In [1]:
```
# import the library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,f1_score,precision_score,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, recall_scor
# ignore the warning if any
import warnings
warnings.filterwarnings("ignore")

# set row/columns
pd.options.display.max_columns= None
pd.options.display.max_rows= None
np.set_printoptions(suppress=True)
```

Dataset Direct download:

https://storage.googleapis.com/kagglesdsdata/datasets/902/370089/accepted_2007_to_2018Q4
X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20231223%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20231223T025121Z&X-Goog-Expires=259200&X-Goog-SignedHeaders=host&X-Goog-Signature=a35dba0c10e728f25bb13af7d77452454c73bd6454a09ada7f69de83226ffc4addb4912

In [2]:
```
# read the dataset
# https://www.kaggle.com/datasets/wordsforthewise/lending-club
df = pd.read_csv("E://accepted_2007_to_2018Q4.csv")
```

In [3]:
```
# display top 5 rows to see how data looks like
df.head(5)
```

Out[3]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	in
0	68407277	NaN	3600.0	3600.0	3600.0	36 months	13.99	
1	68355089	NaN	24700.0	24700.0	24700.0	36 months	11.99	
2	68341763	NaN	20000.0	20000.0	20000.0	60 months	10.78	
3	66310712	NaN	35000.0	35000.0	35000.0	60 months	14.85	
4	68476807	NaN	10400.0	10400.0	10400.0	60 months	22.45	

In [4]:
```
# check the shape
df.shape
```

Out[4]:
```
(2260701, 151)
```

In [5]:
```
# check the data frame information
df.info(verbose= True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2260701 entries, 0 to 2260700
Data columns (total 151 columns):
 #    Column                                        Dtype
---   ------                                        -----
 0    id                                            object
 1    member_id                                     float64
 2    loan_amnt                                     float64
 3    funded_amnt                                   float64
 4    funded_amnt_inv                               float64
 5    term                                          object
 6    int_rate                                      float64
 7    installment                                   float64
 8    grade                                         object
 9    sub_grade                                     object
 10   emp_title                                     object
 11   emp_length                                    object
 12   home_ownership                                object
 13   annual_inc                                    float64
 14   verification_status                           object
 15   issue_d                                       object
 16   loan_status                                   object
 17   pymnt_plan                                    object
 18   url                                           object
 19   desc                                          object
 20   purpose                                       object
 21   title                                         object
 22   zip_code                                      object
 23   addr_state                                    object
 24   dti                                           float64
 25   delinq_2yrs                                   float64
 26   earliest_cr_line                              object
 27   fico_range_low                                float64
 28   fico_range_high                               float64
 29   inq_last_6mths                                float64
 30   mths_since_last_delinq                        float64
 31   mths_since_last_record                        float64
 32   open_acc                                      float64
 33   pub_rec                                       float64
 34   revol_bal                                     float64
 35   revol_util                                    float64
 36   total_acc                                     float64
 37   initial_list_status                           object
 38   out_prncp                                     float64
 39   out_prncp_inv                                 float64
 40   total_pymnt                                   float64
 41   total_pymnt_inv                               float64
 42   total_rec_prncp                               float64
 43   total_rec_int                                 float64
 44   total_rec_late_fee                            float64
 45   recoveries                                    float64
 46   collection_recovery_fee                       float64
 47   last_pymnt_d                                  object
 48   last_pymnt_amnt                               float64
 49   next_pymnt_d                                  object
 50   last_credit_pull_d                            object
```

```
51   last_fico_range_high                        float64
52   last_fico_range_low                         float64
53   collections_12_mths_ex_med                  float64
54   mths_since_last_major_derog                 float64
55   policy_code                                 float64
56   application_type                            object
57   annual_inc_joint                            float64
58   dti_joint                                   float64
59   verification_status_joint                   object
60   acc_now_delinq                              float64
61   tot_coll_amt                                float64
62   tot_cur_bal                                 float64
63   open_acc_6m                                 float64
64   open_act_il                                 float64
65   open_il_12m                                 float64
66   open_il_24m                                 float64
67   mths_since_rcnt_il                          float64
68   total_bal_il                                float64
69   il_util                                     float64
70   open_rv_12m                                 float64
71   open_rv_24m                                 float64
72   max_bal_bc                                  float64
73   all_util                                    float64
74   total_rev_hi_lim                            float64
75   inq_fi                                      float64
76   total_cu_tl                                 float64
77   inq_last_12m                                float64
78   acc_open_past_24mths                        float64
79   avg_cur_bal                                 float64
80   bc_open_to_buy                              float64
81   bc_util                                     float64
82   chargeoff_within_12_mths                    float64
83   delinq_amnt                                 float64
84   mo_sin_old_il_acct                          float64
85   mo_sin_old_rev_tl_op                        float64
86   mo_sin_rcnt_rev_tl_op                       float64
87   mo_sin_rcnt_tl                              float64
88   mort_acc                                    float64
89   mths_since_recent_bc                        float64
90   mths_since_recent_bc_dlq                    float64
91   mths_since_recent_inq                       float64
92   mths_since_recent_revol_delinq              float64
93   num_accts_ever_120_pd                       float64
94   num_actv_bc_tl                              float64
95   num_actv_rev_tl                             float64
96   num_bc_sats                                 float64
97   num_bc_tl                                   float64
98   num_il_tl                                   float64
99   num_op_rev_tl                               float64
100  num_rev_accts                               float64
101  num_rev_tl_bal_gt_0                         float64
102  num_sats                                    float64
103  num_tl_120dpd_2m                            float64
104  num_tl_30dpd                                float64
105  num_tl_90g_dpd_24m                          float64
106  num_tl_op_past_12m                          float64
```

```
 107  pct_tl_nvr_dlq                              float64
 108  percent_bc_gt_75                            float64
 109  pub_rec_bankruptcies                        float64
 110  tax_liens                                   float64
 111  tot_hi_cred_lim                             float64
 112  total_bal_ex_mort                           float64
 113  total_bc_limit                              float64
 114  total_il_high_credit_limit                  float64
 115  revol_bal_joint                             float64
 116  sec_app_fico_range_low                      float64
 117  sec_app_fico_range_high                     float64
 118  sec_app_earliest_cr_line                    object
 119  sec_app_inq_last_6mths                      float64
 120  sec_app_mort_acc                            float64
 121  sec_app_open_acc                            float64
 122  sec_app_revol_util                          float64
 123  sec_app_open_act_il                         float64
 124  sec_app_num_rev_accts                       float64
 125  sec_app_chargeoff_within_12_mths            float64
 126  sec_app_collections_12_mths_ex_med          float64
 127  sec_app_mths_since_last_major_derog         float64
 128  hardship_flag                               object
 129  hardship_type                               object
 130  hardship_reason                             object
 131  hardship_status                             object
 132  deferral_term                               float64
 133  hardship_amount                             float64
 134  hardship_start_date                         object
 135  hardship_end_date                           object
 136  payment_plan_start_date                     object
 137  hardship_length                             float64
 138  hardship_dpd                                float64
 139  hardship_loan_status                        object
 140  orig_projected_additional_accrued_interest  float64
 141  hardship_payoff_balance_amount              float64
 142  hardship_last_payment_amount                float64
 143  disbursement_method                         object
 144  debt_settlement_flag                        object
 145  debt_settlement_flag_date                   object
 146  settlement_status                           object
 147  settlement_date                             object
 148  settlement_amount                           float64
 149  settlement_percentage                       float64
 150  settlement_term                             float64
dtypes: float64(113), object(38)
memory usage: 2.5+ GB
```

In [6]:

```
# Generate descriptive statistics, so as to get the overall behaviour of different
# i.e. its mean, standard deviation, minimum/maximum value ,
# Descriptive statistics include those that summarize the central tendency,
# dispersion and shape of a dataset’s distribution, excluding NaN values.
#For numeric data, the result’s index will include count, mean, std, min, max as we
#50 and upper percentiles. By default the lower percentile is 25 and the upper perc
#The 50 percentile is the same as the median.
df.describe()
```

Out[6]:

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	installm
count	0.0	2.260668e+06	2.260668e+06	2.260668e+06	2.260668e+06	2.260668e
mean	NaN	1.504693e+04	1.504166e+04	1.502344e+04	1.309283e+01	4.458068e
std	NaN	9.190245e+03	9.188413e+03	9.192332e+03	4.832138e+00	2.671735e
min	NaN	5.000000e+02	5.000000e+02	0.000000e+00	5.310000e+00	4.930000e
25%	NaN	8.000000e+03	8.000000e+03	8.000000e+03	9.490000e+00	2.516500e
50%	NaN	1.290000e+04	1.287500e+04	1.280000e+04	1.262000e+01	3.779900e
75%	NaN	2.000000e+04	2.000000e+04	2.000000e+04	1.599000e+01	5.933200e
max	NaN	4.000000e+04	4.000000e+04	4.000000e+04	3.099000e+01	1.719830e

In [7]:

```
# check the number of NaN values (also known as Missing Values) present in each col
df.isnull().sum()
```

```
Out[7]: id                                   0
        member_id                      2260701
        loan_amnt                           33
        funded_amnt                         33
        funded_amnt_inv                     33
        term                                33
        int_rate                            33
        installment                         33
        grade                               33
        sub_grade                           33
        emp_title                       167002
        emp_length                      146940
        home_ownership                      33
        annual_inc                          37
        verification_status                 33
        issue_d                             33
        loan_status                         33
        pymnt_plan                          33
        url                                 33
        desc                           2134634
        purpose                             33
        title                            23358
        zip_code                            34
        addr_state                          33
        dti                               1744
        delinq_2yrs                         62
        earliest_cr_line                    62
        fico_range_low                      33
        fico_range_high                     33
        inq_last_6mths                      63
        mths_since_last_delinq         1158535
        mths_since_last_record         1901545
        open_acc                            62
        pub_rec                             62
        revol_bal                           33
        revol_util                        1835
        total_acc                           62
        initial_list_status                 33
        out_prncp                           33
        out_prncp_inv                       33
        total_pymnt                         33
        total_pymnt_inv                     33
        total_rec_prncp                     33
        total_rec_int                       33
        total_rec_late_fee                  33
        recoveries                          33
        collection_recovery_fee             33
        last_pymnt_d                      2460
        last_pymnt_amnt                     33
        next_pymnt_d                   1345343
        last_credit_pull_d                 105
        last_fico_range_high                33
        last_fico_range_low                 33
        collections_12_mths_ex_med         178
        mths_since_last_major_derog    1679926
        policy_code                         33
```

```
application_type                                33
annual_inc_joint                           2139991
dti_joint                                  2139995
verification_status_joint                  2144971
acc_now_delinq                                  62
tot_coll_amt                                 70309
tot_cur_bal                                  70309
open_acc_6m                                 866163
open_act_il                                 866162
open_il_12m                                 866162
open_il_24m                                 866162
mths_since_rcnt_il                          909957
total_bal_il                                866162
il_util                                    1068883
open_rv_12m                                 866162
open_rv_24m                                 866162
max_bal_bc                                  866162
all_util                                    866381
total_rev_hi_lim                             70309
inq_fi                                      866162
total_cu_tl                                 866163
inq_last_12m                                866163
acc_open_past_24mths                         50063
avg_cur_bal                                  70379
bc_open_to_buy                               74968
bc_util                                      76104
chargeoff_within_12_mths                       178
delinq_amnt                                     62
mo_sin_old_il_acct                          139104
mo_sin_old_rev_tl_op                         70310
mo_sin_rcnt_rev_tl_op                        70310
mo_sin_rcnt_tl                               70309
mort_acc                                     50063
mths_since_recent_bc                         73445
mths_since_recent_bc_dlq                   1741000
mths_since_recent_inq                       295468
mths_since_recent_revol_delinq             1520342
num_accts_ever_120_pd                        70309
num_actv_bc_tl                               70309
num_actv_rev_tl                              70309
num_bc_sats                                  58623
num_bc_tl                                    70309
num_il_tl                                    70309
num_op_rev_tl                                70309
num_rev_accts                                70310
num_rev_tl_bal_gt_0                          70309
num_sats                                     58623
num_tl_120dpd_2m                            153690
num_tl_30dpd                                 70309
num_tl_90g_dpd_24m                           70309
num_tl_op_past_12m                           70309
pct_tl_nvr_dlq                               70464
percent_bc_gt_75                             75412
pub_rec_bankruptcies                          1398
tax_liens                                      138
tot_hi_cred_lim                              70309
```

```
total_bal_ex_mort                               50063
total_bc_limit                                  50063
total_il_high_credit_limit                      70309
revol_bal_joint                               2152681
sec_app_fico_range_low                        2152680
sec_app_fico_range_high                       2152680
sec_app_earliest_cr_line                      2152680
sec_app_inq_last_6mths                        2152680
sec_app_mort_acc                              2152680
sec_app_open_acc                              2152680
sec_app_revol_util                            2154517
sec_app_open_act_il                           2152680
sec_app_num_rev_accts                         2152680
sec_app_chargeoff_within_12_mths              2152680
sec_app_collections_12_mths_ex_med            2152680
sec_app_mths_since_last_major_derog           2224759
hardship_flag                                      33
hardship_type                                 2249784
hardship_reason                               2249784
hardship_status                               2249784
deferral_term                                 2249784
hardship_amount                               2249784
hardship_start_date                           2249784
hardship_end_date                             2249784
payment_plan_start_date                       2249784
hardship_length                               2249784
hardship_dpd                                  2249784
hardship_loan_status                          2249784
orig_projected_additional_accrued_interest    2252050
hardship_payoff_balance_amount                2249784
hardship_last_payment_amount                  2249784
disbursement_method                                33
debt_settlement_flag                               33
debt_settlement_flag_date                     2226455
settlement_status                             2226455
settlement_date                               2226455
settlement_amount                             2226455
settlement_percentage                         2226455
settlement_term                               2226455
dtype: int64
```

In [8]:
```
# Make a copy of df, so that we can apply all the operation on df_copy without modi
df_copy = df.copy()
```

In [9]:
```
# drop all the NaN values, to see is it contains any rows after deleting all NaN va
df_copy.dropna()
```

Out[9]:

id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment

- Since it is returing empty dataframe , we need to consider other preprocessing/ treatment on data to get rid of NaN (Missing) value

In [10]:
```
# compute the number of NaN values for each column
```

```
drop_nan = df_copy.isnull().sum()

# get the column having NaN value more than 30%
drop_nan = drop_nan[drop_nan.values > (len(df_copy) * 0.30)]
```

In [11]:
```
# get the column name
drop_nan.index
```

Out[11]:
```
Index(['member_id', 'desc', 'mths_since_last_delinq', 'mths_since_last_record',
       'next_pymnt_d', 'mths_since_last_major_derog', 'annual_inc_joint',
       'dti_joint', 'verification_status_joint', 'open_acc_6m', 'open_act_il',
       'open_il_12m', 'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il',
       'il_util', 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util',
       'inq_fi', 'total_cu_tl', 'inq_last_12m', 'mths_since_recent_bc_dlq',
       'mths_since_recent_revol_delinq', 'revol_bal_joint',
       'sec_app_fico_range_low', 'sec_app_fico_range_high',
       'sec_app_earliest_cr_line', 'sec_app_inq_last_6mths',
       'sec_app_mort_acc', 'sec_app_open_acc', 'sec_app_revol_util',
       'sec_app_open_act_il', 'sec_app_num_rev_accts',
       'sec_app_chargeoff_within_12_mths',
       'sec_app_collections_12_mths_ex_med',
       'sec_app_mths_since_last_major_derog', 'hardship_type',
       'hardship_reason', 'hardship_status', 'deferral_term',
       'hardship_amount', 'hardship_start_date', 'hardship_end_date',
       'payment_plan_start_date', 'hardship_length', 'hardship_dpd',
       'hardship_loan_status', 'orig_projected_additional_accrued_interest',
       'hardship_payoff_balance_amount', 'hardship_last_payment_amount',
       'debt_settlement_flag_date', 'settlement_status', 'settlement_date',
       'settlement_amount', 'settlement_percentage', 'settlement_term'],
      dtype='object')
```

In [12]:
```
# delete those columns having Missing values more than 30%, because it is not wise
# having most of the values are missing
df_copy.drop(labels= drop_nan.index, inplace = True, axis = 1)
```

In [13]:
```
df_copy.head() # display the top 5 rows of data
```

Out[13]:

	id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	gı
0	68407277	3600.0	3600.0	3600.0	36 months	13.99	123.03	
1	68355089	24700.0	24700.0	24700.0	36 months	11.99	820.28	
2	68341763	20000.0	20000.0	20000.0	60 months	10.78	432.66	
3	66310712	35000.0	35000.0	35000.0	60 months	14.85	829.90	
4	68476807	10400.0	10400.0	10400.0	60 months	22.45	289.91	

In [14]:
```
float_cols = df_copy.select_dtypes('float').columns
```

In [15]:
```
# compute the correlation matrix to know the name of all the columns which are depe
# if two columns (Also know as features or attributes) are dependent it means keepi
# we can reproduce the another feature from 1st one. hence delete the dependent fea
# even if we keep the dependent feature, it will not contribute in improving the ac
# but it will make the program slow because of unnecessary features.

corr_matrix  = df_copy[float_cols].corr().abs()
```

remove dependent features

correlation coeff > 0.98

In [16]:
```
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find features with correlation value greater than 0.98 (i.e. features to be stron
to_drop = [column for column in upper.columns if any(upper[column] > 0.98)]

# Drop features
df_copy.drop(to_drop, axis=1, inplace=True)
```

In [17]:
```
# this is the list of features which is unnecessary (strongly dependent) and we nee
to_drop
```

Out[17]:
```
['funded_amnt',
 'funded_amnt_inv',
 'fico_range_high',
 'out_prncp_inv',
 'total_pymnt_inv',
 'num_rev_tl_bal_gt_0',
 'num_sats']
```

In [18]:
```
df_copy.head()
```

Out[18]:

	id	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	em
0	68407277	3600.0	36 months	13.99	123.03	C	C4	leadman	
1	68355089	24700.0	36 months	11.99	820.28	C	C1	Engineer	
2	68341763	20000.0	60 months	10.78	432.66	B	B4	truck driver	
3	66310712	35000.0	60 months	14.85	829.90	C	C5	Information Systems Officer	
4	68476807	10400.0	60 months	22.45	289.91	F	F1	Contract Specialist	

In [19]:
```python
# based on preliminary observation, these are the extra useless columns which will
Col_drop = ['id','emp_title', 'issue_d', 'pymnt_plan', 'url', 'title', 'zip_code',
            'earliest_cr_line', 'initial_list_status', 'out_prncp', 'total_pymnt',
            'last_credit_pull_d', 'last_fico_range_high', 'last_fico_range_low', 'p
            'disbursement_method', 'debt_settlement_flag']
```

In [20]:
```python
# drop these features as well
df_copy.drop(columns =Col_drop, inplace = True )
```

In [21]:
```python
# check the shape
df_copy.shape
```

Out[21]:
```
(2260701, 65)
```

In [22]:
```python
# get the loan status and their respective count
df_copy['loan_status'].value_counts()
```

Out[22]:
```
Fully Paid                                             1076751
Current                                                 878317
Charged Off                                             268559
Late (31-120 days)                                       21467
In Grace Period                                           8436
Late (16-30 days)                                         4349
Does not meet the credit policy. Status:Fully Paid        1988
Does not meet the credit policy. Status:Charged Off        761
Default                                                     40
Name: loan_status, dtype: int64
```

- we consider the loan with Fully paid or charged off and ignore all the remaining loans
- Also consider 'Does not meet the credit policy. Status:Fully Paid' as Fully Paid
- and 'Does not meet the credit policy. Status:Charged Off' as Charged Off

In [23]:
```python
# replace Does not meet the credit policy. Status:Charged Off' as charged off
# and Does not meet the credit policy. Status:Fully Paid' as Fully Paid

df_copy['loan_status'].replace(['Does not meet the credit policy. Status:Fully Paid
```

In [24]:
```python
df_copy['loan_status'].value_counts()
```

Out[24]:
```
Fully Paid            1078739
Current                878317
Charged Off            269320
Late (31-120 days)      21467
In Grace Period          8436
Late (16-30 days)        4349
Default                    40
Name: loan_status, dtype: int64
```

In [25]:
```python
# Now consider only Fully paid and charged off only
df_copy = df_copy[(df_copy['loan_status']=='Fully Paid') | (df_copy['loan_status']=
```

In [26]:
```python
df_copy['loan_status'].value_counts()
```

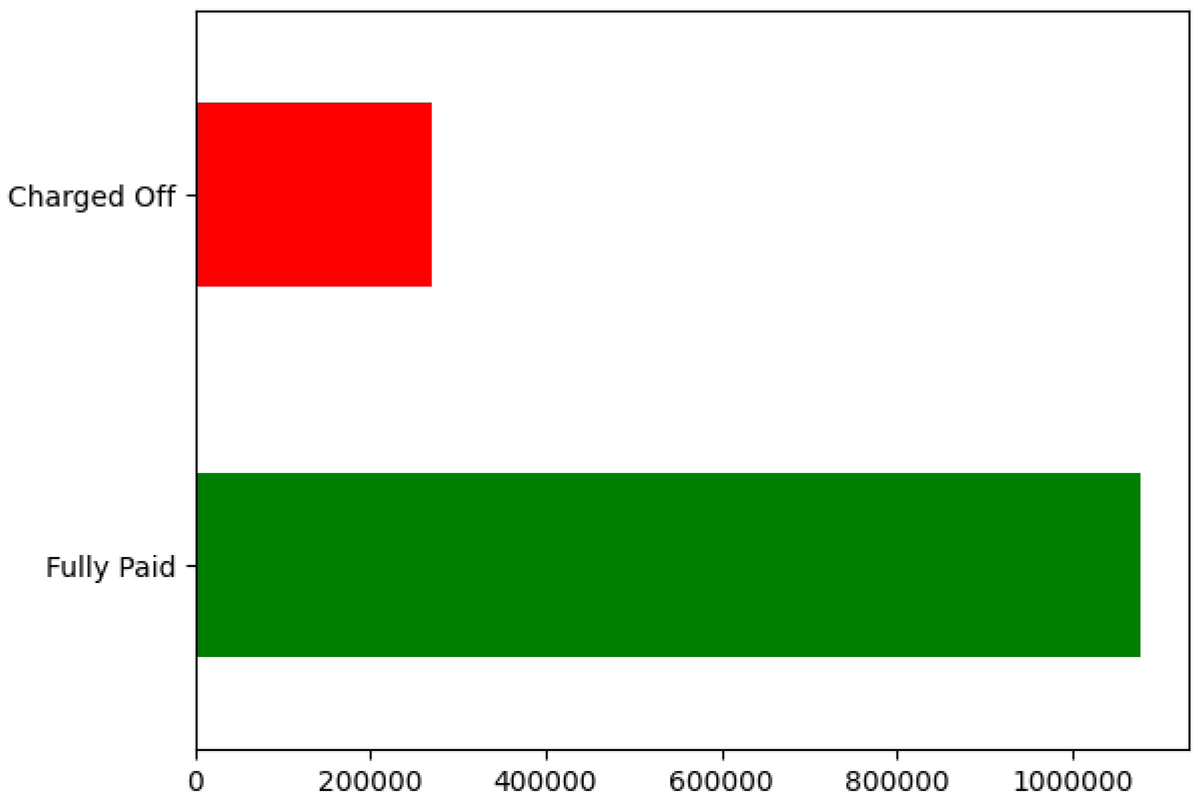
Out[26]:
```
Fully Paid     1078739
Charged Off     269320
Name: loan_status, dtype: int64
```

In [27]:
```
df_copy['loan_status'].value_counts(normalize=True)*100
```

Out[27]:
```
Fully Paid     80.021646
Charged Off    19.978354
Name: loan_status, dtype: float64
```

In [28]:
```
# visualize on the bar plot the count of 'fully paid' and 'Default'
plt.ticklabel_format(style='plain')
t = pd.value_counts(df_copy['loan_status'].values, sort=True)
t.plot.barh(color=['g','r'])
print(df_copy['loan_status'].value_counts(normalize=True)*100)
plt.show()
```

```
Fully Paid     80.021646
Charged Off    19.978354
Name: loan_status, dtype: float64
```



In [29]:
```
# again check for NaN values
df_copy.isnull().sum()
```

```
Out[29]: loan_amnt                          0
         term                               0
         int_rate                           0
         installment                        0
         grade                              0
         sub_grade                          0
         emp_length                     78545
         home_ownership                     0
         annual_inc                         4
         verification_status                0
         loan_status                        0
         purpose                            0
         dti                              374
         delinq_2yrs                       29
         fico_range_low                     0
         inq_last_6mths                    30
         open_acc                          29
         pub_rec                           29
         revol_bal                          0
         revol_util                       897
         total_acc                         29
         total_rec_prncp                    0
         total_rec_int                      0
         total_rec_late_fee                 0
         recoveries                         0
         collection_recovery_fee            0
         collections_12_mths_ex_med       145
         application_type                   0
         acc_now_delinq                    29
         tot_coll_amt                   70276
         tot_cur_bal                    70276
         total_rev_hi_lim               70276
         acc_open_past_24mths           50030
         avg_cur_bal                    70298
         bc_open_to_buy                 63892
         bc_util                        64661
         chargeoff_within_12_mths         145
         delinq_amnt                       29
         mo_sin_old_il_acct            108324
         mo_sin_old_rev_tl_op           70277
         mo_sin_rcnt_rev_tl_op          70277
         mo_sin_rcnt_tl                 70276
         mort_acc                       50030
         mths_since_recent_bc           62970
         mths_since_recent_inq         176820
         num_accts_ever_120_pd          70276
         num_actv_bc_tl                 70276
         num_actv_rev_tl                70276
         num_bc_sats                    58590
         num_bc_tl                      70276
         num_il_tl                      70276
         num_op_rev_tl                  70276
         num_rev_accts                  70277
         num_tl_120dpd_2m              120150
         num_tl_30dpd                   70276
         num_tl_90g_dpd_24m             70276
```

```
num_tl_op_past_12m            70276
pct_tl_nvr_dlq                70430
percent_bc_gt_75              64304
pub_rec_bankruptcies           1365
tax_liens                       105
tot_hi_cred_lim               70276
total_bal_ex_mort             50030
total_bc_limit                50030
total_il_high_credit_limit    70276
dtype: int64
```

Hanlde NaN Value

In [30]:

```
for col in df_copy.columns:

    # replace float attributes with their median value
    if isinstance(df_copy[col][0], float):
        df_copy[col].fillna(df_copy[col].median(), inplace = True)

        # replace other attribute with their mode value
    else:
        df_copy[col].fillna(df_copy[col].mode()[0], inplace = True)
```

In [31]:

```
# again check for NaN values
df_copy.isnull().sum()
```

```
Out[31]: loan_amnt                     0
         term                          0
         int_rate                      0
         installment                   0
         grade                         0
         sub_grade                     0
         emp_length                    0
         home_ownership                0
         annual_inc                    0
         verification_status           0
         loan_status                   0
         purpose                       0
         dti                           0
         delinq_2yrs                   0
         fico_range_low                0
         inq_last_6mths                0
         open_acc                      0
         pub_rec                       0
         revol_bal                     0
         revol_util                    0
         total_acc                     0
         total_rec_prncp               0
         total_rec_int                 0
         total_rec_late_fee            0
         recoveries                    0
         collection_recovery_fee       0
         collections_12_mths_ex_med    0
         application_type              0
         acc_now_delinq                0
         tot_coll_amt                  0
         tot_cur_bal                   0
         total_rev_hi_lim              0
         acc_open_past_24mths          0
         avg_cur_bal                   0
         bc_open_to_buy                0
         bc_util                       0
         chargeoff_within_12_mths      0
         delinq_amnt                   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_recent_bc          0
         mths_since_recent_inq         0
         num_accts_ever_120_pd         0
         num_actv_bc_tl                0
         num_actv_rev_tl               0
         num_bc_sats                   0
         num_bc_tl                     0
         num_il_tl                     0
         num_op_rev_tl                 0
         num_rev_accts                 0
         num_tl_120dpd_2m              0
         num_tl_30dpd                  0
         num_tl_90g_dpd_24m            0
```

```
num_tl_op_past_12m            0
pct_tl_nvr_dlq                0
percent_bc_gt_75              0
pub_rec_bankruptcies          0
tax_liens                     0
tot_hi_cred_lim               0
total_bal_ex_mort             0
total_bc_limit                0
total_il_high_credit_limit    0
dtype: int64
```

- Now there is no NaN Values. i.e. Missing values are handled properly

In [32]:
```
df_copy.dtypes
```

```
Out[32]: loan_amnt                     float64
         term                           object
         int_rate                      float64
         installment                   float64
         grade                          object
         sub_grade                      object
         emp_length                     object
         home_ownership                 object
         annual_inc                    float64
         verification_status            object
         loan_status                    object
         purpose                        object
         dti                           float64
         delinq_2yrs                   float64
         fico_range_low                float64
         inq_last_6mths                float64
         open_acc                      float64
         pub_rec                       float64
         revol_bal                     float64
         revol_util                    float64
         total_acc                     float64
         total_rec_prncp               float64
         total_rec_int                 float64
         total_rec_late_fee            float64
         recoveries                    float64
         collection_recovery_fee       float64
         collections_12_mths_ex_med    float64
         application_type               object
         acc_now_delinq                float64
         tot_coll_amt                  float64
         tot_cur_bal                   float64
         total_rev_hi_lim              float64
         acc_open_past_24mths          float64
         avg_cur_bal                   float64
         bc_open_to_buy                float64
         bc_util                       float64
         chargeoff_within_12_mths      float64
         delinq_amnt                   float64
         mo_sin_old_il_acct            float64
         mo_sin_old_rev_tl_op          float64
         mo_sin_rcnt_rev_tl_op         float64
         mo_sin_rcnt_tl                float64
         mort_acc                      float64
         mths_since_recent_bc          float64
         mths_since_recent_inq         float64
         num_accts_ever_120_pd         float64
         num_actv_bc_tl                float64
         num_actv_rev_tl               float64
         num_bc_sats                   float64
         num_bc_tl                     float64
         num_il_tl                     float64
         num_op_rev_tl                 float64
         num_rev_accts                 float64
         num_tl_120dpd_2m              float64
         num_tl_30dpd                  float64
         num_tl_90g_dpd_24m            float64
```

```
num_tl_op_past_12m            float64
pct_tl_nvr_dlq                float64
percent_bc_gt_75              float64
pub_rec_bankruptcies          float64
tax_liens                     float64
tot_hi_cred_lim               float64
total_bal_ex_mort             float64
total_bc_limit                float64
total_il_high_credit_limit    float64
dtype: object
```

categorical features

In [33]:

```
# get the categorical featues
category_column = df_copy.dtypes.index[df_copy.dtypes=='object']
```

In [34]:

```
# print categorical features and the count of their unique values.
for i in category_column:
    print(i)
    print(df_copy[i].value_counts())
    print(20*'-')
```

```
term
 36 months    1023181
 60 months     324878
Name: term, dtype: int64
--------------------
grade
B    393095
C    382315
A    235188
D    201644
E     94186
F     32305
G      9326
Name: grade, dtype: int64
--------------------
sub_grade
C1    85616
B4    83275
B5    82636
B3    81900
C2    79356
C3    75127
C4    74553
B2    74079
B1    71205
C5    67663
A5    64054
A4    52254
D1    51443
D2    44981
A1    43681
D3    39461
A3    38009
A2    37190
D4    35720
D5    30039
E1    23865
E2    21509
E3    18499
E4    15817
E5    14496
F1    10033
F2     7255
F3     6137
F4     4901
F5     3979
G1     3033
G2     2160
G3     1644
G4     1323
G5     1166
Name: sub_grade, dtype: int64
--------------------
emp_length
10+ years    521214
2 years      122092
```

```
< 1 year     108533
3 years      107863
1 year        88842
5 years       84326
4 years       80761
6 years       62877
8 years       60808
7 years       59724
9 years       51019
Name: emp_length, dtype: int64
--------------------
home_ownership
MORTGAGE    666835
RENT        535684
OWN         145019
ANY            286
OTHER          182
NONE            53
Name: home_ownership, dtype: int64
--------------------
verification_status
Source Verified    521563
Verified           418963
Not Verified       407533
Name: verification_status, dtype: int64
--------------------
loan_status
Fully Paid     1078739
Charged Off     269320
Name: loan_status, dtype: int64
--------------------
purpose
debt_consolidation    781421
credit_card           295619
home_improvement       87718
other                  78299
major_purchase         29548
medical                15612
small_business         15577
car                    14649
moving                  9526
vacation                9084
house                   7297
wedding                 2350
renewable_energy         936
educational              423
Name: purpose, dtype: int64
--------------------
application_type
Individual    1322259
Joint App       25800
Name: application_type, dtype: int64
--------------------
```

In [35]:

```
# there are lot of classification for sub_grade, hence delete it. Also its sub feat
df_copy.drop('sub_grade', axis=1, inplace=True)
```

In [36]:
```
# Convert categorical features to neumerical values
df_copy['term'].replace((' 36 months', ' 60 months'),(36 ,60), inplace = True)
df_copy['grade'].replace(('A','B','C','D','E','F','G'),(1,2,3,4,5,6,7), inplace = T
df_copy['emp_length'].replace(('10+ years','2 years','< 1 year','3 years','1 year',
df_copy['home_ownership'].replace(('MORTGAGE', 'RENT','OWN','ANY', 'OTHER','NONE'),
df_copy['verification_status'].replace(('Source Verified', 'Verified','Not Verified
df_copy['loan_status'].replace(('Fully Paid', 'Charged Off'),(0,1), inplace = True)
df_copy['purpose'].replace(('debt_consolidation', 'credit_card','home_improvement',
df_copy['application_type'].replace(('Individual','Joint App'),(1,2), inplace  = Tr
```

In [37]:
```
df_copy.shape
```

Out[37]:
```
(1348059, 64)
```

Imbalance data

In [38]:
```
# percentage of paid /unpaid
df_copy['loan_status'].value_counts(normalize=True)*100
```

Out[38]:
```
0    80.021646
1    19.978354
Name: loan_status, dtype: float64
```

In [39]:
```
# training and test set splitting

from sklearn.model_selection import train_test_split

# get x and y
x = df_copy.drop(columns='loan_status',axis=1)
y = df_copy['loan_status']

# feature scaling to bring the features into same range
scaler = StandardScaler()
scaler_data = scaler.fit_transform(x)

# split the data. 70% for training and 30% for testing
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.30, shuffle =
```

Baseline Models

In [40]:
```
from sklearn.dummy import DummyClassifier
# DummyClassifier to predict only target 0
dummy = DummyClassifier(strategy='most_frequent').fit(x_train, y_train)
dummy_pred = dummy.predict(x_test)

# checking unique labels
print('Unique predicted labels: ', (np.unique(dummy_pred)))

# checking accuracy
print('Test score: ', accuracy_score(y_test, dummy_pred))
```

```
Unique predicted labels:  [0]
Test score:  0.799848176886291
```

In [41]:
```
roc_auc_score(y_test, dummy_pred)
```

Out[41]:
```
0.5
```

- 0 - Fully Paid
- 1 - Charged Off

As predicted our accuracy score for classifying all Loan as Fully Paid is 80.079%!

As the Dummy Classifier predicts only single class (i.e. Fully Paid), it is clearly not a good option for our objective of correctly classifying.

Let's see how logistic regression performs on this dataset.

Logistic Regression

In [42]:
```
# build the model
lr_model = LogisticRegression(random_state= 42)

# fit the model on training data
lr_model.fit(x_train,y_train)

# make prediction on test data
y_pred = lr_model.predict(x_test)
```

In [43]:
```
# get the accuracy
accuracy_score(y_test, y_pred)
```

Out[43]:
```
0.9942856153781483
```

In [44]:
```
# Checking unique values
predictions = pd.DataFrame(y_pred)
predictions[0].value_counts()
```

Out[44]:
```
0    324848
1     79570
Name: 0, dtype: int64
```

Logistic Regression outperformed the Dummy Classifier! We can see that it predicted 79K approx instances of class 1 (i.e. charged off), so this is definitely an improvement. But can we do better?

Let's see if we can apply some techniques for dealing with class imbalance to improve these results.

Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be misleading. Metrics that can provide better insight include:

- Confusion Matrix: a table showing correct predictions and types of incorrect predictions.
- Precision: the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- Recall: the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- F1: Score: the weighted average of precision and recall.
- Since our main objective with the dataset is to prioritize accuraltely classifying loan status, the recall score can be considered our main metric to use for evaluating outcomes.

objective:- gain high recall, high precision

In [45]:
```
recall_score(y_test, y_pred)
```

Out[45]:
```
0.9772314534560504
```

In [46]:
```
precision_score(y_test, y_pred)
```

Out[46]:
```
0.994118386326505
```

In [47]:
```
roc_auc_score(y_test, y_pred)
```

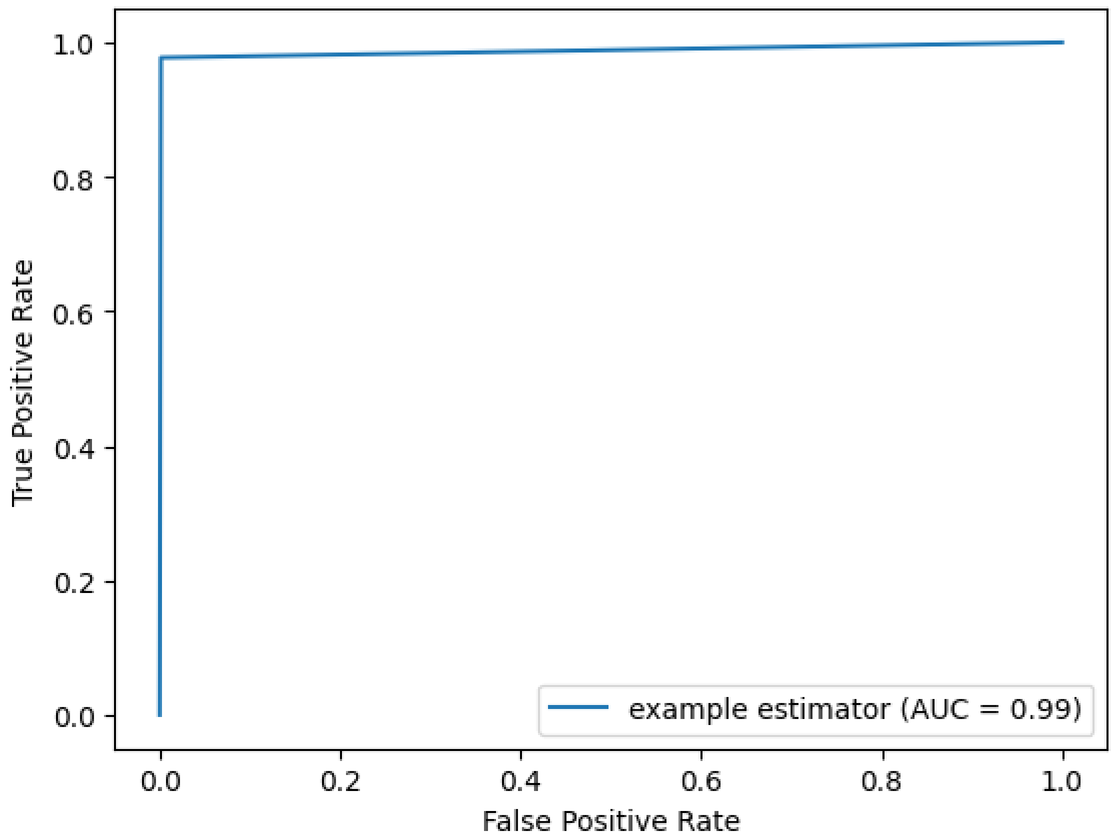
Out[47]:
```
0.9878923278662964
```

In [48]:
```
pd.DataFrame(confusion_matrix(y_test, y_pred),
             columns=['true_fully_paid','true_charged_off'],
             index=['predict_fully_paid','predict_charged_off'])
```

Out[48]:

	true_fully_paid	true_charged_off
predict_fully_paid	323005	468
predict_charged_off	1843	79102

In [49]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
                                  estimator_name='example estimator')
display.plot()
plt.show()
```



We have a very high accuracy score of 0.99 And from the confusion matrix, we can see we are misclassifying several observations leading to a recall score of 0.97 only.

Lets try other method

ref:-
https://www.researchgate.net/publication/341164819_Machine_Learning_with_Oversampling_anc
_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ

1. Oversampling Minority Class

In [50]:
```
X = pd.concat([x_train, y_train], axis=1)
```

In [51]:
```
X.head()
```

Out[51]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	ar
161502	3600.0	36	14.65	124.18	3	1.0	2	
2099214	15300.0	60	15.05	364.39	3	8.0	2	
259562	13000.0	36	12.69	436.09	3	10.0	1	
2178453	8000.0	36	12.74	268.56	3	1.0	2	
221246	8400.0	36	12.29	280.17	3	6.0	2	

In [52]:
```
fully_paid = X[X['loan_status']==0]
charged_off =  X[X['loan_status']==1]
```

In [53]:
```
len(fully_paid) # majority
```

Out[53]: 755266

In [54]:
```
len(charged_off) # minority
```

Out[54]: 188375

In [55]:
```
from sklearn.utils import resample
# upsample minority
charged_off_upsampled = resample(charged_off,
                          replace=True, # sample with replacement
                          n_samples=len(fully_paid), # match number in majority cla
                          random_state=42) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([fully_paid, charged_off_upsampled])

# check new class counts
upsampled['loan_status'].value_counts()
```

Out[55]:
```
0    755266
1    755266
Name: loan_status, dtype: int64
```

In [56]:
```
# trying logistic regression again with the balanced dataset
y_train_upsampled = upsampled['loan_status']
X_train_upsampled = upsampled.drop('loan_status', axis=1)

upsampled = LogisticRegression(random_state= 42).fit(X_train_upsampled, y_train_ups

upsampled_pred = upsampled.predict(x_test)
# Checking accuracy
accuracy_score(y_test, upsampled_pred)
```

Out[56]: 0.9961277687936738

In [57]:
```
recall_score(y_test, upsampled_pred)
```

Out[57]:
```
0.9880412625857063
```

In [58]:
```
precision_score(y_test, upsampled_pred)
```

Out[58]:
```
0.992578343158548
```

In [59]:
```
roc_auc_score(y_test, upsampled_pred)
```

Out[59]:
```
0.9930962883028666
```

In [60]:
```
f1_score(y_test, upsampled_pred)
```

Out[60]:
```
0.9903046062407133
```

Our accuracy score increased a little bit after upsampling, the model is now predicting both classes more equally, making it an improvement over our plain logistic regression above.

2 .Undersampling Majority Class

Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback to undersampling is that we are removing information that may be valuable.

In [61]:
```
len(charged_off)
```

Out[61]:
```
188375
```

In [62]:
```
# downsample majority
fully_paid_downsampled = resample(fully_paid,
                                replace = False, # sample without replacement
                                n_samples = len(charged_off), # match minority n
                                random_state = 42) # reproducible results

# combine minority and downsampled majority
downsampled = pd.concat([fully_paid_downsampled, charged_off])

# checking counts
downsampled['loan_status'].value_counts()
```

Out[62]:
```
0    188375
1    188375
Name: loan_status, dtype: int64
```

In [63]:
```
# trying logistic regression again with the balanced dataset
y_train_downsampled = downsampled['loan_status']
X_train_downsampled = downsampled.drop('loan_status', axis=1)

downsampled_lr = LogisticRegression(random_state= 42).fit(X_train_downsampled, y_tr
```

```
downsampled_pred = downsampled_lr.predict(x_test)
# Checking accuracy
accuracy_score(y_test, downsampled_pred)
```

Out[63]: 0.9963997645999931

In [64]:
```
recall_score(y_test, downsampled_pred)
```

Out[64]: 0.9887701525727346

In [65]:
```
roc_auc_score(y_test, downsampled_pred)
```

Out[65]: 0.9935395652236202

In [66]:
```
print('***********Default Logistic Regression************roc_auc_score:',roc_auc_s
print(classification_report(y_test, y_pred))

print('*********** Logistic Regression- upsampled************roc_auc_score:',roc_a
print(classification_report(y_test, upsampled_pred))

print('*********** Logistic Regression- downsampled************roc_auc_score:',roc
print(classification_report(y_test, downsampled_pred))
```

```
***********Default Logistic Regression************roc_auc_score: 0.9878923278662964
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    323473
           1       0.99      0.98      0.99     80945

    accuracy                           0.99    404418
   macro avg       0.99      0.99      0.99    404418
weighted avg       0.99      0.99      0.99    404418

*********** Logistic Regression- upsampled************roc_auc_score: 0.993096288302
8666
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    323473
           1       0.99      0.99      0.99     80945

    accuracy                           1.00    404418
   macro avg       0.99      0.99      0.99    404418
weighted avg       1.00      1.00      1.00    404418

*********** Logistic Regression- downsampled************roc_auc_score: 0.9935395652
236202
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    323473
           1       0.99      0.99      0.99     80945

    accuracy                           1.00    404418
   macro avg       1.00      0.99      0.99    404418
weighted avg       1.00      1.00      1.00    404418
```

In [67]:

```
cm = confusion_matrix(y_test, y_pred,normalize='all')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Fully Paid','Cha
disp.plot()
plt.show()
```

In [68]:

```
# downsampled
cm = confusion_matrix(y_test, downsampled_pred,normalize='all')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Fully Paid','Cha
disp.plot()
plt.show()
```

In [69]:

```
# upsampled
cm = confusion_matrix(y_test, upsampled_pred,normalize='all')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Fully Paid','Cha
disp.plot()
plt.show()
```

In [70]:
```
result = pd.DataFrame([],columns=['Model','Accuracy','F1','Recall','precision'])
def get_result(true,pred,name):
    result.loc[len(result)] = [name,accuracy_score(true,pred),f1_score(true,pred),r
```

In [71]:
```
get_result(y_test,dummy_pred,'DummyClassifier')
get_result(y_test,y_pred,'LogisticRegression')
get_result(y_test,upsampled_pred,'LogisticRegression+Upsampling')
get_result(y_test,downsampled_pred,'LogisticRegression+Downsampling')
```

Decision Tree and Random Forest

In [72]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

dt_clf = DecisionTreeClassifier(random_state=42)
rf_clf = RandomForestClassifier(random_state=42,n_jobs=-1)

dt_clf.fit(x_train,y_train)
dt_y_pred = dt_clf.predict(x_test)
get_result(y_test,dt_y_pred,'DecisionTree')

rf_clf.fit(x_train,y_train)
rf_y_pred = rf_clf.predict(x_test)
get_result(y_test,rf_y_pred,'Random Forest')
```

In [73]:
```
result
```

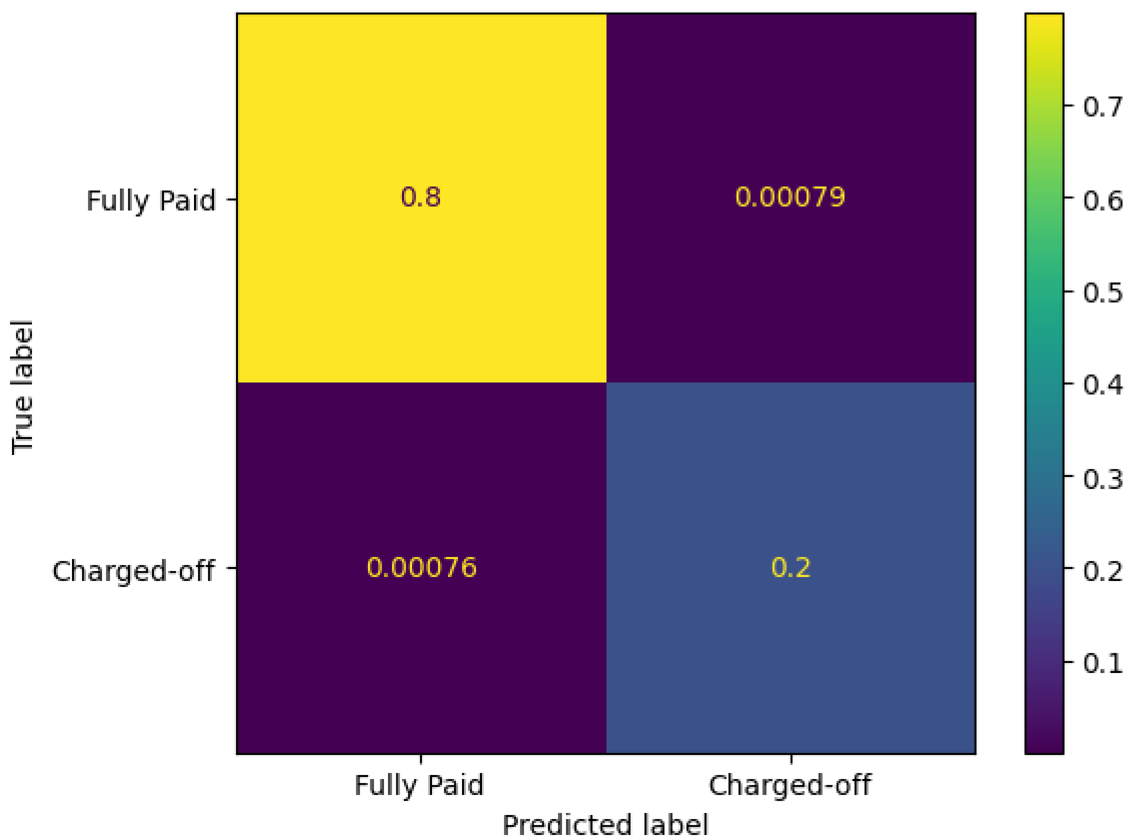
Out[73]:

	Model	Accuracy	F1	Recall	precision
0	DummyClassifier	0.799848	0.000000	0.000000	0.000000
1	LogisticRegression	0.994286	0.985603	0.977231	0.994118
2	LogisticRegression+Upsampling	0.996128	0.990305	0.988041	0.992578
3	LogisticRegression+Downsampling	0.996400	0.990986	0.988770	0.993212
4	DecisionTree	0.998452	0.996133	0.996207	0.996060
5	Random Forest	0.992770	0.981606	0.963877	1.000000

1. **How to predict if the lender will fully pay the money:** Using our DecisionTree Model which is having 99.84% accuracy and 0.9962 as recall and 0.996 as precision
2. **How to decrease the risk of charged off:** first compute the loan payment prediction using our Model, if the model is saying that it will get paid fully, then only give the loan

In [76]:

```
cm = confusion_matrix(y_test, dt_y_pred,normalize='all')
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Fully Paid','Cha
disp.plot()
plt.show()
```



Based on values of confusion matrix,accuracy,precision and recall,among all varients of models used, DecisionTree is the best

```
In [ ]:
```