rate                     0
grade                        0
sub_grade                    0
emp_title                21099
emp_length               17289
home_ownership               0
annual_inc                   0
verification_status          0
loan_status                  0
purpose                      0
dti                          0
open_acc                     0
pub_rec                      0
revol_bal                    0
revol_util                   0
total_acc                    0
initial_list_status          0
application_type             0
mort_acc                 34714
pub_rec_bankruptcies       483
zipcode                      0
issue_year                   0
issue_month                  0
earliest_cr_year             0
earliest_cr_month            0
dtype: int64
```

[ ]:
```
df_new["emp_title"].nunique()
```

[ ]:
```
159668
```

- Since null values in pub_rec_bankruptcies are very less we can just drop those
- we have to impute null values in mort_acc and emp_length
- emp_title column has so many distinct values compared to its null values so that we can remove this column for now for our analysis

[ ]:
```
df_new.groupby(["emp_title","loan_status"]).size().unstack(fill_value=0).
↪sort_values(by="Fully Paid",ascending=False)
```

[ ]:
```
loan_status                          Charged Off  Fully Paid
emp_title
Teacher                                      798        3344
Manager                                      818        2954
Registered Nurse                             342        1376
Supervisor                                   361        1335
```

```
RN                                          341        1322
…                                         …          …
City of Northport                             1          0
City of Northport  Northport, Al              1          0
Warehouse Laborer                             1          0
Warehouse Labor                               1          0
Doumak Inc                                    1          0

[159668 rows x 2 columns]
```

- The above dataframe shows that teacher, manager and Nurse are the most common professions that had been afforded the loan

[ ]:
```
df_new.groupby(["purpose","loan_status"]).size().unstack().
↪sort_values(by="Fully Paid",ascending=False)
```

[ ]:
```
loan_status         Charged Off  Fully Paid
purpose
debt_consolidation        44077      168642
credit_card               12521       61511
home_improvement           3552       17079
other                      4033       15166
major_purchase             1324        6644
car                         579        3765
small_business             1365        3405
medical                     812        2983
moving                      607        2008
vacation                    436        1853
wedding                     203        1511
house                       359        1500
renewable_energy             70         233
educational                  38         200
```

- Most loans were given out for debt_consolidation,credit card refinancing.

[ ]:
```
df_new.isna().sum()
```

[ ]:
```
loan_amnt                  0
term                       0
int_rate                   0
grade                      0
sub_grade                  0
emp_title              21099
emp_length             17289
home_ownership             0
annual_inc                 0
verification_status        0
loan_status                0
```

```
purpose                       0
dti                           0
open_acc                      0
pub_rec                       0
revol_bal                     0
revol_util                    0
total_acc                     0
initial_list_status           0
application_type              0
mort_acc                  34714
pub_rec_bankruptcies        483
zipcode                       0
issue_year                    0
issue_month                   0
earliest_cr_year              0
earliest_cr_month             0
dtype: int64
```

[ ]:
```
# we are removing pub_rec_bankruptices because it has less number of null values
df_new.dropna(subset=["pub_rec_bankruptcies"],inplace=True)
```

[ ]:
```
# we are also removing emp_title because of large number of unique values and␣
↪it is less important column
df_new.drop(columns=["emp_title"],inplace=True)
```

[ ]:
```
df_new.isna().sum()
```

[ ]:
```
loan_amnt                     0
term                          0
int_rate                      0
grade                         0
sub_grade                     0
emp_length                17289
home_ownership                0
annual_inc                    0
verification_status           0
loan_status                   0
purpose                       0
dti                           0
open_acc                      0
pub_rec                       0
revol_bal                     0
revol_util                    0
total_acc                     0
initial_list_status           0
application_type              0
mort_acc                  34231
```

```
pub_rec_bankruptcies        0
zipcode                     0
issue_year                  0
issue_month                 0
earliest_cr_year            0
earliest_cr_month           0
dtype: int64
```

Handling mort_acc and emp_length columns, below strategies can be used for that:

- For emp_length we can use median to fill it or even mean.
- For mort_acc we can fill it with mean, but not the mean of whole data, we can group it by the total accounts and then take mean of mort_acc since if total accounts are more then there is more probability that the person will have more mort_acc.

[ ]:
```
df_new["emp_length"].median()
```

[ ]:
```
6.0
```

[ ]:
```
df_new["emp_length"] = df_new["emp_length"].fillna(df_new["emp_length"].
 ↪median())
```

[ ]:
```
# Now imputing mort_acc with the mean of mort_acc per total_acc
df_mort=round(df_new.groupby('total_acc')[['mort_acc']].mean(),0).reset_index()
```

[ ]:
```
df_mort
```

[ ]:
```
     total_acc  mort_acc
0            2       0.0
1            3       0.0
2            4       0.0
3            5       0.0
4            6       0.0
..         …         …
107        116       1.0
108        117       0.0
109        118       1.0
110        124       1.0
111        135       3.0

[112 rows x 2 columns]
```

[ ]:
```
# creating copy
df_new1=df_new.copy()
```

[ ]:
```
df_merged = df_new.merge(df_mort, how="left", on="total_acc",␣
 ↪suffixes=("","_mean"))
```

[ ]:
```
df_merged['mort_acc'].fillna(df_merged['mort_acc_mean']).isna().value_counts()
```

[ ]:
```
mort_acc
False    355993
Name: count, dtype: int64
```

[ ]:
```
df_merged['mort_acc'] = df_merged['mort_acc'].fillna(df_merged['mort_acc_mean'])
```

[ ]:
```
df_merged.drop(columns=['mort_acc_mean'],inplace=True)
```

[ ]:
```
df_merged
```

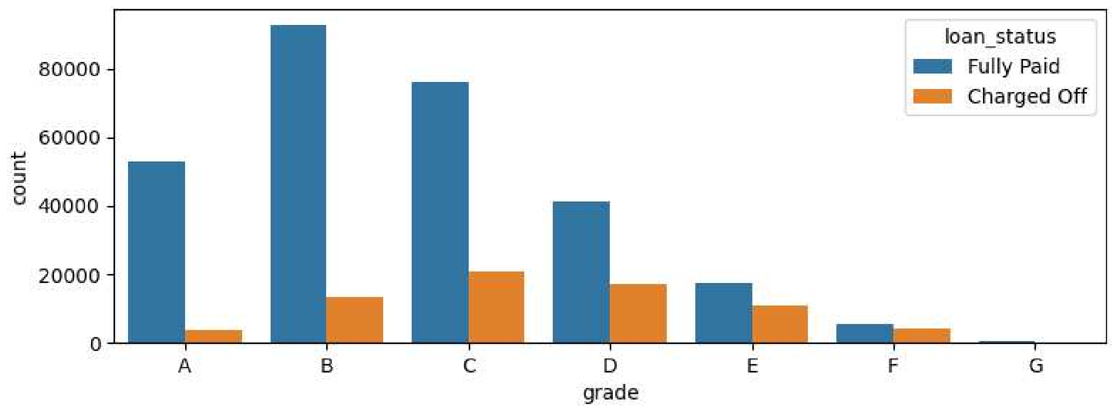
[ ]:
```
        loan_amnt  term  int_rate grade sub_grade  emp_length home_ownership  \
0           10000    36     11.44     B        B4        10.0           RENT
1            8000    36     11.99     B        B5         4.0       MORTGAGE
2           15600    36     10.49     B        B3         1.0           RENT
3            7200    36      6.49     A        A2         6.0           RENT
4           24375    60     17.27     C        C5         9.0       MORTGAGE
…             …   …       …     …     …         …          …
355988       6000    36     13.11     B        B4         5.0           RENT
355989      10000    60     10.99     B        B4         2.0           RENT
355990       5000    36      9.99     B        B1        10.0           RENT
355991      21000    60     15.31     C        C2        10.0       MORTGAGE
355992       2000    36     13.61     C        C2        10.0           RENT

        annual_inc verification_status  loan_status  … total_acc  \
0         117000.0        Not Verified   Fully Paid  …        25
1          65000.0        Not Verified   Fully Paid  …        27
2          43057.0     Source Verified   Fully Paid  …        26
3          54000.0        Not Verified   Fully Paid  …        13
4          55000.0            Verified  Charged Off  …        43
…              …                 …          …   …  …
355988     64000.0        Not Verified   Fully Paid  …         9
355989     40000.0     Source Verified   Fully Paid  …        23
355990     56500.0            Verified   Fully Paid  …        23
355991     64000.0            Verified   Fully Paid  …        20
355992     42996.0            Verified   Fully Paid  …        19

        initial_list_status application_type  mort_acc  pub_rec_bankruptcies  \
0                         w       INDIVIDUAL       0.0                   0.0
1                         f       INDIVIDUAL       3.0                   0.0
2                         f       INDIVIDUAL       0.0                   0.0
3                         f       INDIVIDUAL       0.0                   0.0
4                         f       INDIVIDUAL       1.0                   0.0
…                    …                …         …                   …
355988                    w       INDIVIDUAL       0.0                   0.0
355989                    w       INDIVIDUAL       0.0                   0.0
```

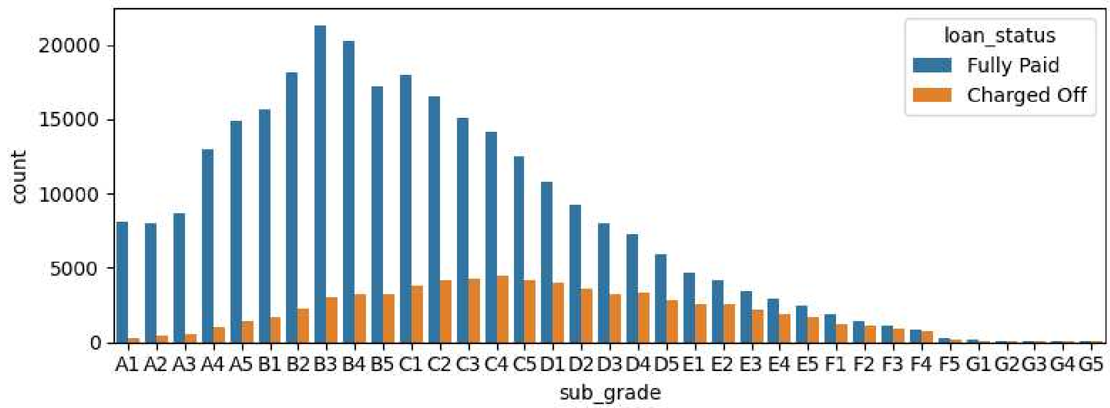
```
355990                    f        INDIVIDUAL       0.0                   0.0
355991                    f        INDIVIDUAL       5.0                   0.0
355992                    f        INDIVIDUAL       1.0                   0.0

        zipcode  issue_year issue_month earliest_cr_year  earliest_cr_month
0         22690        2015           1             1990                  6
1         05113        2015           1             2004                  7
2         05113        2015           1             2007                  8
3         00813        2014          11             2006                  9
4         11650        2013           4             1999                  3
…          …          …          …             …                 …
355988    05113        2013           3             1991                 11
355989    30723        2015          10             2004                 11
355990    70466        2013          10             1997                  3
355991    29597        2012           8             1990                 11
355992    48052        2010           6             1998                  9

[355993 rows x 26 columns]
```

Visualiazation and Graphic Analysis

Count plots of grade and sub-grade

[ ]:
```
fig,axes=plt.subplots(2,1,figsize=(8,6))
sns.countplot(data=df_merged,x='grade',hue='loan_status',ax=axes[0])
sns.countplot(data=df_merged,x='sub_grade',hue='loan_status',ax=axes[1])
plt.tight_layout()
plt.show()
```

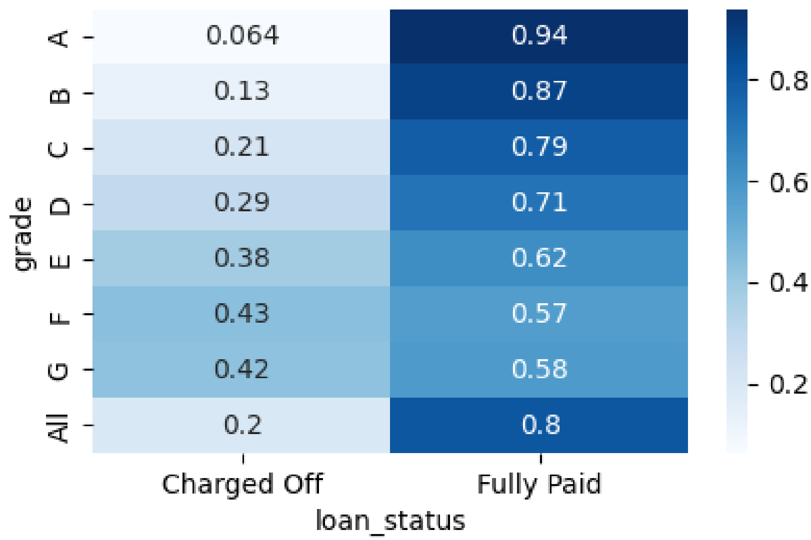




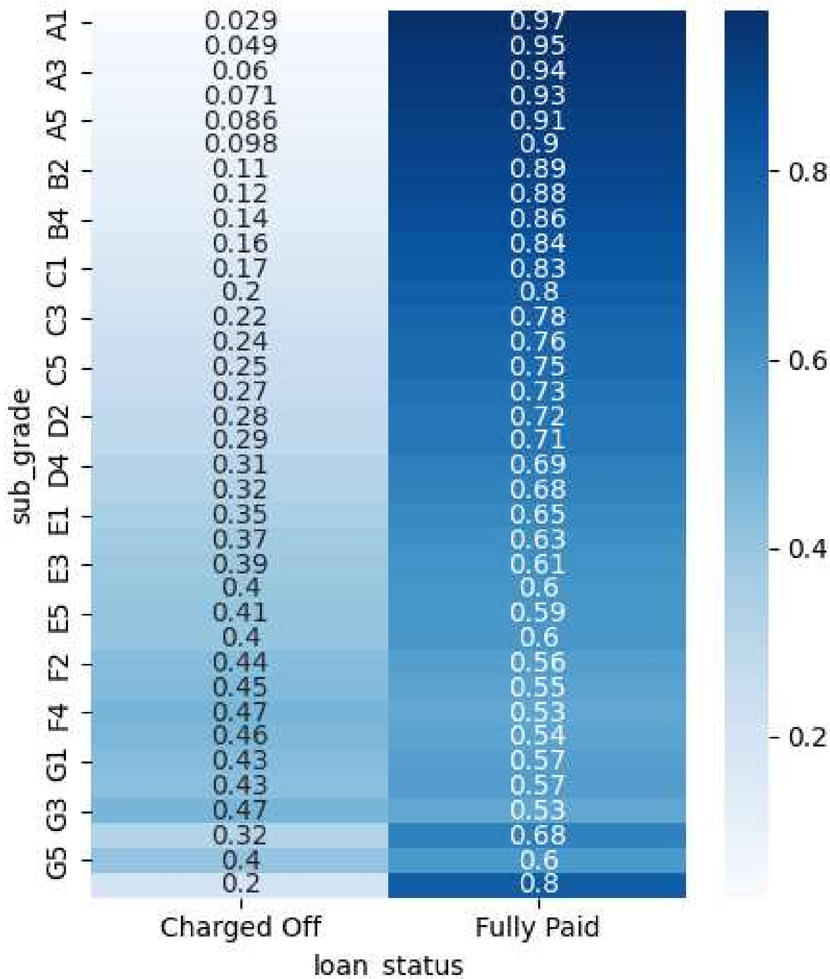
Observations from the above countplots:

- Most number of people in grade B paid off the loan.
- In Grade B most people in sub_grade B3 have paid off their loans.
- Very less number of people are categorised in G grade.
- The percentage of defaulters in sub_grade A1 is the least, implying genuine people.

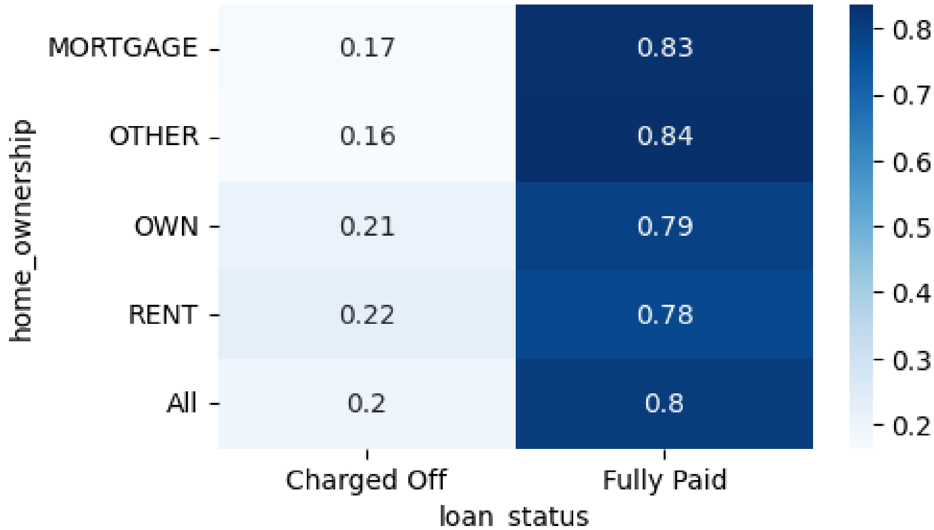
```
[ ]: plt.figure(figsize=(5,3))
     sns.heatmap(pd.
      ↪crosstab(df_merged['grade'],df_merged['loan_status'],normalize='index',margins=True),annot='
                  cmap='Blues')
     plt.show()
```



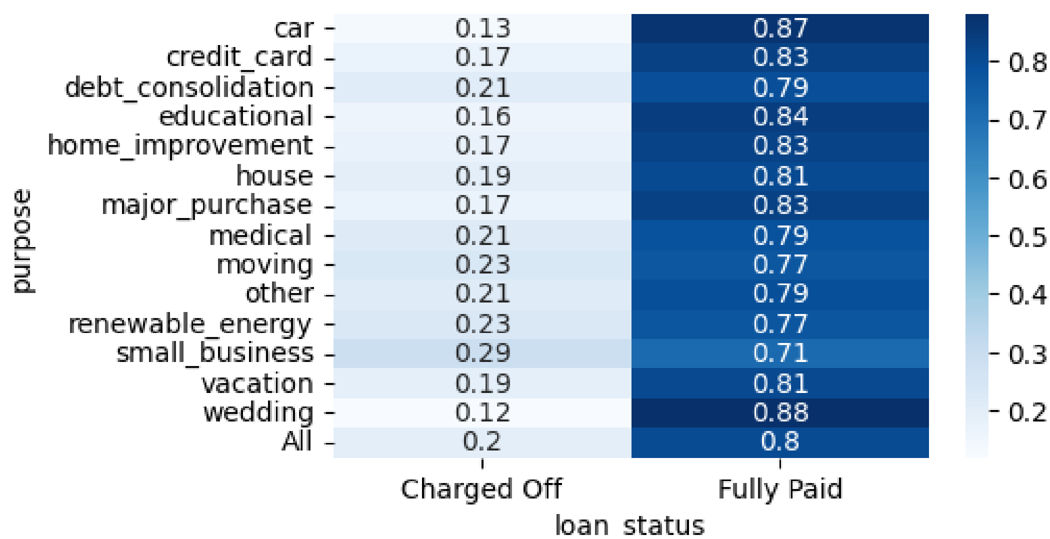
[ ]:
```
plt.figure(figsize=(5,6))
sns.heatmap(pd.
 ↪crosstab(df_merged['sub_grade'],df_merged['loan_status'],normalize='index',margins=True),an
             cmap='Blues')
plt.show()
```



[ ]:
```
plt.figure(figsize=(5,3))
sns.heatmap(pd.
 ↪crosstab(df_merged['home_ownership'],df_merged['loan_status'],normalize='index',margins=Tru
            cmap='Blues')
plt.show()
```




```
[ ]: plt.figure(figsize=(5,3))
sns.heatmap(pd.
  ↪crosstab(df_merged['purpose'],df_merged['loan_status'],normalize='index',margins=True),anno
            cmap='Blues')
plt.show()
```



Observations from the above heatmaps:

- 94% of people marked in grade A paid back the loan.
- The customers who have opted for joint loans are more likely to pay back.

- The people who have taken loans for Wedding, Car and educational purposes are most likely to pay back the loan.
- People who have taken loan for small business are least likely to pay back the loan.

Feature Engineering

[ ]:
```
df_merged
```

[ ]:
```
        loan_amnt  term  int_rate grade sub_grade  emp_length home_ownership  \
0           10000    36     11.44     B        B4        10.0           RENT
1            8000    36     11.99     B        B5         4.0       MORTGAGE
2           15600    36     10.49     B        B3         1.0           RENT
3            7200    36      6.49     A        A2         6.0           RENT
4           24375    60     17.27     C        C5         9.0       MORTGAGE
…             …   …        …   …        …         …          …
355988       6000    36     13.11     B        B4         5.0           RENT
355989      10000    60     10.99     B        B4         2.0           RENT
355990       5000    36      9.99     B        B1        10.0           RENT
355991      21000    60     15.31     C        C2        10.0       MORTGAGE
355992       2000    36     13.61     C        C2        10.0           RENT

        annual_inc verification_status  loan_status  … total_acc  \
0         117000.0        Not Verified   Fully Paid  …        25
1          65000.0        Not Verified   Fully Paid  …        27
2          43057.0     Source Verified   Fully Paid  …        26
3          54000.0        Not Verified   Fully Paid  …        13
4          55000.0            Verified  Charged Off  …        43
…              …                   …          …   …         …
355988     64000.0        Not Verified   Fully Paid  …         9
355989     40000.0     Source Verified   Fully Paid  …        23
355990     56500.0            Verified   Fully Paid  …        23
355991     64000.0            Verified   Fully Paid  …        20
355992     42996.0            Verified   Fully Paid  …        19

        initial_list_status  application_type  mort_acc  pub_rec_bankruptcies  \
0                         w        INDIVIDUAL       0.0                   0.0
1                         f        INDIVIDUAL       3.0                   0.0
2                         f        INDIVIDUAL       0.0                   0.0
3                         f        INDIVIDUAL       0.0                   0.0
4                         f        INDIVIDUAL       1.0                   0.0
…                      …               …       …                   …
355988                    w        INDIVIDUAL       0.0                   0.0
355989                    w        INDIVIDUAL       0.0                   0.0
355990                    f        INDIVIDUAL       0.0                   0.0
355991                    f        INDIVIDUAL       5.0                   0.0
355992                    f        INDIVIDUAL       1.0                   0.0

        zipcode  issue_year issue_month earliest_cr_year  earliest_cr_month
```

0	22690	2015	1	1990	6
1	05113	2015	1	2004	7
2	05113	2015	1	2007	8
3	00813	2014	11	2006	9
4	11650	2013	4	1999	3
…	…	…	…	…	…
355988	05113	2013	3	1991	11
355989	30723	2015	10	2004	11
355990	70466	2013	10	1997	3
355991	29597	2012	8	1990	11
355992	48052	2010	6	1998	9

[355993 rows x 26 columns]

[ ]:
```
# Loan status
map={'Charged Off':1,'Fully Paid':0}
df_merged['loan_status']=df_merged['loan_status'].map(map)
```

[ ]:
```
# Initial_list_status
map={'w':0,'f':1}
df_merged['initial_list_status']=df_merged['initial_list_status'].map(map)
```

[ ]:
```
x = df_merged.drop(columns=['loan_status'])
y = df_merged['loan_status']
```

[ ]:
```
x_train_cv, x_test, y_train_cv, y_test = train_test_split(x, y, test_size=0.25,␣
↪random_state=1)
x_train, x_test_cv, y_train, y_test_cv = train_test_split(x_train_cv,␣
↪y_train_cv, test_size=0.25, random_state=1)
```

[ ]:
```
# Target encoding of categorical columns (I have included zip code,
# year and month as they also have limited categorical values and are not␣
↪continous with more number of values)
encoding_cols =␣
↪['grade','sub_grade','home_ownership','verification_status','purpose',
                'application_type','zipcode']
# Target encoding should only be applied to category or object dtypes, it won't␣
↪work on continuous data
# type like int or float.
# In our dataset zipcode is an object and not int or float.
```

[ ]:
```
!pip install category_encoders
from category_encoders import TargetEncoder

encoder=TargetEncoder()

for i in encoding_cols:
```

```
    x_train[i]=encoder.fit_transform(x_train[i],y_train)
    x_test_cv[i]=encoder.transform(x_test_cv[i])
    x_test[i]=encoder.transform(x_test[i])
```

```
Requirement already satisfied: category_encoders in
/usr/local/lib/python3.10/dist-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (2.0.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->category_encoders) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->category_encoders) (2024.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.20.0->category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn>=0.20.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from
statsmodels>=0.9.0->category_encoders) (24.1)
```

[ ]:
```
# MinMax scaler for numerical columns
from sklearn.preprocessing import MinMaxScaler
numeric_cols =␣
 ↪['loan_amnt','term','int_rate','emp_length','annual_inc','dti','open_acc',
               ␣
 ↪'pub_rec','revol_bal','revol_util','total_acc','mort_acc','pub_rec_bankruptcies',
               ␣
 ↪'issue_year','issue_month','earliest_cr_year','earliest_cr_month']

minmax = MinMaxScaler()
```

[ ]:
```
x_train_scaled = minmax.fit_transform(x_train[numeric_cols])
x_test_cv_scaled=minmax.transform(x_test_cv[numeric_cols])
x_test_scaled=minmax.transform(x_test[numeric_cols])
```

[ ]:
```
x_train[numeric_cols]=x_train_scaled
x_test_cv[numeric_cols]=x_test_cv_scaled
x_test[numeric_cols]=x_test_scaled
```

[ ]:
```
table_styles = {
    'cerulean_palette': [{
        'selector': 'th',
        'props': [('background-color', '#4E79A7'),
                  ('color', 'white'),
                  ('font-family', 'verdana')]
    }, {
        'selector': 'td',
        'props': [('background-color', '#A0CBE8'),
                  ('color', '#000000'),
                  ('font-family', 'verdana')]
    }]
}
display(x_train.head(10).style.
 ↪set_table_styles(table_styles['cerulean_palette']).set_caption("Scaled Final␣
 ↪Data"))
```

```
<pandas.io.formats.style.Styler at 0x7b8ba7d4bcd0>
```

Modeling

[ ]:
```
# Let's fit the logistic Regression model and check

model = LogisticRegression(random_state=1,max_iter=500,n_jobs=-1)
model.fit(x_train,y_train)
```

[ ]:
```
LogisticRegression(max_iter=500, n_jobs=-1, random_state=1)
```

[ ]:
```
y_pred=model.predict(x_test)
print(f" Accuracy score of the {model} = {round(model.
 ↪score(x_test,y_test)*100,2)}")
```

```
 Accuracy score of the LogisticRegression(max_iter=500, n_jobs=-1,
random_state=1) = 88.7
```

The Accuracy of our initial model is 88.68% which seems to be decent, but the catch is accuracy is not the best score when you have imbalanced dataset. Since ours is an imbalanced dataset accuracy can be sometime misleading so we will check some more evaulation metric to see whether we have a good model or not.

[ ]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93     71438
           1       0.92      0.47      0.62     17561

    accuracy                           0.89     88999
   macro avg       0.90      0.73      0.78     88999
weighted avg       0.89      0.89      0.87     88999
```

[ ]:
```
confusion_matrix(y_test,y_pred)
```

[ ]:
```
array([[70760,   678],
       [ 9381,  8180]])
```

Observations from classification report and confusion matrix:

- The precision is high for class 0.
- The precision of class 1 (defaulter) is also high which is a good thing, high pecision for class 1 means if our model predicts that the customer is a defaulter then there is a 92% chance that the person is a defaulter, this might save the bank from defaulters.
- The recall for class 0 is also very high that means it classifies most of the non defaulers correctly.
- The problematic thing is the recall of class 1, basically recall (tp/(tp+fn)) tells us how well it can catch the defaulters, since recall for class 1 is very low which implies false negative is high, this means though the person is a defaulter our model is classifying him/her into non defaulter, this might be because of 2 reasons:
- a) The data is highly imbalanced and there are more records class 0
- b) We have taken a defalt thereshold of 0.5 currently to classify into class 1 or class 0, which should be changed according to problem statement

Evaluation ROC Curve An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

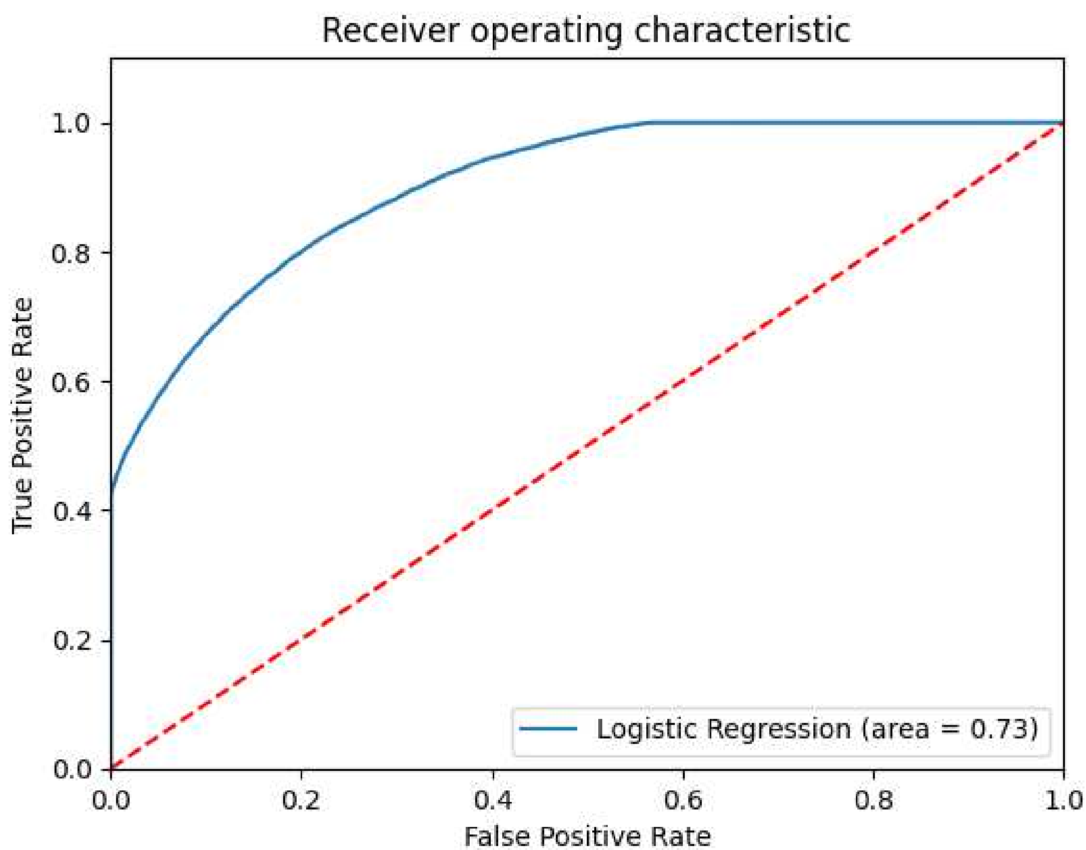
True Positive Rate False Positive Rate True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

TPR=(TP)/(TP+FN) False Positive Rate (FPR) is defined as follows:

FPR=(FP)/(FP+TN) An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

[ ]:
```
roc_auc_score(y_test,y_pred)
fpr,tpr,thr = roc_curve(y_test,model.predict_proba(x_test)[:,1])
```

[ ]:
```
score = roc_auc_score(y_test,y_pred)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % score)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.10])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



The above plot shows us the true positive rate and false positive rates for different thresholds of probabilities.

The higher the area of blue curve the better is our model.

The red dotted line shows a random model which has an roc_auc_score of 0.5.

The threshold is chosen based on optimum value of tpr and fpr according to the business problem being solved.

Precision Recall Curve

[ ]:
```
precision,recall,thr = precision_recall_curve(y_test,model.
 ↪predict_proba(x_test)[:,1])
```

[ ]:
```
# Precision and recall will have one extra value.
# The extra value in the precision and recall arrays ensures that all relevant␣
 ↪decision thresholds are covered, including the boundary cases where the␣
 ↪number of predicted positive samples is either zero or equal to the total␣
 ↪number of samples.
len(thr)
```

[ ]: 88999

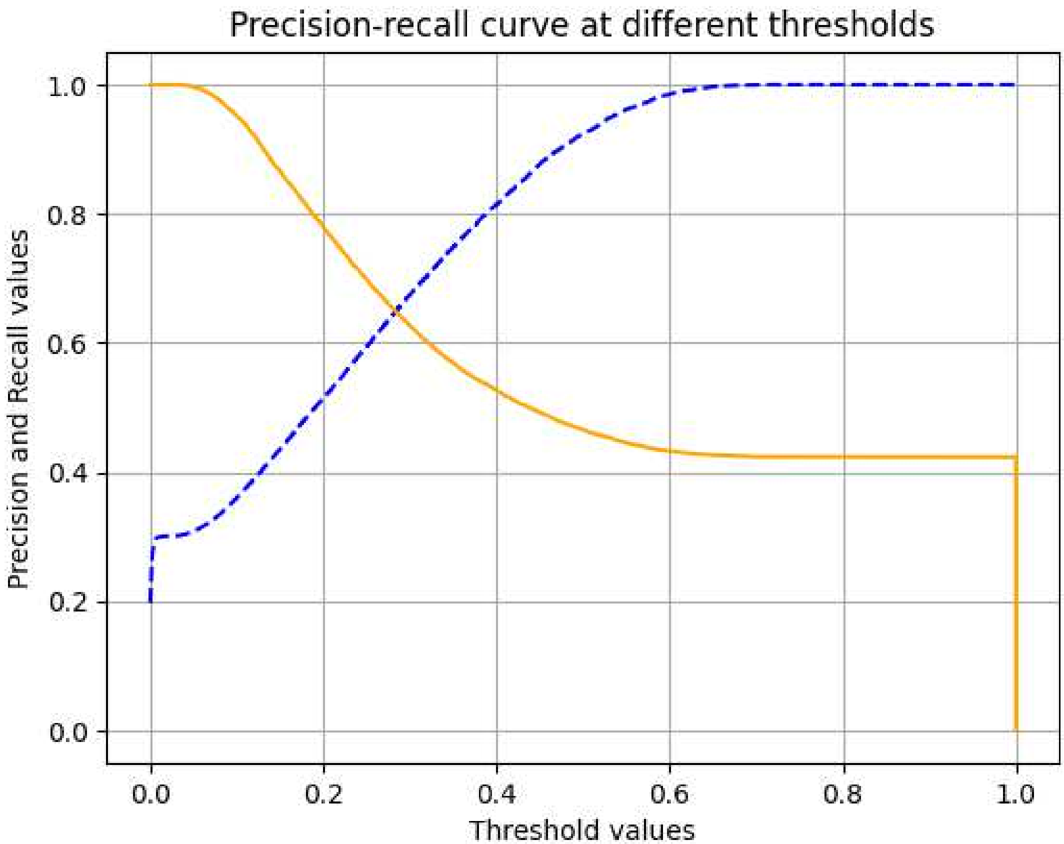
[ ]:
```
len(precision)
```

[ ]: 89000

[ ]:
```
len(recall)
```

[ ]: 89000

[ ]:
```
plt.figure()
plt.plot(thr,precision[0:
 ↪len(thr)],linestyle='--',label='precision',color='blue')
plt.plot(thr,recall[0:len(thr)],label='recall',color='orange')
plt.title("Precision-recall curve at different thresholds")
plt.xlabel("Threshold values")
plt.ylabel("Precision and Recall values")
plt.grid()
plt.show()
```



The above curve shows us precision and recall values for different thresholds, since both precision and recall are important in our business problem we have to consider a value of 0.3 for the threshold, this means if a model predicts the probability for any person to be more than 0.3 then he/she should be classified as defaulter.

[ ]:
```
y_pred = model.predict_proba(x_test)[:,1]

threshold_considered = 0.3

y_pred_custom = (y_pred>threshold_considered).astype('int')
y_pred_custom
```

[ ]:
```
array([0, 1, 1, …, 0, 0, 0])
```

[ ]:
```
print(classification_report(y_test,y_pred_custom))
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92     71438
           1       0.67      0.63      0.65     17561
```

```
    accuracy                           0.87     88999
   macro avg       0.79      0.78      0.78     88999
weighted avg       0.86      0.87      0.86     88999
```

[ ]:
```
confusion_matrix(y_test,y_pred_custom)
```

[ ]:
```
array([[66093,  5345],
       [ 6543, 11018]])
```

[ ]:
```
# Let's do multicollinearity check and remove some features which have high VIF
↪and refit the model
```

Multicollinearity check with VIF

- Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. Multicollinearity can be a problem in a regression model because we would not be able to distinguish between the individual effects of the independent variables on the dependent variable.
- Multicollinearity can be detected via various methods. One such method is Variance Inflation Factor aka VIF. In VIF method, we pick each independent feature and regress it against all of the other independent features. VIF score of an independent variable represents how well the variable is explained by other independent variables.
- VIF = 1/1-R2

[ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(x):
    # Calculating the VIF
    vif = pd.DataFrame()
    vif['Feature'] = x.columns
    vif['VIF'] = [variance_inflation_factor(x.values, i) for i in range(x.
↪shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by='VIF', ascending = False)
    return vif

calc_vif(x_train)
```

[ ]:
```
                 Feature     VIF
17      application_type  252.75
4              sub_grade  173.54
3                  grade  103.68
6         home_ownership   83.07
2               int_rate   79.69
9                purpose   76.10
8    verification_status   36.23
21            issue_year   31.36
23      earliest_cr_year   19.20
```

```
15            total_acc   11.44
11             open_acc   11.37
14           revol_util    9.21
7            annual_inc    9.15
10                  dti    8.11
0             loan_amnt    6.58
13            revol_bal    5.96
22          issue_month    3.85
24    earliest_cr_month    3.84
5            emp_length    3.41
16  initial_list_status    3.23
18             mort_acc    2.74
19 pub_rec_bankruptcies    2.48
12              pub_rec    2.47
1                  term    1.99
20              zipcode    1.63
```

Let's remove the top most vif feature and re-calculate the vif

[ ]:
```python
removed_features =[]
removed_features.append('application_type')
x_train.drop(columns=['application_type'],inplace=True)
```

[ ]:
```python
calc_vif(x_train)
```

[ ]:
```
                Feature     VIF
4             sub_grade  172.45
3                 grade  103.58
2              int_rate   79.55
9               purpose   57.84
6        home_ownership   54.49
8   verification_status   33.45
20           issue_year   28.62
22     earliest_cr_year   17.91
15            total_acc   11.43
11             open_acc   11.31
14           revol_util    9.06
7            annual_inc    9.00
10                  dti    8.10
0             loan_amnt    6.56
13            revol_bal    5.96
23    earliest_cr_month    3.79
21          issue_month    3.75
5            emp_length    3.34
16  initial_list_status    3.09
17             mort_acc    2.57
18 pub_rec_bankruptcies    2.48
```

```
12                 pub_rec    2.47
1                     term    1.99
19                 zipcode    1.63
```

Since the VIF is very high we can remove 3-4 features at once and can check for VIF again

```
[ ]: removed_features.extend(['sub_grade','grade','int_rate','purpose'])
     x_train.drop(columns=['sub_grade','grade','int_rate','purpose'],inplace=True)
```

```
[ ]: calc_vif(x_train)
```

```
[ ]:                  Feature    VIF
     3          home_ownership  43.12
     5     verification_status  31.69
     16             issue_year  26.61
     18       earliest_cr_year  17.53
     11              total_acc  11.40
     7                open_acc  11.19
     4              annual_inc   8.84
     6                     dti   8.05
     10             revol_util   7.95
     0               loan_amnt   6.52
     9               revol_bal   5.76
     19      earliest_cr_month   3.77
     17            issue_month   3.70
     2              emp_length   3.30
     12    initial_list_status   3.00
     13               mort_acc   2.51
     14   pub_rec_bankruptcies   2.48
     8                 pub_rec   2.46
     1                    term   1.62
     15                zipcode   1.61
```

```
[ ]: removed_features.
       ↪extend(['home_ownership','earliest_cr_year','verification_status','issue_year'])
     x_train.
       ↪drop(columns=['home_ownership','earliest_cr_year','verification_status','issue_year'],inpla
```

```
[ ]: calc_vif(x_train)
```

```
[ ]:                  Feature    VIF
     9               total_acc  11.09
     5                open_acc  10.64
     3              annual_inc   7.87
     4                     dti   6.99
     8              revol_util   6.73
     0               loan_amnt   6.14
```

```
7               revol_bal   5.31
15      earliest_cr_month   3.45
14            issue_month   3.29
2              emp_length   3.16
12   pub_rec_bankruptcies   2.47
6                 pub_rec   2.44
10    initial_list_status   2.40
11               mort_acc   2.31
1                    term   1.60
13                zipcode   1.59
```

[ ]:
```
# Now with these features let's refit the model
removed_features
x_test.drop(columns=removed_features,inplace=True)
```

[ ]:
```
x_test_cv.drop(columns=removed_features,inplace=True)
```

[ ]:
```
model2 =␣
 ↪LogisticRegression(max_iter=1000,solver='liblinear',random_state=1,n_jobs=-1)
```

[ ]:
```
model2.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1211:
UserWarning: 'n_jobs' > 1 does not have any effect when 'solver' is set to
'liblinear'. Got 'n_jobs' = 2.
  warnings.warn(
```

[ ]:
```
LogisticRegression(max_iter=1000, n_jobs=-1, random_state=1, solver='liblinear')
```

[ ]:
```
print(classification_report(y_test,model2.predict(x_test)))
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93     71438
           1       0.97      0.44      0.60     17561

    accuracy                           0.89     88999
   macro avg       0.93      0.72      0.77     88999
weighted avg       0.90      0.89      0.87     88999
```

[ ]:
```
confusion_matrix(y_test,model2.predict(x_test))
```

[ ]:
```
array([[71220,   218],
       [ 9862,  7699]])
```

Oversampling using SMOTE to solve the imbalance issue

[ ]:
```python
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=1,n_jobs=-1,k_neighbors=5)
```

[ ]:
```python
x_train_new,y_train_new = sm.fit_resample(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-
packages/imblearn/over_sampling/_smote/base.py:336: FutureWarning: The parameter
`n_jobs` has been deprecated in 0.10 and will be removed in 0.12. You can pass
an nearest neighbors estimator where `n_jobs` is already set instead.
  warnings.warn(
```

[ ]:
```python
print(f"After OverSampling, the shape of train_X:{x_train_new.shape}")
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_new.
 ↪shape))

print(f"After OverSampling, counts of label '1': {(sum(y_train_new == 1))}")
print(f"After OverSampling, counts of label '0':{np.sum(y_train_new==0)}")
```

```
After OverSampling, the shape of train_X:(321892, 16)
After OverSampling, the shape of train_y: (321892,)

After OverSampling, counts of label '1': 160946
After OverSampling, counts of label '0':160946
```

[ ]:
```python
model3 = LogisticRegression(max_iter=1000,random_state=1,n_jobs=-1,)
model3.fit(x_train_new, y_train_new)
predictions = model3.predict(x_test)

# Classification Report
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.80      0.86     71438
           1       0.48      0.77      0.59     17561

    accuracy                           0.79     88999
   macro avg       0.71      0.78      0.73     88999
weighted avg       0.84      0.79      0.81     88999
```

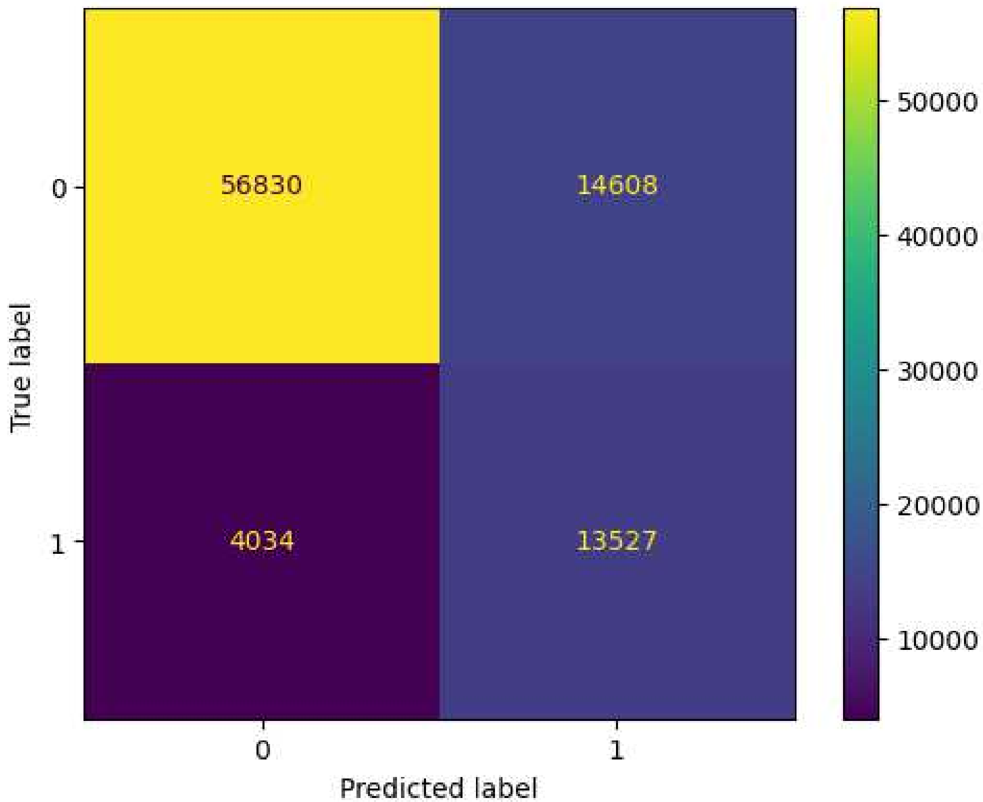
[ ]:
```python
cm=confusion_matrix(y_test,predictions)
```

[ ]:
```python
model3.score(x_test,y_test)
```

[ ]: 0.7905369723255318

[ ]:
```python
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay(cm).plot()
```

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b8ba3e626e0>



0.1 Insights and Recommendations

Insights

1. Loan Grade Concentration: The concentration of loans in the B, C, and A categories suggests these grades are most common and potentially the most reliable segments.
2. Employment Stability and Loan Purpose: The long employment duration of many borrowers indicates stability, which is a positive sign for lenders. The primary purpose of loans being debt consolidation reflects a trend towards financial management and restructuring by borrowers.
3. Correlation Insights: The high correlation between loan amount and installment (0.95 Pearson, 0.97 Spearman) suggests that these two variables are strongly linked in their behavior. The minimal difference in the pearson and spearman correlation coefficients indicate that there is minimal to no involvement of non linearity in the data due to which did not check for any polynomial features.

4. Verification Status: The fact that verification doesn't necessarily guarantee a loan being fully paid implies that other factors play a significant role in loan performance.

Recommendations

1. Tailored Loan Products: Develop tailored loan products for the most common borrower segments (B, C, and A grade borrowers) to enhance product-market fit.
2. Loan Term Structuring: Offer more flexible terms for higher-grade loans and consider stricter terms for longer-term and lower-grade loans to mitigate risk.
3. Further Statistical Analysis: Conduct further statistical tests to validate the significance of observed correlations and insights, ensuring that lending strategies are data-driven.
4. Monitoring and Adjustment of Models: Continuously monitor and adjust credit scoring models in response to changes in borrower behavior and economic conditions.