credit card default

August 6, 2020

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
pd.set_option('display.max_columns', 500)
df = pd.read_csv('UCI_Credit_Card.csv')
df.head()
```

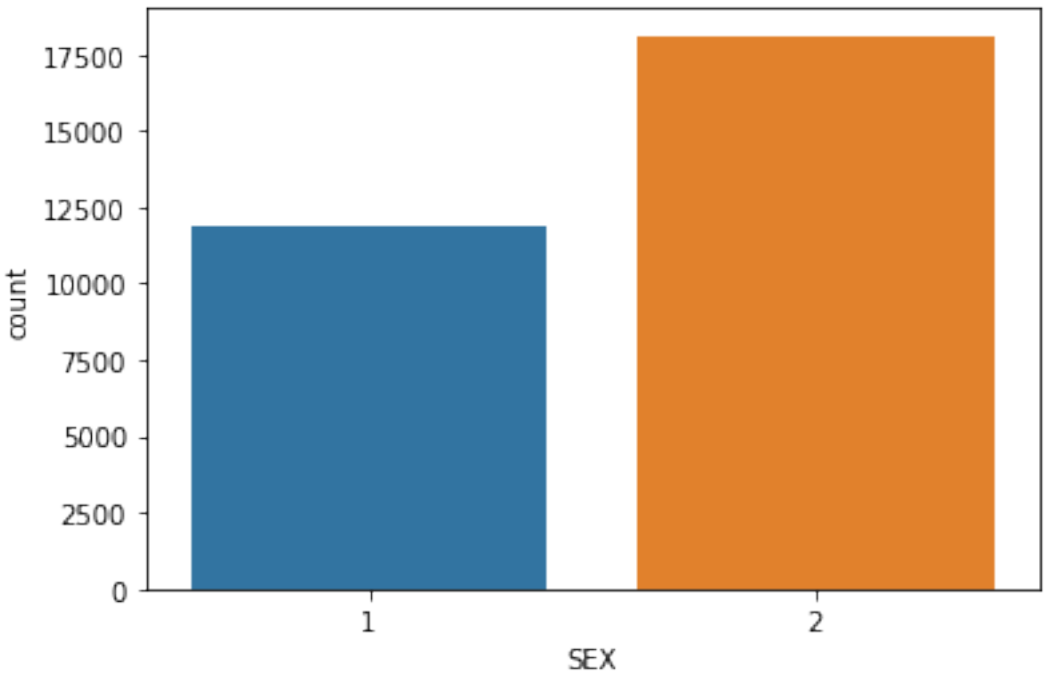
[2]:
```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2      0      0
2   3    90000.0    2          2         2   34      0      0      0      0
3   4    50000.0    2          2         1   37      0      0      0      0
4   5    50000.0    1          2         1   57     -1      0     -1      0

   PAY_5  PAY_6  BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4  BILL_AMT5  \
0     -2     -2     3913.0     3102.0      689.0        0.0        0.0
1      0      2     2682.0     1725.0     2682.0     3272.0     3455.0
2      0      0    29239.0    14027.0    13559.0    14331.0    14948.0
3      0      0    46990.0    48233.0    49291.0    28314.0    28959.0
4      0      0     8617.0     5670.0    35835.0    20940.0    19146.0

   BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6  \
0        0.0       0.0     689.0       0.0       0.0       0.0       0.0
1     3261.0       0.0    1000.0    1000.0    1000.0       0.0    2000.0
2    15549.0    1518.0    1500.0    1000.0    1000.0    1000.0    5000.0
3    29547.0    2000.0    2019.0    1200.0    1100.0    1069.0    1000.0
4    19131.0    2000.0   36681.0   10000.0    9000.0     689.0     679.0

   default.payment.next.month
0                           1
1                           1
2                           0
3                           0
4                           0
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
ID                          30000 non-null int64
LIMIT_BAL                   30000 non-null float64
SEX                         30000 non-null int64
EDUCATION                   30000 non-null int64
MARRIAGE                    30000 non-null int64
AGE                         30000 non-null int64
PAY_0                       30000 non-null int64
PAY_2                       30000 non-null int64
PAY_3                       30000 non-null int64
PAY_4                       30000 non-null int64
PAY_5                       30000 non-null int64
PAY_6                       30000 non-null int64
BILL_AMT1                   30000 non-null float64
BILL_AMT2                   30000 non-null float64
BILL_AMT3                   30000 non-null float64
BILL_AMT4                   30000 non-null float64
BILL_AMT5                   30000 non-null float64
BILL_AMT6                   30000 non-null float64
PAY_AMT1                    30000 non-null float64
PAY_AMT2                    30000 non-null float64
PAY_AMT3                    30000 non-null float64
PAY_AMT4                    30000 non-null float64
PAY_AMT5                    30000 non-null float64
PAY_AMT6                    30000 non-null float64
default.payment.next.month  30000 non-null int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

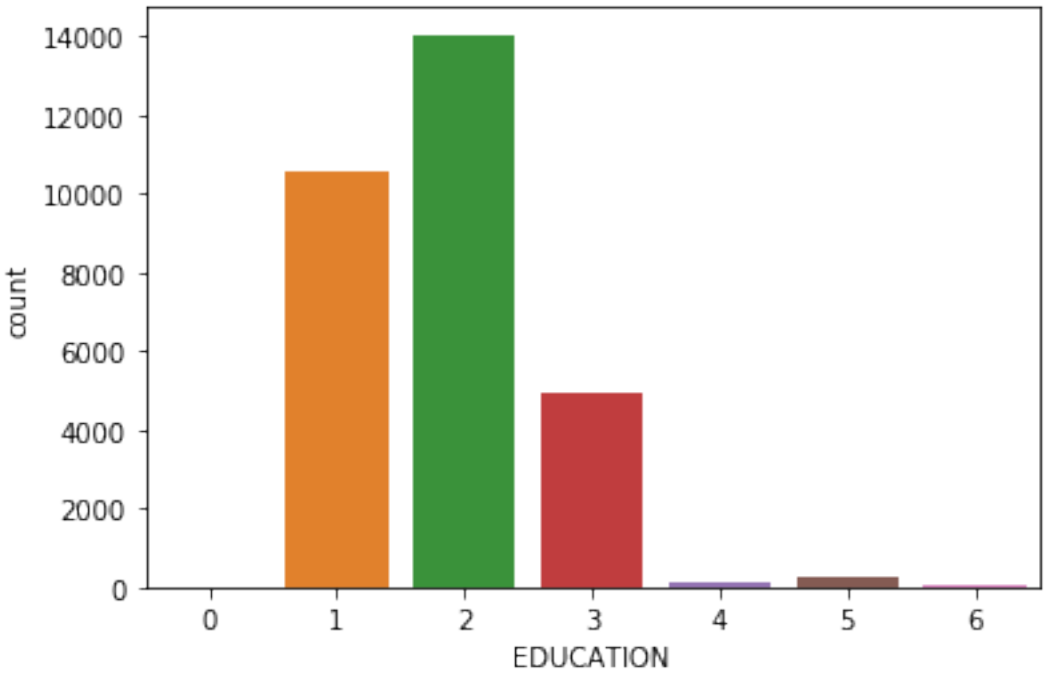
[4]:
```
sns.countplot(df['SEX'])
```

[4]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f86f830de10>`



[5]:
```
sns.countplot(df['EDUCATION'])
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f86f856a2e8>
```



[6]:
```
df['EDUCATION'] = df['EDUCATION'].replace([5, 6], 4)
col = ['PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
df[col] = df[col].replace(0, -1)
```

[7]:
```
df[df['EDUCATION'] == 0]
```

[7]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	\
3769	3770	290000.0	2	0	2	38	1	-1	-1	
5945	5946	270000.0	1	0	2	39	1	-1	-1	
6876	6877	360000.0	1	0	2	30	-1	-1	-1	
14631	14632	350000.0	2	0	2	53	-1	-1	-1	
15107	15108	210000.0	1	0	2	45	-2	-2	-2	
16881	16882	100000.0	1	0	2	37	-1	-1	-2	
16896	16897	200000.0	1	0	2	40	1	-2	-1	
17414	17415	230000.0	2	0	2	47	-1	-1	-1	
19920	19921	50000.0	2	0	1	40	-1	-1	-1	
20030	20031	200000.0	2	0	2	30	-1	-1	2	
23234	23235	220000.0	2	0	1	35	-2	-2	-2	
24137	24138	150000.0	1	0	2	28	-1	-1	-1	
27155	27156	160000.0	1	0	1	47	-1	-1	-1	
27270	27271	250000.0	1	0	1	35	-2	-2	-2	

	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	\
3769	-1	-1	-1	0.0	1437.0	3070.0	1406.0	
5945	-1	-1	-2	0.0	10193.0	69553.0	18607.0	
6876	-1	-1	-1	40250.0	23022.0	12272.0	34345.0	
14631	-1	-1	-1	5095.0	4815.0	61044.0	22611.0	
15107	-2	-2	-2	2563.0	5854.0	1032.0	788.0	
16881	-2	-2	-2	7642.0	0.0	0.0	0.0	
16896	-1	-1	-2	0.0	0.0	200.0	1000.0	
17414	2	-1	-1	8394.0	5743.0	1336.0	255.0	
19920	-1	-1	-1	44749.0	46229.0	46798.0	47647.0	
20030	-1	-1	-1	17160.0	7289.0	2868.0	9470.0	
23234	-2	-2	-2	0.0	319.0	10567.0	319.0	
24137	-1	-1	-1	15855.0	27241.0	20818.0	9864.0	
27155	-1	-1	-1	386.0	907.0	3707.0	6987.0	
27270	-2	-2	-2	22839.0	7745.0	12035.0	33604.0	

	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	\
3769	2196.0	1481.0	1437.0	3078.0	1406.0	2196.0	1481.0	
5945	0.0	0.0	10193.0	70213.0	19008.0	399.0	0.0	
6876	36777.0	30.0	23000.0	12280.0	25007.0	25008.0	1767.0	
14631	1385.0	6043.0	4840.0	61349.0	22687.0	1389.0	6058.0	
15107	3499.0	3372.0	5854.0	1032.0	788.0	3565.0	3372.0	
16881	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16896	0.0	0.0	0.0	200.0	1000.0	0.0	0.0	
17414	5425.0	4838.0	5743.0	1598.0	0.0	5425.0	4838.0	
19920	40500.0	41921.0	2229.0	2298.0	2100.0	2500.0	1921.0	

```
20030     5816.0     7809.0    2880.0       0.0    9470.0    5834.0    7809.0
23234      319.0      319.0     319.0   10567.0     319.0     319.0     319.0
24137     3957.0     2205.0   18056.0    4065.0    1058.0    3976.0    2216.0
27155     3853.0     4613.0     907.0    3707.0    6991.0      77.0    4613.0
27270        0.0     1190.0    7783.0   12046.0   33718.0       0.0    1190.0

       PAY_AMT6  default.payment.next.month
3769        0.0                           0
5945        0.0                           0
6876     3300.0                           0
14631    1153.0                           0
15107   15381.0                           0
16881       0.0                           0
16896       0.0                           0
17414    3840.0                           0
19920    8432.0                           0
20030    2886.0                           0
23234    2420.0                           0
24137       0.0                           0
27155    4099.0                           0
27270     590.0                           0
```

[8]:
```
df = df.drop(index = df[df['EDUCATION'] == 0].index, axis = 0)
df['EDUCATION'].unique()
```

[8]:
```
array([2, 1, 3, 4])
```

[9]:
```
df['LIMIT_BAL'].describe()
```

[9]:
```
count      29986.000000
mean      167461.137864
std       129760.982745
min        10000.000000
25%        50000.000000
50%       140000.000000
75%       240000.000000
max      1000000.000000
Name: LIMIT_BAL, dtype: float64
```

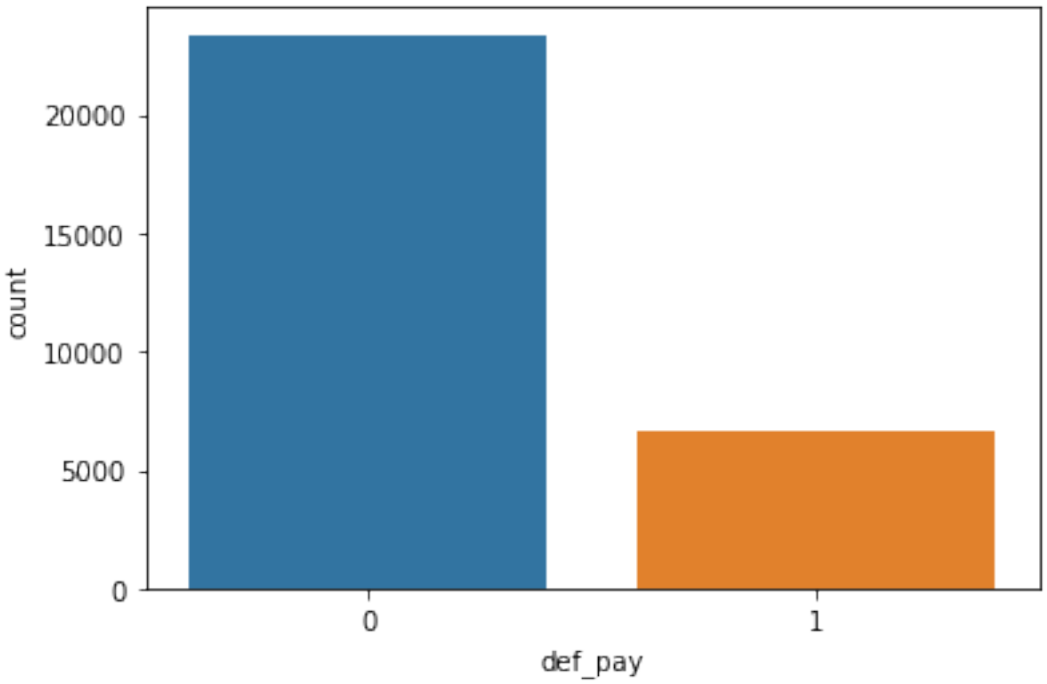
[10]:
```
df = df.rename(columns = {'default.payment.next.month': 'def_pay', 'PAY_0':␣
↪'PAY_1'})
df.head()
```

[10]:
```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_1  PAY_2  PAY_3  PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2     -1     -1
2   3    90000.0    2          2         2   34     -1     -1     -1     -1
3   4    50000.0    2          2         1   37     -1     -1     -1     -1
4   5    50000.0    1          2         1   57     -1     -1     -1     -1
```

```
   PAY_5  PAY_6  BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4  BILL_AMT5  \
0     -2     -2     3913.0     3102.0      689.0        0.0        0.0
1     -1      2     2682.0     1725.0     2682.0     3272.0     3455.0
2     -1     -1    29239.0    14027.0    13559.0    14331.0    14948.0
3     -1     -1    46990.0    48233.0    49291.0    28314.0    28959.0
4     -1     -1     8617.0     5670.0    35835.0    20940.0    19146.0

   BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6  \
0        0.0       0.0     689.0       0.0       0.0       0.0       0.0
1     3261.0       0.0    1000.0    1000.0    1000.0       0.0    2000.0
2    15549.0    1518.0    1500.0    1000.0    1000.0    1000.0    5000.0
3    29547.0    2000.0    2019.0    1200.0    1100.0    1069.0    1000.0
4    19131.0    2000.0   36681.0   10000.0    9000.0     689.0     679.0

   def_pay
0        1
1        1
2        0
3        0
4        0
```

[11]:
```
sns.countplot(df['def_pay'])
print(df['def_pay'].sum()/len(df['def_pay']))
```

```
0.2213032748616021
```



[12]:
```
df = df.reset_index(drop = True)
df.head()
```

[12]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	\
0	1	20000.0	2	2	1	24	2	2	-1	-1	
1	2	120000.0	2	2	2	26	-1	2	-1	-1	
2	3	90000.0	2	2	2	34	-1	-1	-1	-1	
3	4	50000.0	2	2	1	37	-1	-1	-1	-1	
4	5	50000.0	1	2	1	57	-1	-1	-1	-1	

	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	\
0	-2	-2	3913.0	3102.0	689.0	0.0	0.0	
1	-1	2	2682.0	1725.0	2682.0	3272.0	3455.0	
2	-1	-1	29239.0	14027.0	13559.0	14331.0	14948.0	
3	-1	-1	46990.0	48233.0	49291.0	28314.0	28959.0	
4	-1	-1	8617.0	5670.0	35835.0	20940.0	19146.0	

	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	\
0	0.0	0.0	689.0	0.0	0.0	0.0	0.0	
1	3261.0	0.0	1000.0	1000.0	1000.0	0.0	2000.0	
2	15549.0	1518.0	1500.0	1000.0	1000.0	1000.0	5000.0	
3	29547.0	2000.0	2019.0	1200.0	1100.0	1069.0	1000.0	
4	19131.0	2000.0	36681.0	10000.0	9000.0	689.0	679.0	

	def_pay
0	1
1	1
2	0
3	0
4	0

[13]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import␣
 ↪confusion_matrix,accuracy_score,classification_report
from sklearn.metrics import roc_auc_score,roc_curve,scorer, auc
from sklearn.metrics import precision_score,recall_score, f1_score
```

```
/Users/shijiecai/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:143: FutureWarning: The
sklearn.metrics.scorer module is  deprecated in version 0.22 and will be removed
in version 0.24. The corresponding classes / functions should instead be
imported from sklearn.metrics. Anything that cannot be imported from
sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[14]:
```
y = df['def_pay'].copy()
X=df.drop(['ID'], axis = 1)
```

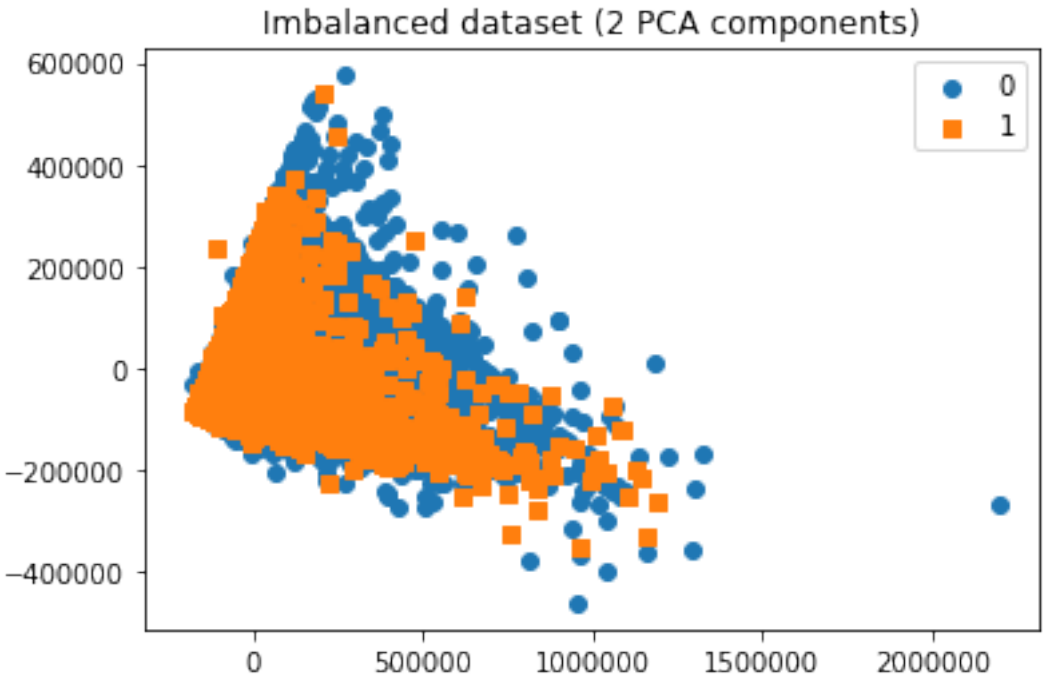
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42)
```

[15]:
```
log = LogisticRegression()
log.fit(X_train, y_train)
y_pred = log.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.87      4660
           1       0.00      0.00      0.00      1338

    accuracy                           0.78      5998
   macro avg       0.39      0.50      0.44      5998
weighted avg       0.60      0.78      0.68      5998


/Users/shijiecai/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_logistic.py:764: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/shijiecai/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[16]:
```
### use pca to check how the 2 classes are distributed
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

def plot_2d_space(X, y, label='Classes'):
    colors = ['#1F77B4', '#FF7F0E']
    markers = ['o', 's']
    for l, c, m in zip(np.unique(y), colors, markers):
        plt.scatter(
            X[y==l, 0],
```

```
            X[y==l, 1],
            c=c, label=l, marker=m
        )
    plt.title(label)
    plt.legend(loc='upper right')
    plt.show()


plot_2d_space(X_pca, y_train, 'Imbalanced dataset (2 PCA components)')
```



[35]:
```
colors = ['#1F77B4', '#FF7F0E']
markers = ['o', 's']
for l, c, m in zip(np.unique(y_train), colors, markers):
    print(l, c, m)
```

```
0 #1F77B4 o
1 #FF7F0E s
```

[38]:
```
n = [1 ,2, 3]
l = ['a', 'b', 'c']
zip(n, l)
list(zip(n, l))

for i, j in zip(n, l):
```

```
    print(i, j)
```

```
1 a
2 b
3 c
```

[ ]:

[25]:
```
X_pca[y_train == 1, 1].shape
```

[25]:
```
(5298,)
```

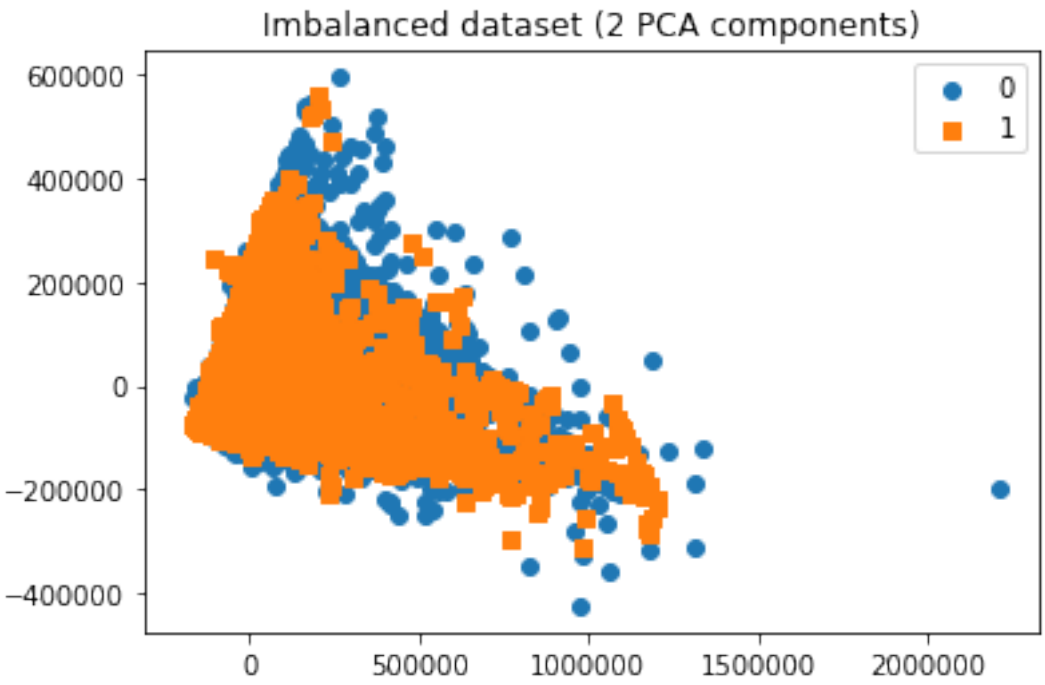
[19]:
```
y_train == 1
```

[19]:
```
17557    False
618      False
17084    False
22586    False
26947    False
29423    False
15865    False
9687     False
18671    False
16278    False
8761     False
8704      True
29138     True
10992    False
17455     True
22688    False
5403      True
23866    False
1474      True
18363    False
8649     False
27373    False
21614    False
10366     True
18502    False
1713     False
7228     False
3995      True
465       True
20526    False
         ...
20939    False
17568    False
6420     False
5051     False
```

```
5311     False
2433     False
23333    False
26967    False
769      False
1685     False
8322     False
16023     True
27495     True
11363    False
28020    False
14423    False
21962     True
4426     False
29910    False
16850    False
6265     False
22118    False
11284    False
11964    False
21575    False
29802    False
5390     False
860      False
15795    False
23654    False
Name: def_pay, Length: 23988, dtype: bool
```

[17]:
```
### using smote for unbalanced data
from imblearn.over_sampling import SMOTE

os = SMOTE(sampling_strategy = 'minority', random_state = 0, )
X_train_os, y_train_os = os.fit_sample(X_train, y_train)
```

[18]:
```
X_pca = pca.fit_transform(X_train_os)

plot_2d_space(X_pca, y_train_os, 'Imbalanced dataset (2 PCA components)')
```



[19]:
```
log = LogisticRegression()
log.fit(X_train_os, y_train_os)
y_pred = log.predict(X_test)
print(classification_report(y_test, y_pred))
```

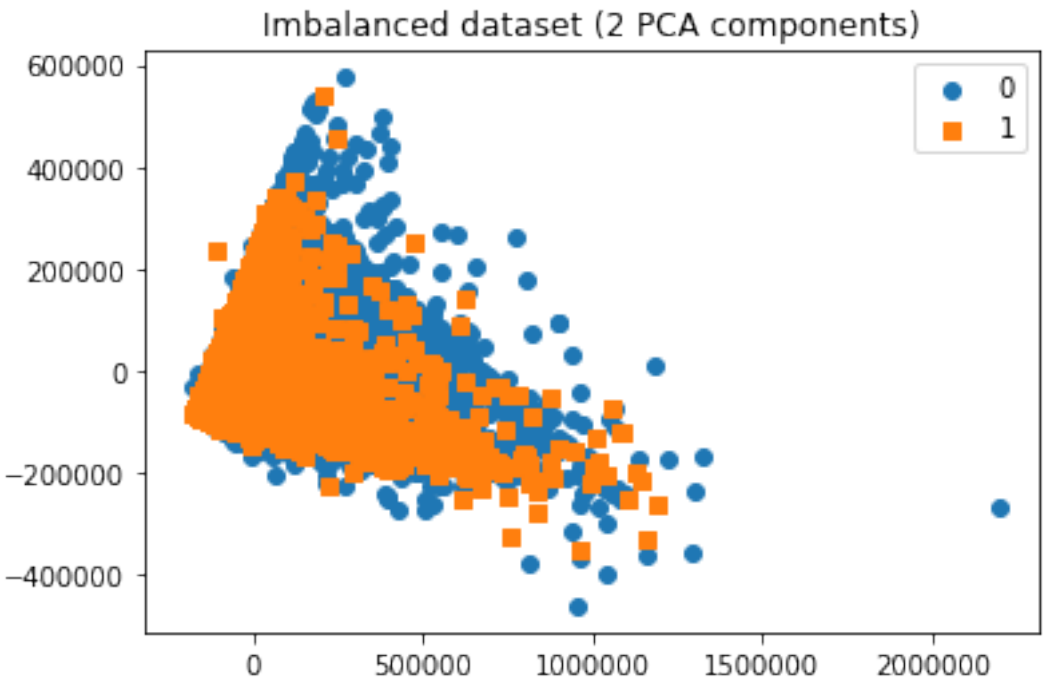
```
              precision    recall  f1-score   support

           0       0.86      0.53      0.65      4660
           1       0.30      0.69      0.41      1338

    accuracy                           0.57      5998
   macro avg       0.58      0.61      0.53      5998
weighted avg       0.73      0.57      0.60      5998


/Users/shijiecai/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_logistic.py:764: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[20]:
```
## cleaning

df['Closeness_6'] = (df.LIMIT_BAL - df.BILL_AMT6) / df.LIMIT_BAL
df['Closeness_5'] = (df.LIMIT_BAL - df.BILL_AMT5) / df.LIMIT_BAL
df['Closeness_4'] = (df.LIMIT_BAL - df.BILL_AMT4) / df.LIMIT_BAL
df['Closeness_3'] = (df.LIMIT_BAL - df.BILL_AMT3) / df.LIMIT_BAL
df['Closeness_2'] = (df.LIMIT_BAL - df.BILL_AMT2) / df.LIMIT_BAL
df['Closeness_1'] = (df.LIMIT_BAL - df.BILL_AMT1) / df.LIMIT_BAL
```

[22]:
```
df['Avg_exp_5'] = ((df['BILL_AMT5'] - (df['BILL_AMT6'] - df['PAY_AMT5']))) /␣
 ↪df['LIMIT_BAL']
df['Avg_exp_4'] = (((df['BILL_AMT5'] - (df['BILL_AMT6'] - df['PAY_AMT5'])) +
                   (df['BILL_AMT4'] - (df['BILL_AMT5'] - df['PAY_AMT4']))) / 2) /␣
 ↪df['LIMIT_BAL']
df['Avg_exp_3'] = (((df['BILL_AMT5'] - (df['BILL_AMT6'] - df['PAY_AMT5'])) +
                   (df['BILL_AMT4'] - (df['BILL_AMT5'] - df['PAY_AMT4'])) +
                   (df['BILL_AMT3'] - (df['BILL_AMT4'] - df['PAY_AMT3']))) / 3) /␣
 ↪df['LIMIT_BAL']
df['Avg_exp_2'] = (((df['BILL_AMT5'] - (df['BILL_AMT6'] - df['PAY_AMT5'])) +
                   (df['BILL_AMT4'] - (df['BILL_AMT5'] - df['PAY_AMT4'])) +
                   (df['BILL_AMT3'] - (df['BILL_AMT4'] - df['PAY_AMT3'])) +
                   (df['BILL_AMT2'] - (df['BILL_AMT3'] - df['PAY_AMT2']))) / 4) /␣
 ↪df['LIMIT_BAL']
df['Avg_exp_1'] = (((df['BILL_AMT5'] - (df['BILL_AMT6'] - df['PAY_AMT5'])) +
                   (df['BILL_AMT4'] - (df['BILL_AMT5'] - df['PAY_AMT4'])) +
                   (df['BILL_AMT3'] - (df['BILL_AMT4'] - df['PAY_AMT3'])) +
                   (df['BILL_AMT2'] - (df['BILL_AMT3'] - df['PAY_AMT2'])) +
                   (df['BILL_AMT1'] - (df['BILL_AMT2'] - df['PAY_AMT1']))) / 5) /␣
 ↪df['LIMIT_BAL']
```

[24]:
```
features = ['LIMIT_BAL', 'EDUCATION', 'MARRIAGE', 'PAY_1','PAY_2', 'PAY_3',
            'PAY_4', 'PAY_5', 'PAY_6','BILL_AMT1', 'BILL_AMT2',
            'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
            'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', ␣
 ↪'Avg_exp_5', 'Avg_exp_4',
            'Avg_exp_3', 'Avg_exp_2', 'Avg_exp_1', 'Closeness_5',
            'Closeness_4', 'Closeness_3', 'Closeness_2','Closeness_1']
y = df['def_pay'].copy() # target
X = df[features].copy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=42)
```

[26]:
```
X_pca = pca.fit_transform(X_train)

plot_2d_space(X_pca, y_train, 'Imbalanced dataset (2 PCA components)')
```



[27]:
```
from sklearn.ensemble import RandomForestClassifier , GradientBoostingClassifier
from xgboost import XGBClassifier
```

[28]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      4660
           1       0.64      0.36      0.46      1338

    accuracy                           0.81      5998
   macro avg       0.74      0.65      0.67      5998
weighted avg       0.79      0.81      0.79      5998
```

[41]:
```
from sklearn.model_selection import GridSearchCV

param = {
    'n_estimators' : [200],
    'max_features' : ['sqrt'],
    'max_depth' : [None]
}
```

```
grid_rf = GridSearchCV(rf, param_grid = param, scoring = 'accuracy', cv=5)
grid_rf.fit(X_train_os, y_train_os)
```

[41]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(max_features='sqrt',
                                              n_estimators=200),
             param_grid={'max_depth': [None], 'max_features': ['sqrt'],
                         'n_estimators': [200]},
             scoring='accuracy')
```

[37]:
```
grid_rf.best_params_
```

[37]:
```
{'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200}
```

[40]:
```
param = {
    'n_estimators' : 200,
    'max_features' : 'sqrt',
    'max_depth' : None
}

rf = RandomForestClassifier(**param)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      4660
           1       0.65      0.37      0.47      1338

    accuracy                           0.81      5998
   macro avg       0.74      0.66      0.68      5998
weighted avg       0.80      0.81      0.79      5998
```

[43]:
```
os = SMOTE(sampling_strategy = 'minority', random_state = 0, )
X_train_os, y_train_os = os.fit_sample(X_train, y_train)

rf = RandomForestClassifier(**param)
rf.fit(X_train_os, y_train_os)
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      4660
           1       0.55      0.45      0.50      1338
```

```
    accuracy                           0.80      5998
   macro avg       0.70      0.67      0.68      5998
weighted avg       0.78      0.80      0.79      5998
```

[44]:
```
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      4660
           1       0.66      0.36      0.47      1338

    accuracy                           0.82      5998
   macro avg       0.75      0.65      0.68      5998
weighted avg       0.80      0.82      0.80      5998
```

[45]:
```
gb.fit(X_train_os, y_train_os)
y_pred = gb.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.87      4660
           1       0.53      0.49      0.51      1338

    accuracy                           0.79      5998
   macro avg       0.69      0.68      0.69      5998
weighted avg       0.78      0.79      0.79      5998
```

[ ]:
```

```