Loan Repayment Assessment in Banking By Rishabh Yadav

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sweetviz as sv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score,cross_val_predict
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.utils import resample
from sklearn.decomposition import PCA

%matplotlib inline

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

In [3]:
```
# Read dataset for exploratory analysis
data = pd.read_csv('train_loan_data (1) (1).csv')
```

In [4]:
```
data.head()
```

Out[4]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_range_low
0	CO	85000.0	Jul-97	10+ years	Deputy	744	74(
1	CA	40000.0	Apr-87	10+ years	Department of Veterans Affairs	724	72(
2	FL	60000.0	Aug-07	10+ years	Marble polishing	679	675
3	IL	100742.0	Sep-80	10+ years	printer	664	66(
4	MD	80000.0	Jul-99	10+ years	Southern Mgmt	669	665

5 rows × 28 columns

In [5]:
```
# Shape command will give the number of rows and columns present in the dataset

data.shape
```

Out[5]:
```
(80000, 28)
```

In [6]:
```
#The info command will help us to understand the different columns present in the d

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80000 entries, 0 to 79999
Data columns (total 28 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   addr_state            80000 non-null  object 
 1   annual_inc            80000 non-null  float64
 2   earliest_cr_line      80000 non-null  object 
 3   emp_length            75412 non-null  object 
 4   emp_title             74982 non-null  object 
 5   fico_range_high       80000 non-null  int64  
 6   fico_range_low        80000 non-null  int64  
 7   grade                 80000 non-null  object 
 8   home_ownership        80000 non-null  object 
 9   application_type      80000 non-null  object 
 10  initial_list_status   80000 non-null  object 
 11  int_rate              80000 non-null  float64
 12  loan_amnt             80000 non-null  int64  
 13  num_actv_bc_tl        76052 non-null  float64
 14  mort_acc              77229 non-null  float64
 15  tot_cur_bal           76052 non-null  float64
 16  open_acc              80000 non-null  int64  
 17  pub_rec               80000 non-null  int64  
 18  pub_rec_bankruptcies  79969 non-null  float64
 19  purpose               80000 non-null  object 
 20  revol_bal             80000 non-null  int64  
 21  revol_util            79947 non-null  float64
 22  sub_grade             80000 non-null  object 
 23  term                  80000 non-null  object 
 24  title                 79030 non-null  object 
 25  total_acc             80000 non-null  int64  
 26  verification_status   80000 non-null  object 
 27  loan_status           80000 non-null  object 
dtypes: float64(7), int64(7), object(14)
memory usage: 17.1+ MB
```

In [7]:
```
data.describe()
```

Out[7]:

	annual_inc	fico_range_high	fico_range_low	int_rate	loan_amnt	num_actv_bc_tl
count	8.000000e+04	80000.000000	80000.000000	80000.000000	80000.000000	76052.000000
mean	7.604614e+04	699.987975	695.987813	13.232898	14403.867813	3.633790
std	6.902006e+04	31.734840	31.734075	4.771705	8703.826298	2.262505
min	0.000000e+00	664.000000	660.000000	5.310000	750.000000	0.000000
25%	4.600000e+04	674.000000	670.000000	9.750000	7925.000000	2.000000
50%	6.500000e+04	694.000000	690.000000	12.740000	12000.000000	3.000000
75%	9.000000e+04	714.000000	710.000000	15.990000	20000.000000	5.000000
max	7.141778e+06	850.000000	845.000000	30.990000	40000.000000	32.000000

In [8]:
```
# Let us run sweetviz for exploratory data analysis

eda = sv.analyze(data)
eda.show_html('eda.html')
eda.show_notebook()
```

```
| [  0%]   00:00 ->…
Report eda.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop u
p, regardless, the report IS saved in your notebook/colab files.
```

SweetVIZ
2.3.1
Get updates, docs & report issues here
Created & maintained by Francois Bertrand
Graphic design by Jean-Francois Hains

Da

ASSOCIATIONS

1 addr_state

VALUES:	80,000 (100%)
MISSING:	---
DISTINCT:	51 (<1%)

2 annual_inc

VALUES:	80,000 (100%)
MISSING:	---
DISTINCT:	7,536 (9%)
ZEROES:	17 (<1%)

MAX	7.1M
95%	0.2M
Q3	0.1M
AVG	0.1M
MEDIAN	0.1M
Q1	0.0M
5%	0.0M
MIN	0.0M

3 earliest_cr_line

VALUES:	80,000 (100%)
MISSING:	---
DISTINCT:	640 (<1%)

547	<1%	Sep-03
545	<1%	Aug-03
544	<1%	Aug-01
541	<1%	Oct-01
539	<1%	Sep-02
537	<1%	Sep-04
535	<1%	Oct-00
76,212	95%	(Other)

4 emp_length

VALUES:	75,412 (94%)

In [9]:
```
## Find the missing values

data.isnull().sum()* 100 / len(data)
```

Out[9]:
```
addr_state              0.00000
annual_inc              0.00000
earliest_cr_line        0.00000
emp_length              5.73500
emp_title               6.27250
fico_range_high         0.00000
fico_range_low          0.00000
grade                   0.00000
home_ownership          0.00000
application_type        0.00000
initial_list_status     0.00000
int_rate                0.00000
loan_amnt               0.00000
num_actv_bc_tl          4.93500
mort_acc                3.46375
tot_cur_bal             4.93500
open_acc                0.00000
pub_rec                 0.00000
pub_rec_bankruptcies    0.03875
purpose                 0.00000
revol_bal               0.00000
revol_util              0.06625
sub_grade               0.00000
term                    0.00000
title                   1.21250
total_acc               0.00000
verification_status     0.00000
loan_status             0.00000
dtype: float64
```

In [10]:
```
# Replace missing values from columns with object datatype with the mode

colso = ['emp_length','emp_title','title']
for col in colso:
    data[col].fillna(data[col].mode()[0],inplace=True)
```

In [11]:
```
# Replace missing values from the remaining columns with float datatype with the me

colsf = ['num_actv_bc_tl','mort_acc','tot_cur_bal','pub_rec_bankruptcies','revol_ut
for col in colsf:
    data[col].fillna(data[col].mean(), inplace=True)
```

In [12]:
```
## Check for missing values again

data.isnull().sum()* 100 / len(data)
```

Out[12]:
```
addr_state              0.0
annual_inc              0.0
earliest_cr_line        0.0
emp_length              0.0
emp_title               0.0
fico_range_high         0.0
fico_range_low          0.0
grade                   0.0
home_ownership          0.0
application_type        0.0
initial_list_status     0.0
int_rate                0.0
loan_amnt               0.0
num_actv_bc_tl          0.0
mort_acc                0.0
tot_cur_bal             0.0
open_acc                0.0
pub_rec                 0.0
pub_rec_bankruptcies    0.0
purpose                 0.0
revol_bal               0.0
revol_util              0.0
sub_grade               0.0
term                    0.0
title                   0.0
total_acc               0.0
verification_status     0.0
loan_status             0.0
dtype: float64
```

In [13]:
```
data.head()
```

Out[13]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_range_low
0	CO	85000.0	Jul-97	10+ years	Deputy	744	740
1	CA	40000.0	Apr-87	10+ years	Department of Veterans Affairs	724	720
2	FL	60000.0	Aug-07	10+ years	Marble polishing	679	675
3	IL	100742.0	Sep-80	10+ years	printer	664	660
4	MD	80000.0	Jul-99	10+ years	Southern Mgmt	669	665

5 rows × 28 columns

In [14]:
```
# Let us check the number of unique values in each column

data.nunique()
```

Out[14]:
```
addr_state                51
annual_inc              7536
earliest_cr_line         640
emp_length                11
emp_title              36661
fico_range_high           38
fico_range_low            38
grade                      7
home_ownership             6
application_type           2
initial_list_status        2
int_rate                 549
loan_amnt               1373
num_actv_bc_tl            29
mort_acc                  29
tot_cur_bal            65411
open_acc                  56
pub_rec                   15
pub_rec_bankruptcies       9
purpose                   14
revol_bal              32971
revol_util              1081
sub_grade                 35
term                       2
title                   5348
total_acc                107
verification_status        3
loan_status                2
dtype: int64
```

In [15]:
```
# Let us list the datatype for each column against the number of unique values

list(zip(data.columns,data.dtypes,data.nunique()))
```

Out[15]:
```
[('addr_state', dtype('O'), 51),
 ('annual_inc', dtype('float64'), 7536),
 ('earliest_cr_line', dtype('O'), 640),
 ('emp_length', dtype('O'), 11),
 ('emp_title', dtype('O'), 36661),
 ('fico_range_high', dtype('int64'), 38),
 ('fico_range_low', dtype('int64'), 38),
 ('grade', dtype('O'), 7),
 ('home_ownership', dtype('O'), 6),
 ('application_type', dtype('O'), 2),
 ('initial_list_status', dtype('O'), 2),
 ('int_rate', dtype('float64'), 549),
 ('loan_amnt', dtype('int64'), 1373),
 ('num_actv_bc_tl', dtype('float64'), 29),
 ('mort_acc', dtype('float64'), 29),
 ('tot_cur_bal', dtype('float64'), 65411),
 ('open_acc', dtype('int64'), 56),
 ('pub_rec', dtype('int64'), 15),
 ('pub_rec_bankruptcies', dtype('float64'), 9),
 ('purpose', dtype('O'), 14),
 ('revol_bal', dtype('int64'), 32971),
 ('revol_util', dtype('float64'), 1081),
 ('sub_grade', dtype('O'), 35),
 ('term', dtype('O'), 2),
 ('title', dtype('O'), 5348),
 ('total_acc', dtype('int64'), 107),
 ('verification_status', dtype('O'), 3),
 ('loan_status', dtype('O'), 2)]
```

In [16]:
```
# Let us format the emp_length column properly and change the datatype to int

data['emp_length'] = data['emp_length'].replace({'years':'','year':'',' ':'','<':'
data['emp_length'] = data['emp_length'].apply(lambda x:int(x))
```

In [17]:
```
# Let us convert earliest_cr_line column to numeric value

data['earliest_cr_line'] = pd.to_datetime(data['earliest_cr_line'],format='%b-%y')

# Convert datetime format to numeric format

data['earliest_cr_line'] = pd.to_numeric(data['earliest_cr_line'])
```

In [18]:
```
# Convert the term column to numeric value

# Remove the 'months' string from the column
data['term'] = data['term'].replace(' months','',regex=True)

# Convert the column to numeric format
data['term'] = pd.to_numeric(data['term'])
```

In [19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80000 entries, 0 to 79999
Data columns (total 28 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   addr_state            80000 non-null  object 
 1   annual_inc            80000 non-null  float64
 2   earliest_cr_line      80000 non-null  int64  
 3   emp_length            80000 non-null  int64  
 4   emp_title             80000 non-null  object 
 5   fico_range_high       80000 non-null  int64  
 6   fico_range_low        80000 non-null  int64  
 7   grade                 80000 non-null  object 
 8   home_ownership        80000 non-null  object 
 9   application_type      80000 non-null  object 
 10  initial_list_status   80000 non-null  object 
 11  int_rate              80000 non-null  float64
 12  loan_amnt             80000 non-null  int64  
 13  num_actv_bc_tl        80000 non-null  float64
 14  mort_acc              80000 non-null  float64
 15  tot_cur_bal           80000 non-null  float64
 16  open_acc              80000 non-null  int64  
 17  pub_rec               80000 non-null  int64  
 18  pub_rec_bankruptcies  80000 non-null  float64
 19  purpose               80000 non-null  object 
 20  revol_bal             80000 non-null  int64  
 21  revol_util            80000 non-null  float64
 22  sub_grade             80000 non-null  object 
 23  term                  80000 non-null  int64  
 24  title                 80000 non-null  object 
 25  total_acc             80000 non-null  int64  
 26  verification_status   80000 non-null  object 
 27  loan_status           80000 non-null  object 
dtypes: float64(7), int64(10), object(11)
memory usage: 17.1+ MB
```

In [20]:
```
# Let us drop addr_state and emp_tittle columns from our dataset
```

In [21]:
```
# Function to separate the numerical and categorical columns

def data_type(dataset):
    """
    Function to identify the numerical and categorical data columns
    :param dataset: Dataframe
    :return: list of numerical and categorical columns
    """
    numerical = dataset.select_dtypes(include=['int64','float64'])
    categorical = dataset.select_dtypes(include=['object'])
    return numerical, categorical
```

In [22]:
```
# Separate numerical and categorical columns

numerical, categorical = data_type(data)
```

In [23]:
```
# Function to identify binary columns and ignore them from scaling
def binary_columns(df):
    """
    Generates a list of binary columns in a dataframe.
    """
    binary_cols = []
    for col in df.select_dtypes(include=['int', 'float']).columns:
        unique_values = df[col].unique()
        if np.in1d(unique_values, [0, 1]).all():
            binary_cols.append(col)
    return binary_cols
```

In [24]:
```
# Remove the binary columns from the numerical columns

binary_cols = binary_columns(data)

numerical = [i for i in numerical if i not in binary_cols]
```

In [25]:
```
# Function to encode categorical columns

def encoding(dataset, categorical):
    """
    Function to automate the process of encoding the categorical data
    :param dataset: Dataframe
    :param categorical: List of categorical columns
    :return: Dataframe
    """
    for i in categorical:
        dataset[i] = dataset[i].astype('category')
        dataset[i] = dataset[i].cat.codes
    return dataset
```

In [26]:
```
# Encode categorical columns

data = encoding(data, categorical)
```

In [27]:
```
# Function to perform feature scaling of numerical data

def feature_scaling(dataset, numerical):
    """
    Function to automate the process of feature scaling the numerical data
    :param dataset: Dataframe
    :param numerical: List of numerical columns
    :return: Dataframe
    """
```

```
    sc = StandardScaler()
    dataset[numerical] = sc.fit_transform(dataset[numerical])
    return dataset
```

In [28]:
```
data.head()
```

Out[28]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_rang
0	5	85000.0	867715200000000000	10	7926	744	
1	4	40000.0	544233600000000000	10	7872	724	
2	9	60000.0	1185926400000000000	10	17127	679	
3	14	100742.0	336614400000000000	10	35120	664	
4	20	80000.0	930787200000000000	10	26487	669	

5 rows × 28 columns

In [29]:
```
status = data['loan_status'].value_counts()
status
```

Out[29]:
```
1    64030
0    15970
Name: loan_status, dtype: int64
```

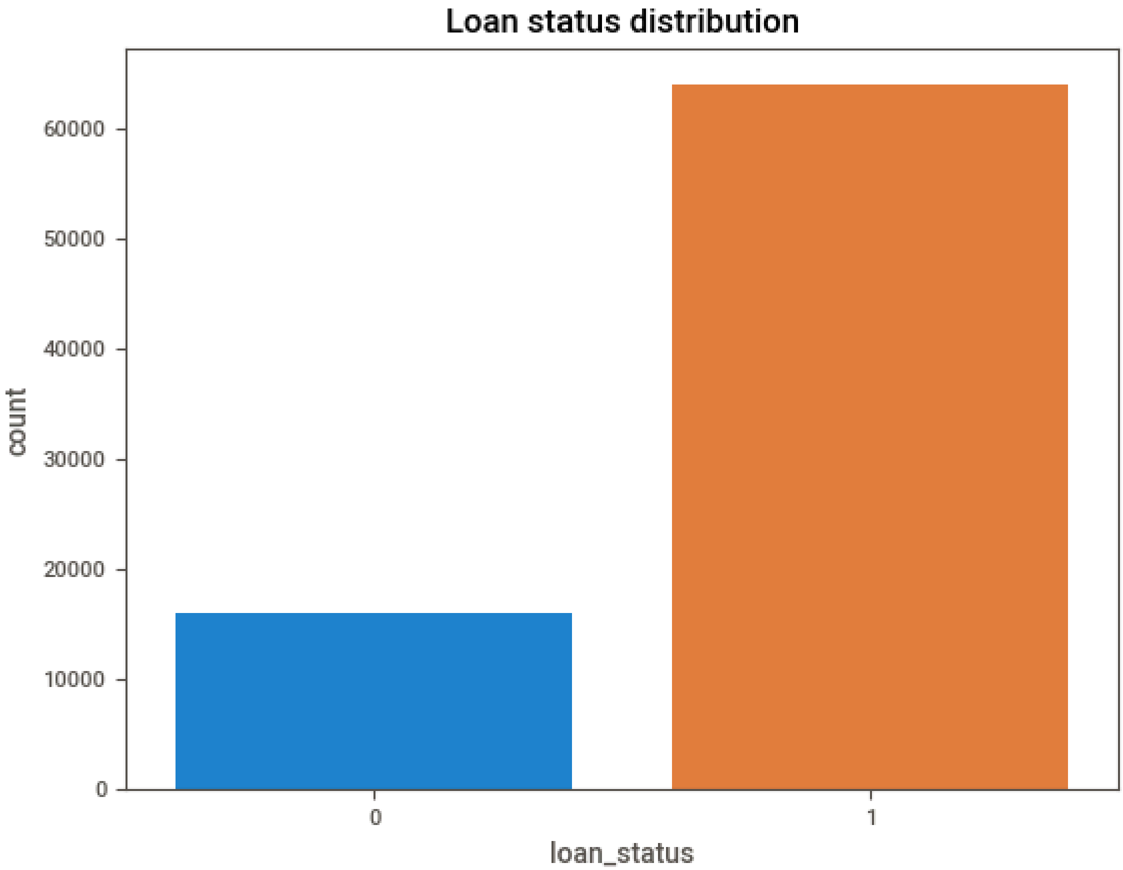
In [30]:
```
# Let us obtain the percentage for each class starting with paid

paid = round((status[1]/data['loan_status'].count()*100),2)
paid
```

Out[30]:
```
80.04
```

In [31]:
```
# Percentage of customers that defaulted

defaulted = round((status[0]/data['loan_status'].count()*100),2)
defaulted
```

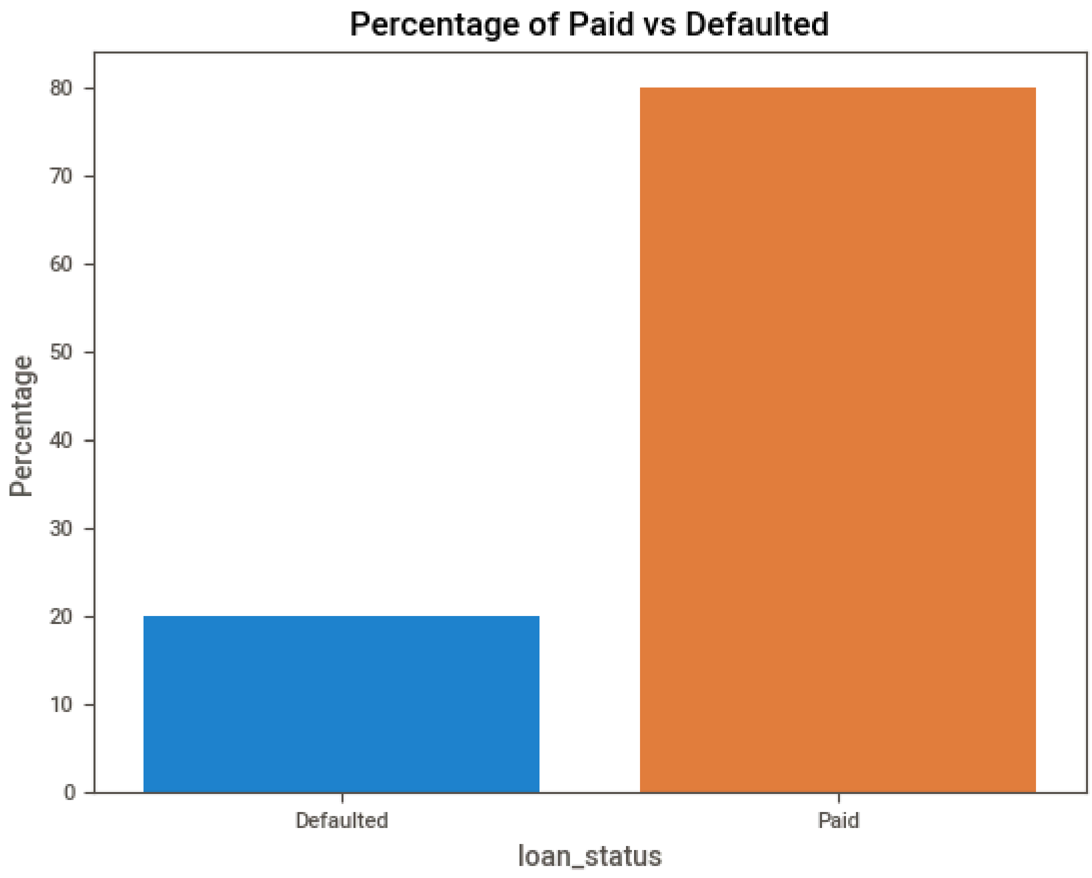
Out[31]:
```
19.96
```

In [32]:
```
# Let us display paid against defaulted

sns.countplot(x='loan_status', data=data)
plt.title('Loan status distribution')
plt.show()
```



In [33]:

```
# Display percentage of paid against defaulted

status_percentage = {'loan_status':['Defaulted', 'Paid'], 'Percentage':[defaulted,
df_status_percentage = pd.DataFrame(status_percentage)
sns.barplot(x='loan_status',y='Percentage', data=df_status_percentage)
plt.title('Percentage of Paid vs Defaulted')
plt.show()
```



In [34]:
```
# Create paid dataframe
data_paid = data[data['loan_status'] == 1]
# Create non fraudulent dataframe
data_defaulted = data[data['loan_status'] == 0]
```

In [35]:
```
data_paid.shape
```

Out[35]:
```
(64030, 28)
```

In [36]:
```
data_defaulted.shape
```

Out[36]:
```
(15970, 28)
```

In [37]:
```
# Paid transactions

data_paid.loan_amnt.describe()
```

Out[37]:
```
count    64030.000000
mean     14122.549977
std       8657.016959
min        750.000000
25%       7500.000000
50%      12000.000000
75%      20000.000000
max      40000.000000
Name: loan_amnt, dtype: float64
```

In [38]:
```
data_paid.loan_amnt.sum()
```

Out[38]:
```
904266875
```

In [39]:
```
# Defaulted transactions

data_defaulted.loan_amnt.describe()
```

Out[39]:
```
count    15970.000000
mean     15531.781465
std       8799.439167
min       1000.000000
25%       9000.000000
50%      14387.500000
75%      20125.000000
max      40000.000000
Name: loan_amnt, dtype: float64
```

In [40]:
```
data_defaulted.loan_amnt.sum()
```

Out[40]:
```
248042550
```

In [41]:
```
# Perform feature scaling of numerical data

data = feature_scaling(data, numerical)
```

In [42]:
```
data.head()
```

Out[42]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_range_low
0	5	0.129729	-0.238254	1.042009	7926	1.386876	1.386915
1	4	-0.522259	-1.501316	1.042009	7872	0.756650	0.756674
2	9	-0.232487	1.004228	1.042009	17127	-0.661358	-0.661369
3	14	0.357809	-2.311982	1.042009	35120	-1.134028	-1.134050
4	20	0.057286	0.008016	1.042009	26487	-0.976472	-0.976490

5 rows × 28 columns

In [43]:
```
# Put feature variables into X

# Drop emp_title, addr_state and title columns in addition to the target column Loa

X = data.drop(['loan_status','emp_title','addr_state','title'], axis=1)
```

In [44]:
```
# Put target variable to y
y = data['loan_status']
```

In [45]:
```
#Split the dataset into train and test based on the 80-20 ratio

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [46]:
```
# Instantiate Logistic Regression and fit it.

lr = LogisticRegression()
lr.fit(X_train,y_train)
```

Out[46]:
```
▾ LogisticRegression
LogisticRegression()
```

In [47]:
```
y_pred = lr.predict(X_test)
```

In [48]:
```
accuracy_score(y_test,y_pred)
```
Out[48]:
```
0.8033125
```

In [49]:
```
f1_score(y_test,y_pred)
```
Out[49]:
```
0.8893031763340251
```

In [50]:
```
# Define hyperparameters to search over

parameters = {'C': [0.001,0.01,0.1,1,10,100,1000]}

# Perform a grid search over the hyperparameters

grid_search = GridSearchCV(lr,param_grid=parameters,scoring='f1',cv=10)
grid_search.fit(X_train,y_train)

# Print best hyperparameters and F1 score

print('Best hyperparameters: ',grid_search.best_params_)
print('F1 score: ',grid_search.best_score_)
```
```
Best hyperparameters:  {'C': 0.001}
F1 score:  0.886692613553619
```

In [51]:
```
# Define the number of folds for k-fold cross-validation

k = 10

# Define the k-fold cross-validation object

kf = KFold(n_splits=k,shuffle=True,random_state=1)

# Perform k-fold cross-validation on the model

f1_scores = cross_val_score(lr,X_train,y_train,cv=kf,scoring='f1')

# Print the mean F1 score and standard deviation

print('Mean F1 score: ',f1_scores.mean())

print('Standard deviation: ',f1_scores.std())
```
```
Mean F1 score:  0.8865745397188443
Standard deviation:  0.0022794382466072246
```

In [52]:
```
# Instantiate Random Forest Model and fit it

classifier = RandomForestClassifier(n_estimators=10,criterion='entropy',random_stat
classifier.fit(X_train,y_train)
```
Out[52]:
```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state
=0)
```

In [53]:
```
y_pred2 = classifier.predict(X_test)
```

In [54]:
```
accuracy_score(y_test,y_pred2)
```
Out[54]:
```
0.78125
```

In [55]:
```
f1_score(y_test,y_pred2)
```

Out[55]:
```
0.8718981040919406
```

In [56]:
```
#Lets create a PCA object, with 3 components, i.e we want to create 3 columns from
pca = PCA(n_components=3)
#lets fit the pca on our feature scaled dataframe.
pca.fit(X_train)
#create a dataframe from the newly created PCA results.
X_train_new = pd.DataFrame(pca.transform(X_train), columns=(["new1","new2", "new3"]
```

In [57]:
```
#lets look at our new feature dataset
X_train_new.head()
```

Out[57]:

	new1	new2	new3
0	10.783673	-2.272124	-1.741526
1	-0.673693	-2.480081	2.005823
2	5.639230	-0.256948	-2.805807
3	-5.867541	-0.777823	0.983234
4	-4.690140	-2.359649	0.972511

In [58]:
```
# Define the hyperparameters to search over

parameters = {'n_estimators': [50,100,150],
              'max_depth': [5,10,15],
              'min_samples_split': [2,5,10],
              'min_samples_leaf': [1,2,4]
             }

# Apply grid search to our Random Forest Model

grid_search = GridSearchCV(classifier,param_grid=parameters,cv=5,scoring='f1')
grid_search.fit(X_train_new,y_train)

# Print the best hyperparameters and F1 score

print('Best hyperparameters: ', grid_search.best_params_)
print('F1 score: ', grid_search.best_score_)
```

```
Best hyperparameters:  {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_spli
t': 2, 'n_estimators': 150}
F1 score:  0.8882947097933578
```

In [59]:
```
# Separate majority (paid) status and minority (defaulted) status

paid_status = data[data['loan_status']==1]
default_status = data[data['loan_status']==0]
```

In [60]:
```
# Check the shape of the paid_status dataset

paid_status.shape
```

Out[60]:
```
(64030, 28)
```

In [61]:
```
# Check the shape of the default_status dataset

default_status.shape
```

Out[61]:
```
(15970, 28)
```

In [62]:
```
# Let us undersample the majority (paid_status) dataset

undersampled_paid = resample(paid_status,
                             replace=False,
                             n_samples=len(default_status),
                             random_state=2)
```

In [63]:
```
# Let us combine the default_status with undersampled_paid

undersampled_data = pd.concat([default_status,undersampled_paid])

# Shuffle the dataset

undersampled_data = undersampled_data.sample(frac=1,random_state=2)
```

In [64]:
```
undersampled_data.head()
```

Out[64]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_range_l
68256	4	0.636831	0.870635	0.202751	23105	-0.976472	-0.976
60642	43	0.709274	-0.412330	1.042009	25605	2.489772	2.489
47446	31	0.015733	-0.053046	1.042009	27098	-0.661358	-0.661
64196	9	-0.580214	-1.501316	0.762256	13392	1.071763	1.071
45896	36	-0.478793	-0.248375	-0.636507	5674	1.386876	1.386

5 rows × 28 columns

In [65]:
```
# Put feature variables into X

# Drop emp_title, addr_state and title columns in addition to the target column Loa

X = undersampled_data.drop(['loan_status','emp_title','addr_state','title'], axis=1
```

In [66]:
```
# Put target variable to y

y = undersampled_data['loan_status']
```

In [67]:
```
#Split the dataset into train and test based on the 80-20 ratio

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [68]:
```
# Instantiate Logistic Regression and fit it.

regressor = LogisticRegression()
regressor.fit(X_train,y_train)
```

Out[68]:
```
▾ LogisticRegression
LogisticRegression()
```

In [69]:
```
y1_pred = regressor.predict(X_test)
```

In [70]:
```
accuracy_score(y_test,y1_pred)
```
Out[70]:
```
0.6484032561051972
```

In [71]:
```
f1_score(y_test,y1_pred)
```
Out[71]:
```
0.6549923195084486
```

In [72]:
```
# Define hyperparameters to search over

parameters = {'C': [0.001,0.01,0.1,1,10,100,1000]}

# Perform a grid search over the hyperparameters

grid_search = GridSearchCV(regressor,param_grid=parameters,scoring='f1',cv=10)
grid_search.fit(X_train,y_train)

# Print best hyperparameters and F1 score

print('Best hyperparameters: ',grid_search.best_params_)
print('F1 score: ',grid_search.best_score_)
```
```
Best hyperparameters:  {'C': 0.001}
F1 score:  0.6542187227709134
```

In [73]:
```
# Instantiate Random Forest Model and fit it

rclassifier = RandomForestClassifier(n_estimators=10,criterion='entropy',random_sta
rclassifier.fit(X_train,y_train)
```
Out[73]:
```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state
=0)
```

In [74]:
```
y2_pred = rclassifier.predict(X_test)
```

In [75]:
```
accuracy_score(y_test,y2_pred)
```
Out[75]:
```
0.606762680025047
```

In [76]:
```
f1_score(y_test,y2_pred)
```
Out[76]:
```
0.5735144312393887
```

In [77]:
```
#Lets create a PCA object, with 3 components, i.e we want to create 3 columns from
pca = PCA(n_components=3)
#lets fit the pca on our feature scaled dataframe.
pca.fit(X_train)
#create a dataframe from the newly created PCA results.
X_train_new = pd.DataFrame(pca.transform(X_train), columns=(["new1","new2", "new3"]
```

In [78]:
```
rfclassifier = RandomForestClassifier(n_estimators=50,criterion='entropy',max_depth
                                     min_samples_leaf=1,min_samples_split=2,random_
```

In [79]:
```
# Fit the model
rfclassifier.fit(X_train,y_train)
```

Out[79]:

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=5, n_estimators=50,
                       random_state=0)
```

In [80]:

```
y3_pred = rfclassifier.predict(X_test)
```

In [81]:

```
f1_score(y_test,y3_pred)
```

Out[81]:

```
0.6207126207126207
```

In [82]:

```
# Create instance of the best model

rfclassifier = RandomForestClassifier(n_estimators=150,criterion='entropy',max_dept
                                     min_samples_leaf=1,min_samples_split=2,random_
```

In [83]:

```
#Train the best model on the entire dataset obtained after undersampling

rfclassifier.fit(X,y)
```

Out[83]:

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=15, n_estimators=15
0,
                       random_state=0)
```

In [84]:

```
# Import joblib to be used in saving the model

import joblib
```

In [85]:

```
#Save the model

joblib.dump(rfclassifier, "loan_repayment_assessment_model")
```

Out[85]:

```
['loan_repayment_assessment_model']
```

In [86]:

```
test_df = pd.read_csv('test_loan_data (1) (1).csv')
test_df.head()
```

Out[86]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_range_
0	MO	50000.0	May-2012	1 year	Tower technician	719.0	7
1	HI	92000.0	Dec-2001	10+ years	Supervisor	684.0	68
2	TX	89000.0	Mar-1989	10+ years	APPLICATIONS PROGRAMMER	679.0	67
3	CA	33000.0	Nov-2004	9 years	San Diego Unified School District	674.0	67
4	MI	35580.0	Feb-1997	NaN	NaN	704.0	70

5 rows × 27 columns

In [87]:
```
# Use shape command on the dataset

test_df.shape
```

Out[87]:
```
(20000, 27)
```

In [88]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   addr_state            20000 non-null  object 
 1   annual_inc            20000 non-null  float64
 2   earliest_cr_line      20000 non-null  object 
 3   emp_length            18742 non-null  object 
 4   emp_title             18622 non-null  object 
 5   fico_range_high       20000 non-null  float64
 6   fico_range_low        20000 non-null  float64
 7   grade                 20000 non-null  object 
 8   home_ownership        20000 non-null  object 
 9   application_type      20000 non-null  object 
 10  initial_list_status   20000 non-null  object 
 11  int_rate              20000 non-null  float64
 12  loan_amnt             20000 non-null  float64
 13  num_actv_bc_tl        18989 non-null  float64
 14  mort_acc              19296 non-null  float64
 15  tot_cur_bal           18989 non-null  float64
 16  open_acc              20000 non-null  float64
 17  pub_rec               20000 non-null  float64
 18  pub_rec_bankruptcies  19989 non-null  float64
 19  purpose               20000 non-null  object 
 20  revol_bal             20000 non-null  float64
 21  revol_util            19987 non-null  float64
 22  sub_grade             20000 non-null  object 
 23  term                  20000 non-null  object 
 24  title                 19753 non-null  object 
 25  total_acc             20000 non-null  float64
 26  verification_status   20000 non-null  object 
dtypes: float64(14), object(13)
memory usage: 4.1+ MB
```

In [89]:
```
## Find the missing values

test_df.isnull().sum()* 100 / len(test_df)
```

Out[89]:
```
addr_state              0.000
annual_inc              0.000
earliest_cr_line        0.000
emp_length              6.290
emp_title               6.890
fico_range_high         0.000
fico_range_low          0.000
grade                   0.000
home_ownership          0.000
application_type        0.000
initial_list_status     0.000
int_rate                0.000
loan_amnt               0.000
num_actv_bc_tl          5.055
mort_acc                3.520
tot_cur_bal             5.055
open_acc                0.000
pub_rec                 0.000
pub_rec_bankruptcies    0.055
purpose                 0.000
revol_bal               0.000
revol_util              0.065
sub_grade               0.000
term                    0.000
title                   1.235
total_acc               0.000
verification_status     0.000
dtype: float64
```

In [90]:
```
# Replace missing values from columns with object datatype with the mode

cols = ['emp_length','emp_title','title']
for col in cols:
    test_df[col].fillna(test_df[col].mode()[0],inplace=True)
```

In [91]:
```
# Replace missing values from the remaining columns with float datatype with the me

colsf = ['num_actv_bc_tl','mort_acc','tot_cur_bal','pub_rec_bankruptcies','revol_ut
for col in colsf:
    test_df[col].fillna(test_df[col].mean(), inplace=True)
```

In [92]:
```
# Let us format the emp_length column properly and change the datatype to int

test_df['emp_length'] = test_df['emp_length'].replace({'years':'','year':'',' ':'',
test_df['emp_length'] = test_df['emp_length'].apply(lambda x:int(x))
```

In [93]:
```
# Convert the term column to numeric value

# Remove the 'months' string from the column
test_df['term'] = test_df['term'].replace(' months','',regex=True)

# Convert the column to numeric format
test_df['term'] = pd.to_numeric(test_df['term'])
```

In [94]:
```
# Separate numerical and categorical columns

test_numerical, test_categorical = data_type(test_df)
```

In [95]:
```
# Remove the binary columns from the numerical columns

binary_cols = binary_columns(test_df)
```

```
test_numerical = [i for i in test_numerical if i not in binary_cols]
```

In [96]:
```
# Encode categorical columns

test_df = encoding(test_df, test_categorical)
```

In [97]:
```
# Perform feature scaling of numerical data

test_df = feature_scaling(test_df, test_numerical)
```

In [98]:
```
test_df.head()
```

Out[98]:

	addr_state	annual_inc	earliest_cr_line	emp_length	emp_title	fico_range_high	fico_range_low
0	23	-0.309267	425	-1.479155	8992	0.591656	0.591674
1	11	0.180936	126	1.032191	8525	-0.510089	-0.510096
2	42	0.145921	356	1.032191	98	-0.667482	-0.667492
3	4	-0.507683	462	0.753153	7462	-0.824874	-0.824888
4	21	-0.477570	167	1.032191	8743	0.119480	0.119487

5 rows × 27 columns

In [99]:
```
test_df = test_df.drop(['emp_title','addr_state','title'], axis=1)
```

In [100…
```
test_df.head()
```

Out[100]:

	annual_inc	earliest_cr_line	emp_length	fico_range_high	fico_range_low	grade	home_ownershi
0	-0.309267	425	-1.479155	0.591656	0.591674	2	
1	0.180936	126	1.032191	-0.510089	-0.510096	1	
2	0.145921	356	1.032191	-0.667482	-0.667492	1	
3	-0.507683	462	0.753153	-0.824874	-0.824888	2	
4	-0.477570	167	1.032191	0.119480	0.119487	1	

5 rows × 24 columns

In [101…
```
#Load the saved model for use

model = joblib.load("loan_repayment_assessment_model")
```

In [102…
```
# Perform prediction with the saved model on the test_loan dataset

predict = model.predict(test_df)
```

In [103…
```
predict
```

Out[103]:
```
array([1, 1, 1, ..., 1, 1, 0], dtype=int8)
```

In [104…
```
test_df['predicted'] = pd.DataFrame(predict)
```

In [105…

```
test_df.head()
```

Out[105]:

	annual_inc	earliest_cr_line	emp_length	fico_range_high	fico_range_low	grade	home_ownershi
0	-0.309267	425	-1.479155	0.591656	0.591674	2	
1	0.180936	126	1.032191	-0.510089	-0.510096	1	
2	0.145921	356	1.032191	-0.667482	-0.667492	1	
3	-0.507683	462	0.753153	-0.824874	-0.824888	2	
4	-0.477570	167	1.032191	0.119480	0.119487	1	

5 rows × 25 columns

In [106…

```
test_df.to_csv('predicted.csv',index=False)
```

In [ ]: