```
     ...:
     ...: @author: beileitang
     ...: """
     ...:
     ...: import pandas as pd
     ...: import numpy as np           # for math calculation
     ...: import matplotlib            # for plotting graphs
     ...: import matplotlib.pyplot as plt # for plotting graphs
     ...: import seaborn as sns           # for data
visualization
     ...: import warnings                        # To ignore any
warnings
     ...: warnings.filterwarnings("ignore")
     ...:
     ...: file="~/Desktop/loan data/new_loan_data.csv"
     ...: data=pd.read_csv(file)
     ...: data.describe().transpose()
Out[112]:
                     count          mean     ...           75%
max
loan_amnt           5839.0  15081.867614     ...      20000.00
40000.00
int_rate            5839.0     14.412286     ...         17.97
30.99
installment         5839.0    452.934025     ...        620.11
1489.07
annual_inc          5839.0  80372.299800     ...      98000.00
1045000.00
dti                 5825.0     18.692050     ...         23.51
654.77
delinq_2yrs         5839.0      0.253811     ...          0.00
12.00
loan_default        5839.0      0.180853     ...          0.00
1.00
house_onwer         5839.0      0.133413     ...          0.00
1.00
emp_year_greater_5  5839.0      0.770680     ...          1.00
1.00

[9 rows x 8 columns]

In [113]: data.isnull().sum()
     ...: data['dti'].fillna(data['dti'].median(), inplace=True)
     ...: data.isnull().sum()
     ...: data.drop(["loan_status"],axis=1, inplace= True)
```

```
In [114]: obj_df = data.select_dtypes(include=['object']).copy()
     ...: obj_df.tail()
     ...: obj_df.isnull().sum()
Out[114]:
term       0
grade      0
purpose    0
dtype: int64

In [115]: from sklearn.preprocessing import OneHotEncoder
     ...: from sklearn.preprocessing import LabelEncoder
     ...:
     ...: label= LabelEncoder()
     ...:
     ...: obj_df['term']=label.fit_transform(obj_df['term'])
     ...: print(obj_df['term'])
     ...: term=obj_df['term']
     ...:
     ...:
obj_df['purpose']=label.fit_transform(obj_df['purpose'])
     ...: print(obj_df['purpose'])
     ...: purpose=obj_df['purpose']
     ...:
     ...: obj_df['grade']=label.fit_transform(obj_df['grade'])
     ...: print(obj_df['grade'])
     ...: grade=obj_df['grade']
0       0
1       0
2       0
3       0
4       0
5       0
6       1
7       0
8       0
9       0
10      0
11      1
12      1
13      0
14      0
15      0
16      0
17      1
18      1
19      0
```

```
20      0
21      0
22      0
23      0
24      1
25      0
26      1
27      0
28      1
29      0
       ..
5809    0
5810    1
5811    0
5812    1
5813    1
5814    0
5815    0
5816    0
5817    0
5818    0
5819    0
5820    0
5821    0
5822    0
5823    0
5824    0
5825    0
5826    0
5827    0
5828    0
5829    1
5830    0
5831    0
5832    0
5833    0
5834    0
5835    0
5836    0
5837    0
5838    0
Name: term, Length: 5839, dtype: int64
0       11
1        1
2        2
3        2
```

```
4        4
5        2
6        1
7        2
8        1
9        2
10       1
11       2
12       2
13       2
14       2
15       5
16       3
17       2
18       2
19       2
20       5
21      10
22       1
23      11
24       2
25       3
26       2
27       2
28       2
29       2
        ..
5809     1
5810     2
5811     8
5812     2
5813     1
5814     3
5815     3
5816     1
5817     1
5818     1
5819     2
5820     2
5821     5
5822     2
5823     2
5824     4
5825     1
5826     1
5827     1
```

```
5828     1
5829     8
5830     1
5831     2
5832     2
5833     2
5834     2
5835     8
5836     2
5837     2
5838     1
Name: purpose, Length: 5839, dtype: int64
0       2
1       0
2       0
3       2
4       3
5       1
6       1
7       0
8       2
9       1
10      0
11      1
12      3
13      0
14      3
15      1
16      2
17      1
18      2
19      1
20      1
21      3
22      0
23      1
24      4
25      0
26      3
27      1
28      2
29      0
       ..
5809    1
5810    3
5811    1
```

```
5812    2
5813    0
5814    1
5815    0
5816    1
5817    0
5818    0
5819    3
5820    2
5821    2
5822    2
5823    1
5824    3
5825    2
5826    2
5827    0
5828    4
5829    4
5830    2
5831    2
5832    2
5833    3
5834    0
5835    2
5836    2
5837    4
5838    1
Name: grade, Length: 5839, dtype: int64

In [116]: onehotencoder = OneHotEncoder()
     ...:
     ...: ## convge to muerical
     ...: term =
onehotencoder.fit_transform(obj_df.term.values.reshape(-1,1)).toa
rray()
     ...: dfOneHot_term = pd.DataFrame(term, columns =
["loan_term"+str(int(i)) for i in range(term.shape[1])]) #term 0
is 36 and term 1 is 60
     ...:
     ...: purpose =
onehotencoder.fit_transform(obj_df.purpose.values.reshape(-1,1)).
toarray()
     ...: dfOneHot_purpose = pd.DataFrame(purpose, columns =
["loam_purpose"+str(int(i)) for i in range(purpose.shape[1])])
#term 0 is 36 and term 1 is 60
     ...:
```

```
     ...: #int_rate =
onehotencoder.fit_transform(obj_df.int_rate.values.reshape(-1,1))
.toarray()
     ...: #dfOneHot_int = pd.DataFrame(int_rate, columns =
["loan_int_rate"+str(int(i)) for i in range(int_rate.shape[1])])
     ...:
     ...: grade =
onehotencoder.fit_transform(obj_df.grade.values.reshape(-1,1)).to
array()
     ...: dfOneHot_grade = pd.DataFrame(grade, columns =
["loan_grade"+str(int(i)) for i in range(grade.shape[1])])
     ...:
     ...:
     ...: data=pd.concat([data, dfOneHot_grade], axis=1)
     ...: #data=pd.concat([data, dfOneHot_int], axis=1)
     ...: data=pd.concat([data, dfOneHot_purpose], axis=1)
     ...: data=pd.concat([data, dfOneHot_term], axis=1)

In [117]: data.isnull().sum()
Out[117]:
loan_amnt             0
term                  0
int_rate              0
installment           0
grade                 0
annual_inc            0
purpose               0
dti                   0
delinq_2yrs           0
loan_default          0
house_onwer           0
emp_year_greater_5    0
loan_grade0           0
loan_grade1           0
loan_grade2           0
loan_grade3           0
loan_grade4           0
loan_grade5           0
loan_grade6           0
loam_purpose0         0
loam_purpose1         0
loam_purpose2         0
loam_purpose3         0
loam_purpose4         0
loam_purpose5         0
loam_purpose6         0
```

```
loam_purpose7         0
loam_purpose8         0
loam_purpose9         0
loam_purpose10        0
loam_purpose11        0
loan_term0            0
loan_term1            0
dtype: int64

In [118]: data.drop(['grade'],axis=1,inplace=True)
     ...: data.drop(['purpose'],axis=1,inplace=True)
     ...: #data.drop(['int_rate'],axis=1,inplace=True)
     ...: data.drop(["term"],axis=1, inplace= True)
     ...:
     ...: data.drop(["loam_purpose11"],axis=1, inplace= True)
     ...: data.drop(["loan_term1"],axis=1, inplace= True)
     ...: data.drop(["loan_grade6"],axis=1, inplace= True)

In [119]: from sklearn.model_selection import train_test_split
     ...: y = data.loan_default # define the target variable
(dependent variable) as y
     ...: # create training and testing vars
     ...: X_train, X_test, y_train, y_test =
train_test_split(data, y, test_size=0.2)
     ...: print (X_train.shape, y_train.shape)
     ...: print (X_test.shape, y_test.shape)
(4671, 27) (4671,)
(1168, 27) (1168,)

In [120]: from sklearn.preprocessing import StandardScaler
     ...: from sklearn.metrics import classification_report
     ...: from sklearn.linear_model import LogisticRegression
     ...: from sklearn.svm import SVC
     ...: from sklearn import datasets
     ...: from sklearn import preprocessing
     ...: from sklearn.metrics import accuracy_score

In [121]: scaler = StandardScaler()
     ...: Xtrain_std = scaler.fit_transform(X_train)
     ...: Xtest_std = scaler.fit_transform(X_test)
     ...: # normalization
     ...: X = preprocessing.normalize(X_train, norm='l2')

In [122]: svc_rbf = SVC(kernel='rbf', class_weight='balanced',
C=10.0, random_state=0)
     ...: model = svc_rbf.fit(Xtrain_std, y_train)
```

```
     ...: # Create support vector classifier
     ...: svc = SVC(kernel='linear', class_weight='balanced',
C=10.0, random_state=0)
     ...: model = svc.fit(Xtrain_std, y_train)
     ...:
     ...: logmodel = LogisticRegression()
     ...: logmodel.fit(Xtrain_std,y_train) # traning the model by
logistic
Out[122]:
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None,
solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

In [123]: predictions = logmodel.predict(X_test) # input new data
to predict the result
     ...: print(classification_report(y_test,predictions)) #
compare the predicted and y_test
     ...: print("accuracy_score")
     ...: print( accuracy_score(predictions, y_test) )
              precision    recall  f1-score   support

           0       0.85      0.49      0.62       951
           1       0.21      0.61      0.32       217

   micro avg       0.51      0.51      0.51      1168
   macro avg       0.53      0.55      0.47      1168
weighted avg       0.73      0.51      0.56      1168

accuracy_score
0.5128424657534246

In [124]: svm_prediction=svc.predict(X_test)
     ...: print(classification_report(y_test,svm_prediction))
     ...: print("accuracy_score")
     ...: print( accuracy_score(svm_prediction, y_test) )
              precision    recall  f1-score   support

           0       0.92      0.70      0.79       951
           1       0.36      0.74      0.48       217

   micro avg       0.71      0.71      0.71      1168
   macro avg       0.64      0.72      0.64      1168
weighted avg       0.82      0.71      0.74      1168
```

```
accuracy_score
0.7054794520547946

In [125]: svm_prediction_rbf=svc_rbf.predict(X_test)
     ...: print(classification_report(y_test,svm_prediction_rbf))
     ...: print("accuracy_score")
     ...: print( accuracy_score(svm_prediction_rbf, y_test) )
              precision    recall  f1-score   support

           0       0.81      1.00      0.90       951
           1       0.00      0.00      0.00       217

   micro avg       0.81      0.81      0.81      1168
   macro avg       0.41      0.50      0.45      1168
weighted avg       0.66      0.81      0.73      1168

accuracy_score
0.8142123287671232

In [126]: from sklearn.utils import resample
     ...: df_majority = data[data.loan_default==0]
     ...: df_minority = data[data.loan_default==1]
     ...:
     ...: # Downsample majority class
     ...: df_majority_downsampled = resample(df_majority,
     ...:                                    replace=False,    #
sample without replacement
     ...:                                    n_samples=1056,     #
to match minority class
     ...:                                    random_state=123) #
reproducible results
     ...:
     ...: # Combine minority class with downsampled majority
class
     ...: df_downsampled  = pd.concat([df_majority_downsampled,
df_minority])
     ...:
     ...: # Display new class counts
     ...: df_downsampled.loan_default.value_counts()
     ...: # 1    49
     ...: # 0    49
     ...: # Name: balance, dtype: int64
Out[126]:
1    1056
0    1056
```

```
Name: loan_default, dtype: int64

In [127]: y = df_downsampled.loan_default # define the target
variable (dependent variable) as y
     ...: # create training and testing vars
     ...: X_train, X_test, y_train, y_test =
train_test_split(df_downsampled, y, test_size=0.2)
     ...: print (X_train.shape, y_train.shape)
     ...: print (X_test.shape, y_test.shape)
(1689, 27) (1689,)
(423, 27) (423,)

In [128]: scaler = StandardScaler()
     ...: Xtrain_std = scaler.fit_transform(X_train)
     ...: Xtest_std = scaler.fit_transform(X_test)
     ...: # normalization
     ...: X = preprocessing.normalize(X_train, norm='l2')
     ...:
     ...: svc_rbf = SVC(kernel='rbf', class_weight='balanced',
C=10.0, random_state=0)
     ...: model = svc_rbf.fit(Xtrain_std, y_train)
     ...: # Create support vector classifier
     ...: svc = SVC(kernel='linear', class_weight='balanced',
C=10.0, random_state=0)
     ...: model = svc.fit(Xtrain_std, y_train)
     ...:
     ...: logmodel = LogisticRegression()
     ...: logmodel.fit(Xtrain_std,y_train) # traning the model by
logistic
Out[128]:
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None,
solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

In [129]: predictions = logmodel.predict(X_test) # input new data
to predict the result
     ...: print(classification_report(y_test,predictions)) #
compare the predicted and y_test
     ...: print("accuracy_score")
     ...: print( accuracy_score(predictions, y_test) )
              precision    recall  f1-score   support

           0       0.54      0.60      0.57       209
```

```
           1       0.56      0.50      0.53       214

   micro avg       0.55      0.55      0.55       423
   macro avg       0.55      0.55      0.55       423
weighted avg       0.55      0.55      0.55       423

accuracy_score
0.5508274231678487

In [130]: svm_prediction=svc.predict(X_test)
     ...: print(classification_report(y_test,svm_prediction))
     ...: print("accuracy_score")
     ...: print( accuracy_score(svm_prediction, y_test) )
     ...:
     ...: #
==============================================================
============
              precision    recall  f1-score   support

           0       0.50      1.00      0.66       209
           1       1.00      0.01      0.02       214

   micro avg       0.50      0.50      0.50       423
   macro avg       0.75      0.50      0.34       423
weighted avg       0.75      0.50      0.34       423

accuracy_score
0.4988179669030733

In [131]: svm_prediction_rbf=svc_rbf.predict(X_test)
     ...: print(classification_report(y_test,svm_prediction_rbf))
     ...: print("accuracy_score")
     ...: print( accuracy_score(svm_prediction_rbf, y_test) )
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       209
           1       0.51      1.00      0.67       214

   micro avg       0.51      0.51      0.51       423
   macro avg       0.25      0.50      0.34       423
weighted avg       0.26      0.51      0.34       423

accuracy_score
0.5059101654846335

In [132]:
```