In [1]:
```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
```

In [2]:
```
credit_df = pd.read_csv(r".\Project 1 Finance Predictive Analysis-ML.csv")
credit_df.head()
```

Out[2]:

	customer_id	loan_id	loan_type	loan_amount	interest_rate	loan_term	employment_type	inc
0	CUST-00004912	LN00004170	Car Loan	16795	0.051852	15	Self-employed	
1	CUST-00004194	LN00002413	Personal Loan	1860	0.089296	56	Full-time	
2	CUST-00003610	LN00000024	Personal Loan	77820	0.070470	51	Full-time	
3	CUST-00001895	LN00001742	Car Loan	55886	0.062155	30	Full-time	
4	CUST-00003782	LN00003161	Home Loan	7265	0.070635	48	Part-time	

In [3]:
```
credit_df["default_status"].value_counts()
```

Out[3]:
```
default_status
False    4001
True      999
Name: count, dtype: int64
```

Choose only relevant columns

In [4]:
```
#the application_date, approval_date, disbursement_date, due_date has been dropped
credit_df = credit_df[["loan_type", "loan_amount","interest_rate","employment_type","i
credit_df.head()
```

Out[4]:

	loan_type	loan_amount	interest_rate	employment_type	income_level	credit_score	gender	marit
0	Car Loan	16795	0.051852	Self-employed	Medium	833	Male	
1	Personal Loan	1860	0.089296	Full-time	Medium	776	Female	
2	Personal Loan	77820	0.070470	Full-time	Low	697	Male	
3	Car Loan	55886	0.062155	Full-time	Low	795	Female	
4	Home Loan	7265	0.070635	Part-time	Low	519	Female	

In [5]:

```
credit_df["default_status"].value_counts()
```

Out[5]:

```
default_status
False    4001
True      999
Name: count, dtype: int64
```

Change the categorical data to numeric

In [6]:

```
# The categorical data are loan_type, employment_type, income_level, gender, marital_s
from sklearn.preprocessing import LabelEncoder
LabelEncoder = LabelEncoder()

credit_df["loan_type"] = LabelEncoder.fit_transform(credit_df["loan_type"])
credit_df["employment_type"] = LabelEncoder.fit_transform(credit_df["employment_type"]
credit_df["income_level"] = LabelEncoder.fit_transform(credit_df["income_level"])
credit_df["gender"] = LabelEncoder.fit_transform(credit_df["gender"])
credit_df["marital_status"] = LabelEncoder.fit_transform(credit_df["marital_status"])
credit_df["education_level"] = LabelEncoder.fit_transform(credit_df["education_level"]
credit_df["default_status"] = LabelEncoder.fit_transform(credit_df["default_status"])

credit_df
```

Out[6]:

	loan_type	loan_amount	interest_rate	employment_type	income_level	credit_score	gender	m
0	0	16795	0.051852	2	2	833	1	
1	3	1860	0.089296	0	2	776	0	
2	3	77820	0.070470	0	1	697	1	
3	0	55886	0.062155	0	1	795	0	
4	2	7265	0.070635	1	1	519	0	
...	...	...	...	...	...	...	...	
4995	0	37945	0.070087	2	0	511	1	
4996	3	48937	0.056405	1	2	502	1	
4997	2	7476	0.064212	0	0	452	0	
4998	0	52756	0.094914	2	2	728	1	
4999	3	91101	0.083821	2	1	586	1	

5000 rows × 10 columns

In [7]:

```
credit_df.columns
```

Out[7]:

```
Index(['loan_type', 'loan_amount', 'interest_rate', 'employment_type',
       'income_level', 'credit_score', 'gender', 'marital_status',
       'education_level', 'default_status'],
      dtype='object')
```

Identify the features and Targets

In [8]:

```
X = np.asanyarray(credit_df[['loan_type', 'loan_amount', 'interest_rate', 'employment_
       'income_level', 'credit_score', 'gender', 'marital_status',
       'education_level']])
X
```

Out[8]:

```
array([[0.0000000e+00, 1.6795000e+04, 5.1851709e-02, ..., 1.0000000e+00,
        2.0000000e+00, 2.0000000e+00],
       [3.0000000e+00, 1.8600000e+03, 8.9295672e-02, ..., 0.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [3.0000000e+00, 7.7820000e+04, 7.0469564e-02, ..., 1.0000000e+00,
        0.0000000e+00, 1.0000000e+00],
       ...,
       [2.0000000e+00, 7.4760000e+03, 6.4211792e-02, ..., 0.0000000e+00,
        2.0000000e+00, 1.0000000e+00],
       [0.0000000e+00, 5.2756000e+04, 9.4914482e-02, ..., 1.0000000e+00,
        1.0000000e+00, 3.0000000e+00],
       [3.0000000e+00, 9.1101000e+04, 8.3820967e-02, ..., 1.0000000e+00,
        2.0000000e+00, 2.0000000e+00]])
```

In [9]:

```
y = np.asanyarray(credit_df["default_status"])
y
```

Out[9]:

```
array([0, 0, 0, ..., 1, 0, 0], dtype=int64)
```

Normalize dataset

In [10]:
```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[10]:
```
array([[-1.33965636, -1.15378622, -1.82077885,  1.22327807,  1.21940701,
         1.63771851,  0.98333878,  1.23850864,  0.43330448],
       [ 1.32085541, -1.6738365 ,  0.63804616, -1.22818099,  1.21940701,
         1.27839553, -1.01694352,  0.01104715, -1.34544953],
       [ 1.32085541,  0.97115975, -0.59820399, -1.22818099, -0.01406154,
         0.78038648,  0.98333878, -1.21641433, -0.45607253],
       [-1.33965636,  0.20739793, -1.14419476, -1.22818099, -0.01406154,
         1.39816986, -1.01694352,  0.01104715,  1.32268148],
       [ 0.43401815, -1.48562948, -0.58735223, -0.00245146, -0.01406154,
        -0.34170985, -1.01694352,  0.01104715, -0.45607253]])
```

To avoid oversampling

In [11]:
```
# Apply SMOTE (Over-sampling)
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
print("Balanced Classes:", Counter(y_resampled))
```

```
Balanced Classes: Counter({0: 4001, 1: 4001})
```

Split into train /test dataset

In [12]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X_resampled, y_resampled, test_si
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (5601, 9) (5601,)
Test set: (2401, 9) (2401,)
```

Using Logistic Regression: BaseLine Model

In [13]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(class_weight='balanced').fit(X_train,y_train)
LR
```

Out[13]:
```
LogisticRegression
LogisticRegression(class_weight='balanced')
```

In [14]:
```
yhat = LR.predict(X_test)
yhat
```

Out[14]:
```
array([0, 1, 1, ..., 0, 0, 1], dtype=int64)
```

In [15]:
```python
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[15]:
```
array([[0.51450651, 0.48549349],
       [0.48403872, 0.51596128],
       [0.46184774, 0.53815226],
       ...,
       [0.54536605, 0.45463395],
       [0.55695312, 0.44304688],
       [0.48099621, 0.51900379]])
```

Accuracy

In [16]:
```python
from sklearn.metrics import f1_score
f1 = f1_score(y_test, yhat,average='weighted')
print("the f1 score of the dataset using xgboost is:", f1)
```

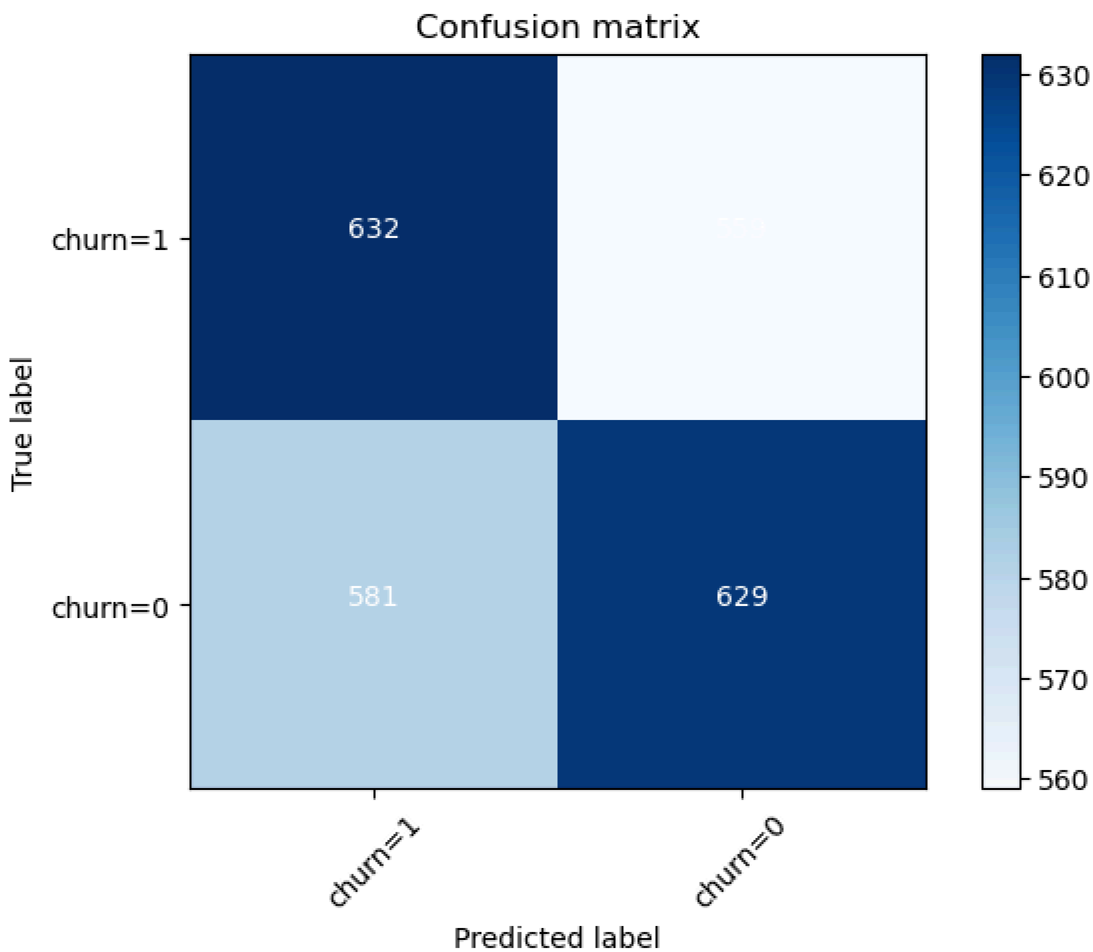
```
the f1 score of the dataset using xgboost is: 0.5251923983166351
```

In [17]:
```python
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[632 559]
 [581 629]]
```

In [18]:
```python
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  tit
```

```
Confusion matrix, without normalization
[[632 559]
 [581 629]]
```



In [19]:
```python
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.53      0.52      0.52      1210
           1       0.52      0.53      0.53      1191

    accuracy                           0.53      2401
   macro avg       0.53      0.53      0.53      2401
weighted avg       0.53      0.53      0.53      2401
```

In [20]:
```python
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

Out[20]:
```
0.6926675767869327
```

For XGBOOST : Advanced model

In [21]:
```
from xgboost import XGBClassifier
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
model.fit(X_train, y_train)

# 5 Make Predictions
y_hat_new = model.predict(X_test)
```

```
c:\Users\Owner\anaconda3\envs\geospatial\lib\site-packages\xgboost\core.py:158: UserW
arning: [21:06:52] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscali
ng-group-i-08cbc0333d8d4aae1-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

In [22]:
```
yhat_prob_new = model.predict_proba(X_test)
yhat_prob_new
```

Out[22]:
```
array([[0.45, 0.55],
       [0.4 , 0.6 ],
       [0.08, 0.92],
       ...,
       [0.16, 0.84],
       [0.79, 0.21],
       [0.35, 0.65]], dtype=float32)
```

In [23]:
```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_hat_new,average='weighted')
print("the f1 score of the dataset using xgboost is:", f1)
```

```
the f1 score of the dataset using xgboost is: 0.8322759026936165
```

In [24]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
```

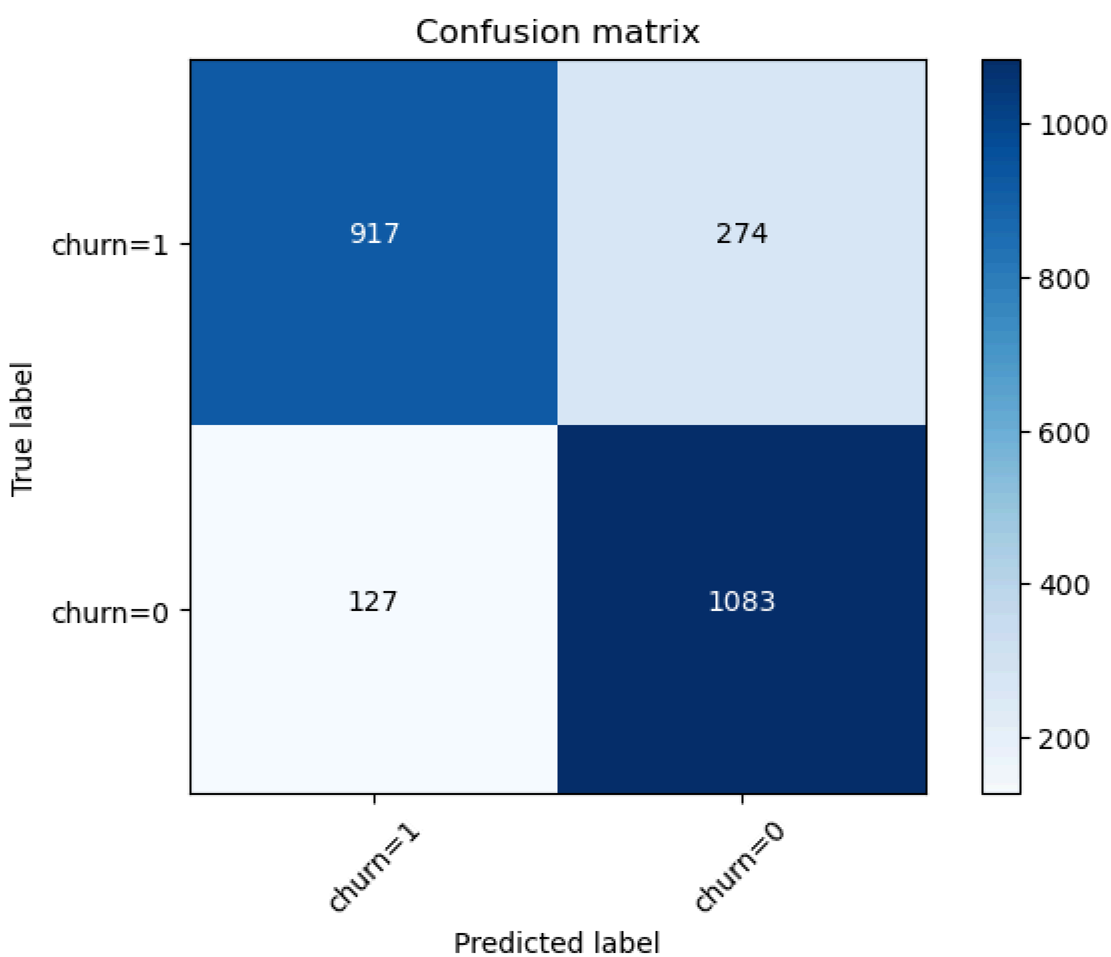
```
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[632 559]
 [581 629]]
```

In [25]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, y_hat_new, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  tit
```

```
Confusion matrix, without normalization
[[ 917  274]
 [ 127 1083]]
```



In [26]:

```
print (classification_report(y_test, y_hat_new))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.84      1210
           1       0.88      0.77      0.82      1191

    accuracy                           0.83      2401
   macro avg       0.84      0.83      0.83      2401
weighted avg       0.84      0.83      0.83      2401
```

In [27]:
```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob_new)
```

Out[27]:
```
0.38048987960747194
```