In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
import seaborn as sns
from pandas.plotting import scatter_matrix

from sklearn.model_selection import train_test_split
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix

from imblearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

from numpy import mean
from numpy import var
import time
from sklearn.model_selection import cross_validate
import shap

from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import ADASYN
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import BorderlineSMOTE
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import cohen_kappa_score
from math import sqrt
from sklearn.metrics import matthews_corrcoef
```

匯入資料

In [2]:
```
data=pd.read_csv("data-question/train.csv")
data.head()
```

Out[2]:

	ID	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	E
0	8773	0	0.000000	0	0.0	1	0.000000	0.200000	0
1	6709	0	0.000000	0	0.0	1	0.000000	0.200000	0
2	1463	9	301.000000	0	0.0	38	2621.621429	0.021212	0
3	4095	2	13.333333	0	0.0	105	2062.443592	0.012205	0
4	3346	0	0.000000	0	0.0	19	220.384849	0.010526	0

確認資料

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8100 entries, 0 to 8099
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       8100 non-null   int64  
 1   Administrative           8100 non-null   int64  
 2   Administrative_Duration  8100 non-null   float64
 3   Informational            8100 non-null   int64  
 4   Informational_Duration   8100 non-null   float64
 5   ProductRelated           8100 non-null   int64  
 6   ProductRelated_Duration  8099 non-null   float64
 7   BounceRates              8100 non-null   float64
 8   ExitRates                8100 non-null   float64
 9   PageValues               8100 non-null   float64
 10  SpecialDay               8100 non-null   float64
 11  Month                    8100 non-null   int64  
 12  OperatingSystems         8099 non-null   float64
 13  Browser                  8099 non-null   float64
 14  Region                   8099 non-null   float64
 15  TrafficType              8099 non-null   float64
 16  VisitorType              8099 non-null   float64
 17  Weekend                  8100 non-null   int64  
 18  Revenue                  8100 non-null   int64  
dtypes: float64(12), int64(7)
memory usage: 1.2 MB
```

In [4]:
```
#尋找相關性
#使用corr()計算每一對屬性之間的標準相關性係數
corr_matrix = data.corr()

#查看每一個屬性與是否訂房之間的相關性有多大
corr_matrix['Revenue'].sort_values(ascending=False)
```

Out[4]:
```
Revenue                    1.000000
PageValues                 0.499500
ProductRelated             0.149177
Administrative             0.142747
Month                      0.127990
Administrative_Duration    0.095692
Informational              0.091298
Informational_Duration     0.068661
Weekend                    0.003893
ID                        -0.001342
ProductRelated_Duration   -0.005530
Region                    -0.010402
Browser                   -0.018902
BounceRates               -0.050082
OperatingSystems          -0.070977
SpecialDay                -0.076422
TrafficType               -0.088077
VisitorType               -0.111334
ExitRates                 -0.207791
Name: Revenue, dtype: float64
```

確認遺失值

In [5]:
```
data.isnull().sum()
```

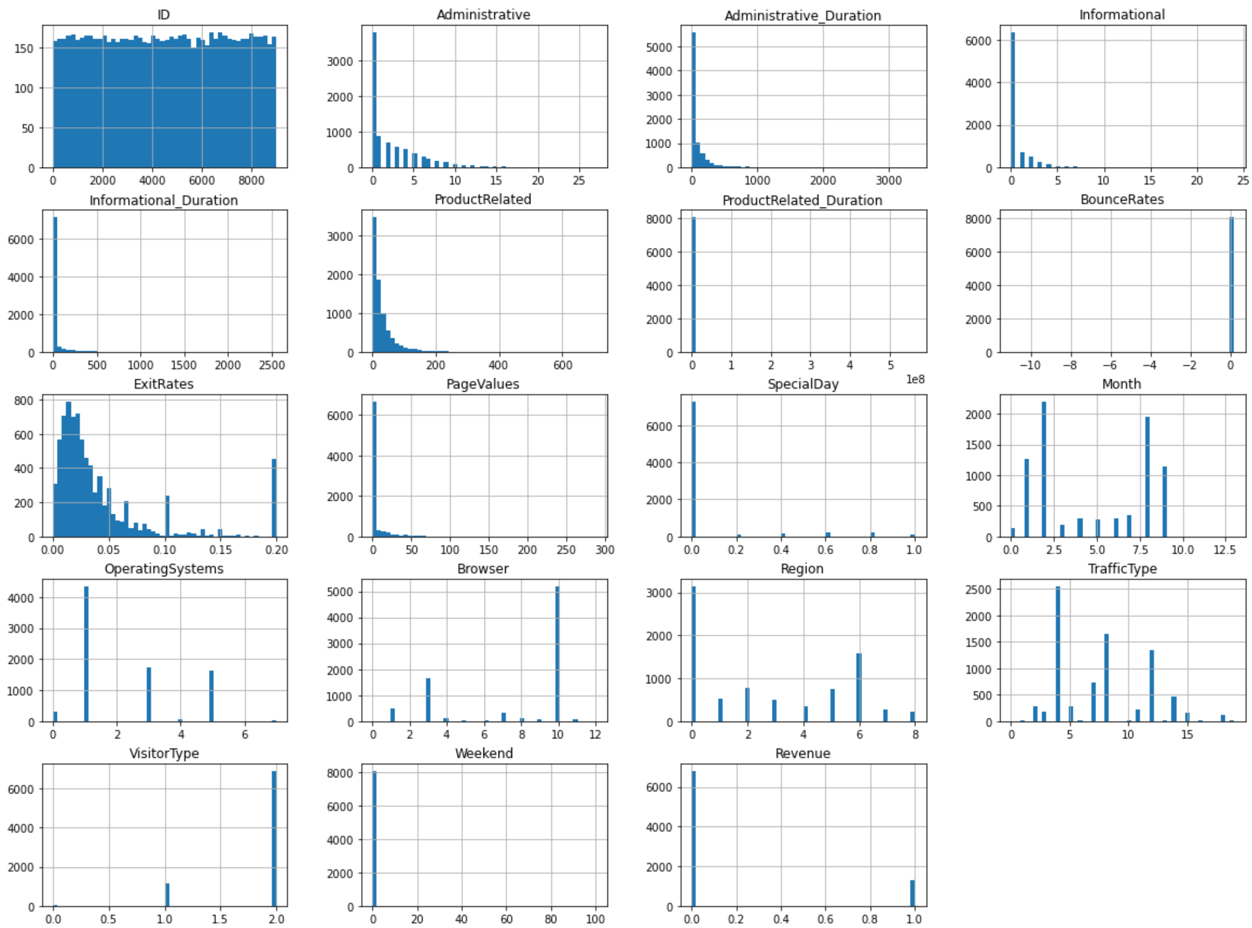
Out[5]:
```
ID                         0
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    1
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           1
Browser                    1
Region                     1
TrafficType                1
VisitorType                1
Weekend                    0
Revenue                    0
dtype: int64
```

In [6]:
```
#查看其他欄位的數值屬性摘要
data.describe()
```

Out[6]:

	ID	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	Boun
count	8100.000000	8100.000000	8100.000000	8100.000000	8100.000000	8100.000000	8.099000e+03	810(
mean	4500.375432	2.309877	80.926113	0.498025	32.884300	31.787160	8.311549e+04	(
std	2601.276244	3.311618	180.089694	1.258087	135.210888	44.961092	6.359096e+06	(
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	-11
25%	2237.750000	0.000000	0.000000	0.000000	0.000000	7.000000	1.837708e+02	(
50%	4504.500000	1.000000	7.000000	0.000000	0.000000	18.000000	5.988738e+02	(
75%	6760.250000	4.000000	91.988636	0.000000	0.000000	38.000000	1.462142e+03	(
max	8999.000000	27.000000	3398.750000	24.000000	2549.375000	705.000000	5.634924e+08	(

In [7]:
```
#將每一數值屬性畫出直方圖
%matplotlib inline
data.hist(bins=50, figsize=(20,15))
plt.show()
```



處理遺失值

In [8]:
```python
df = data.dropna(axis=0)
## 確認遺失值
df.isnull().sum()
```

Out[8]:
```
ID                         0
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

Split the Features (X) and Target (Y)

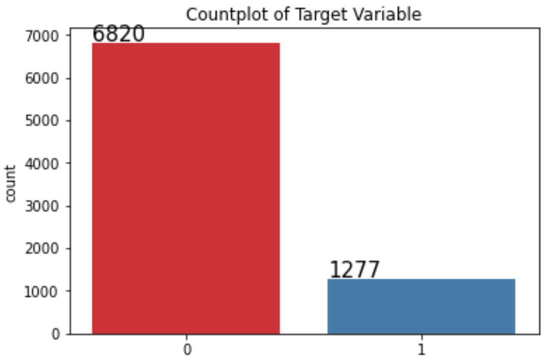
In [9]:
```python
y = df['Revenue'].values
X = df.drop(['ID','Revenue'],axis=1)
```

In [10]:
```python
# See the relationshipt between Classes
from collections import Counter
import seaborn as sns
counter = Counter(y)
print(counter)
digit_count = sns.countplot(y,palette="Set1")
plt.title("Countplot of Target Variable")

for p in digit_count.patches:
    digit_count.annotate(f'\n{p.get_height()}', (p.get_x(), p.get_height()+50), color='black', size=15)
plt.show()
```

```
Counter({0: 6820, 1: 1277})
```

```
Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```



可由上圖看到資料為不平衡資料 18.72%

In [11]:
```
def get_data(c):
    TN = c[0,0]
    FP = c[0,1]
    FN = c[1,0]
    TP = c[1,1]
    return TP, FP, FN, TN

def cohen_kappa(TP, FP, FN, TN):
    p_0 = (TN + TP) / (TN + FP + FN + TP)
    p_c = ((TN+FN)*(TN+FP) + (FN+TP)*(FP+TP)) / (TN + FP + FN + TP)**2
    kappa = (p_0 - p_c) / (1-p_c)
    return kappa
## cohen_kappa_score(y_pred, y_truth)

def mcc(TP, FP, FN, TN):
    numerator = (TP * TN) - (FP * FN)
    denominator = sqrt((TP+FP) * (TP+FN) * (TN+FP) * (TN+FN))
    mcc = numerator / denominator
    return mcc
## matthews_corrcoef(y_truth, y_pred)
```

In [12]:
```
## 用於合併多餘類別項目
def new_month(value):
  if value == 1:
    return 1
  elif value == 2:
    return 2
  elif value == 8:
    return 3
  elif value == 9:
    return 4
  else:
    return 0

def new_OperatingSystems(value):
  if value == 1:
    return 1
  elif value == 3:
    return 2
  elif value == 5:
    return 3
  else:
    return 0

def new_browser(value):
  if value == 10:
    return 0
  elif value == 3:
    return 1
  else:
    return 2

def new_TrafficType(value):
  if value == 4:
    return 1
  elif value == 8:
    return 2
  elif value == 12:
    return 3
  elif value == 7:
    return 4
  elif value == 14:
    return 5
  elif value == 2:
    return 5
  elif value == 5:
    return 6
  elif value == 11:
    return 6
  elif value == 3:
    return 6
  else:
    return 0
```

分割訓練集 / 測試集 80/20

In [13]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Case 1 :不做資料前處理

In [14]:
```
xgbc =XGBClassifier()
xgbc.fit(x_train , y_train)
xgbc.score(x_train, y_train), xgbc.score(x_test , y_test)
```

```
[05:45:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.


The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

Out[14]:
```
(0.9936699089084453, 0.8851851851851852)
```

In [15]:
```
y_pred = xgbc.predict(x_test)
recall_test = recall_score(np.array(y_test), y_pred, average=None)
precision_test = precision_score(np.array(y_test), y_pred, average=None)
f1_score_test = f1_score(np.array(y_test), y_pred, average=None)
c=confusion_matrix(y_test,y_pred)
c
```

Out[15]:
```
array([[1289,   60],
       [ 126,  145]], dtype=int64)
```

In [16]:
```
recall_test, precision_test,f1_score_test
```

Out[16]:
```
(array([0.95552261, 0.53505535]),
 array([0.91095406, 0.70731707]),
 array([0.93270622, 0.6092437 ]))
```

In [17]:
```
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

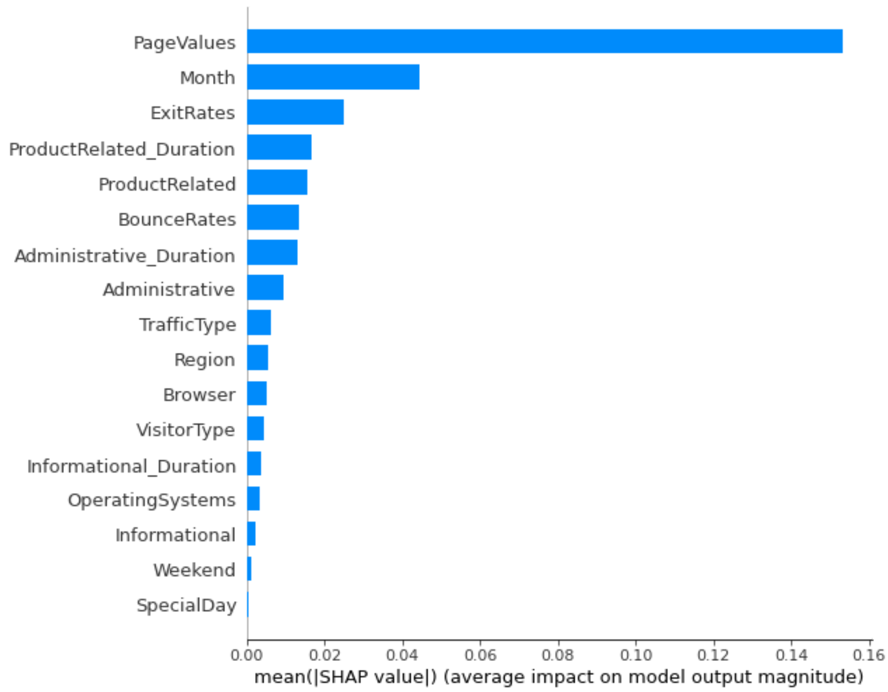
Out[17]:
```
0.8851851851851852
```

In [18]:
```
TP, FP, FN, TN = get_data(c)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[18]:
```
(0.5434614627051106, 0.5507360465234322)
```

In [19]:
```
# 特徵重要程度
explainer = shap.TreeExplainer(xgbc,x_train, model_output='probability')
shap_values = explainer.shap_values(x_train)
shap.summary_plot(shap_values,x_train,plot_type="bar")
```

```
 97%|=================== | 6261/6477 [00:17<00:00]
```



In [20]:
```
## 特徵重要程度占比
shap_value_0 = pd.DataFrame(shap_values)
shap_value_1_0 = abs(shap_value_0)
shap_value_col_1 = np.sum(shap_value_1_0 , axis = 0)/len(shap_values)
shap_value_col_1 = shap_value_col_1.sort_values()
p = shap_value_col_1/sum(shap_value_col_1)*100
p.sort_values(ascending=False)
```

Out[20]:
```
8     47.418551
10    13.737964
7      7.770030
5      5.164457
4      4.806718
6      4.196754
1      4.032485
0      2.867653
14     1.969275
13     1.706412
12     1.549552
15     1.396531
3      1.182401
11     0.973131
2      0.687917
16     0.366023
9      0.174147
dtype: float64
```

特徵處理

類別型特徵處理

刪除異常值

在前面可以發現類別型特徵裡，Month多了13月、weekend多了100(一般只有0or1)

In [21]:
```
data2 = df[-df.Month.isin([13])]
data3 = data2[-data2.Weekend.isin([100])]
data3
```

Out[21]:

	ID	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates
0	8773	0	0.000000	0	0.0	1	0.000000	0.200000
1	6709	0	0.000000	0	0.0	1	0.000000	0.200000
2	1463	9	301.000000	0	0.0	38	2621.621429	0.021212
3	4095	2	13.333333	0	0.0	105	2062.443592	0.012205
4	3346	0	0.000000	0	0.0	19	220.384849	0.010526
...	...	...	...	...	...	...	...	...
8095	3758	0	0.000000	0	0.0	4	81.000000	0.000000
8096	4437	1	15.200000	2	62.6	84	4941.698611	0.017647
8097	7449	0	0.000000	0	0.0	25	701.883333	0.000000
8098	665	9	183.785714	1	90.0	95	3346.501984	0.002118
8099	552	3	49.750000	0	0.0	20	547.400794	0.000000

8095 rows × 19 columns

對類別型特徵做One-hot-encoding

Month、OperatingSystems、Browser、Region、TrafficType、VisitorType、Weekend

In [22]:
```
df_str = data3.astype({'Month':'category','OperatingSystems':'category','Browser':'category','Region':'category','Traf
df_str
df_dum = pd.get_dummies(df_str[['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','Weekend']])

df_str.drop(['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','Weekend'], axis=1, inplace=Tru

df_new = pd.concat([df_dum,df_str],axis=1)
df_new
```

Out[22]:

	Month_0	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8	Month_9	...	Administrative_Duration	Information
0	0	0	1	0	0	0	0	0	0	0	...	0.000000	
1	0	0	0	0	0	0	0	0	0	1	...	0.000000	
2	0	0	0	0	0	0	0	0	1	0	...	301.000000	
3	0	0	1	0	0	0	0	0	0	0	...	13.333333	
4	0	0	1	0	0	0	0	0	0	0	...	0.000000	
...	...	...	...	...	...	...	...	...	...	...	...	...	
8095	0	0	1	0	0	0	0	0	0	0	...	0.000000	
8096	0	0	0	0	0	0	1	0	0	0	...	15.200000	
8097	0	0	0	0	0	0	1	0	0	0	...	0.000000	
8098	0	0	0	0	0	0	0	0	1	0	...	183.785714	
8099	0	0	0	0	0	0	0	0	1	0	...	49.750000	

8095 rows × 77 columns

Split x and y

In [23]:
```
y_ohe=df_new['Revenue'].values
X_ohe=df_new.drop(['ID','Revenue'],axis=1)
```

分割訓練集 / 測試集 80/20

In [24]:
```
x_train_ohe, x_test_ohe, y_train_ohe, y_test_ohe = train_test_split(X_ohe, y_ohe, test_size=0.2, random_state=1)
```

數值型特徵標準化

Administrative、Administrative_Duration、Informational、nformational_Duration、ProductRelated、ProductRelated_Duration、BounceRates 、ExitRates、PageValues、SpecialDay為連續型特徵，但區間過大，需先做標準化，以利於再做PCA

StandardScaler適用於本身服從常態分佈的數據，與本題不符
MinMaxScaler適用於分布範圍較穩定的數據，亦與本題不符
故採用RobustScaler適用於包含許多異常值的數據，可弱化Outlier的影響

In [25]:
```
rob_scaler = RobustScaler()
rob_data = x_train_ohe.copy()


rob_data['rob_scaled_Administrative'] = rob_scaler.fit_transform(rob_data['Administrative'].values.reshape(-1,1))
rob_data['rob_scaled_Administrative_Duration'] = rob_scaler.fit_transform(rob_data['Administrative_Duration'].values.r
rob_data['rob_scaled_Informational'] = rob_scaler.fit_transform(rob_data['Informational'].values.reshape(-1,1))
rob_data['rob_scaled_Informational_Duration'] = rob_scaler.fit_transform(rob_data['Informational_Duration'].values.res
rob_data['rob_scaled_ProductRelated'] = rob_scaler.fit_transform(rob_data['ProductRelated'].values.reshape(-1,1))
rob_data['rob_scaled_ProductRelated_Duration'] = rob_scaler.fit_transform(rob_data['ProductRelated_Duration'].values.r
rob_data['rob_scaled_BounceRates'] = rob_scaler.fit_transform(rob_data['BounceRates'].values.reshape(-1,1))
rob_data['rob_scaled_ExitRates'] = rob_scaler.fit_transform(rob_data['ExitRates'].values.reshape(-1,1))
rob_data['rob_scaled_PageValues'] = rob_scaler.fit_transform(rob_data['PageValues'].values.reshape(-1,1))
rob_data['rob_scaled_SpecialDay'] = rob_scaler.fit_transform(rob_data['SpecialDay'].values.reshape(-1,1))




rob_data.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelated','P

rob_data.describe()
```

Out[25]:

	Month_0	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8	Month_9	...
count	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	...
mean	0.015596	0.157505	0.274707	0.020692	0.035825	0.034898	0.034744	0.045244	0.237492	0.143298	...
std	0.123916	0.364304	0.446401	0.142361	0.185867	0.183536	0.183144	0.207855	0.425579	0.350404	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

8 rows × 75 columns

Case 2 : 做完資料處理 no PCA

In [26]:
```
xgbc_2 =XGBClassifier()
xgbc_2.fit(rob_data , y_train_ohe)
xgbc_2.score(rob_data , y_train_ohe)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:45:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[26]: 0.993514515132798

In [27]:
```
rob_scaler = RobustScaler()
rob_data_test = x_test_ohe.copy()


rob_data_test['rob_scaled_Administrative'] = rob_scaler.fit_transform(rob_data_test['Administrative'].values.reshape(-
rob_data_test['rob_scaled_Administrative_Duration'] = rob_scaler.fit_transform(rob_data_test['Administrative_Duration'
rob_data_test['rob_scaled_Informational'] = rob_scaler.fit_transform(rob_data_test['Informational'].values.reshape(-1,
rob_data_test['rob_scaled_Informational_Duration'] = rob_scaler.fit_transform(rob_data_test['Informational_Duration'].
rob_data_test['rob_scaled_ProductRelated'] = rob_scaler.fit_transform(rob_data_test['ProductRelated'].values.reshape(-
rob_data_test['rob_scaled_ProductRelated_Duration'] = rob_scaler.fit_transform(rob_data_test['ProductRelated_Duration'
rob_data_test['rob_scaled_BounceRates'] = rob_scaler.fit_transform(rob_data_test['BounceRates'].values.reshape(-1,1))
rob_data_test['rob_scaled_ExitRates'] = rob_scaler.fit_transform(rob_data_test['ExitRates'].values.reshape(-1,1))
rob_data_test['rob_scaled_PageValues'] = rob_scaler.fit_transform(rob_data_test['PageValues'].values.reshape(-1,1))
rob_data_test['rob_scaled_SpecialDay'] = rob_scaler.fit_transform(rob_data_test['SpecialDay'].values.reshape(-1,1))




rob_data_test.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelate

rob_data_test.describe()
```

Out[27]:

	Month_0	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8	Month_9	...
count	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	...
mean	0.018530	0.151328	0.261272	0.034589	0.034589	0.035825	0.040148	0.038295	0.254478	0.130945	...
std	0.134899	0.358479	0.439464	0.182794	0.182794	0.185910	0.196367	0.191967	0.435702	0.337444	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	...
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

8 rows × 75 columns

In [28]:
```
y_pred_2 = xgbc_2.predict(rob_data_test)
recall_test_2 = recall_score(np.array(y_test_ohe), y_pred_2, average=None)
precision_test_2 = precision_score(np.array(y_test_ohe), y_pred_2, average=None)
f1_score_test_2 = f1_score(np.array(y_test_ohe), y_pred_2, average=None)
c2=confusion_matrix(y_test_ohe,y_pred_2)
c2
```

Out[28]:
```
array([[1288,   51],
       [ 115,  165]], dtype=int64)
```

In [29]:
```
recall_test_2, precision_test_2,f1_score_test_2
```

Out[29]:
```
(array([0.96191187, 0.58928571]),
 array([0.91803279, 0.76388889]),
 array([0.93946025, 0.66532258]))
```

In [30]:
```
accuracy_2 = accuracy_score(y_test_ohe,y_pred_2)
accuracy_2
```

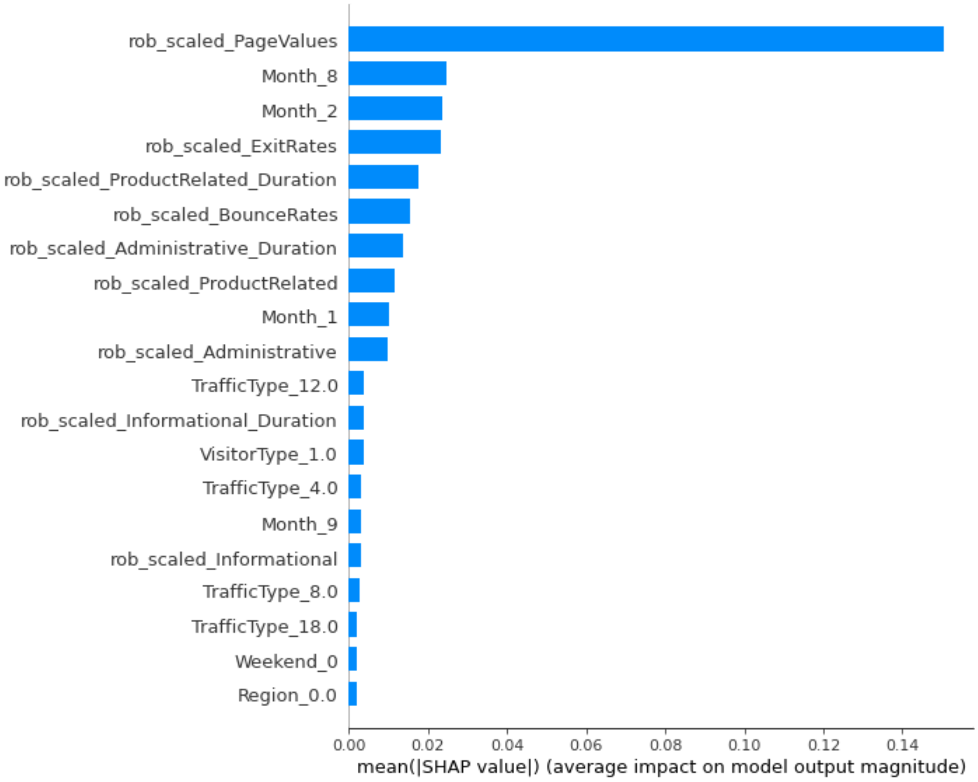
Out[30]:
```
0.897467572575664
```

In [31]:
```
TP, FP, FN, TN = get_data(c2)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[31]:
```
(0.6059695277862489, 0.6130852986739118)
```

In [32]:
```
# 特徵重要程度
explainer_1 = shap.TreeExplainer(xgbc_2,rob_data, model_output='probability')
shap_values_1 = explainer_1.shap_values(rob_data)
shap.summary_plot(shap_values_1,rob_data,plot_type="bar")
```

```
 98%|===================| 6345/6476 [00:18<00:00]
```



In [33]:
```
shap_value_2 = pd.DataFrame(shap_values_1)
shap_value_1_0 = abs(shap_value_2)
shap_value_col_1 = np.sum(shap_value_1_0 , axis = 0)/len(shap_values_1)
shap_value_col_1 = shap_value_col_1.sort_values()
p = shap_value_col_1/sum(shap_value_col_1)*100
p.sort_values(ascending=False)
```

Out[33]:
```
73    41.722780
8      6.824459
2      6.606756
72     6.508299
70     4.896873
         ...
53     0.000000
59     0.000000
12     0.000000
56     0.000000
18     0.000000
Length: 75, dtype: float64
```

Case8: 做完資料處理 + Imbalanced data處理

In [34]:
```
# summarize class distribution
counter = Counter(y_train_ohe)
print(counter)
# define pipeline
over = ADASYN(sampling_strategy=1.0, random_state=1)
#under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over)]
pipeline = Pipeline(steps=steps)
# transform the dataset
OU_X_8, OU_y_8 = pipeline.fit_resample(rob_data, y_train_ohe)
# summarize the new class distribution
counter = Counter(OU_y_8)
print(counter)
```

```
Counter({0: 5479, 1: 997})
Counter({1: 5581, 0: 5479})
```

In [35]:
```
xgbc_8 =XGBClassifier()
xgbc_8.fit(OU_X_8 , OU_y_8)
xgbc_8.score(OU_X_8 , OU_y_8)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:46:16] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[35]:
```
0.9944846292947559
```

In [36]:
```
xgbc_8.score(rob_data_test , y_test_ohe)
```

Out[36]:
```
0.8795552810376775
```

In [37]:
```
y_pred_8 = xgbc_8.predict(rob_data_test)
recall_test_8 = recall_score(np.array(y_test_ohe), y_pred_8, average=None)
precision_test_8 = precision_score(np.array(y_test_ohe), y_pred_8, average=None)
f1_score_test_8 = f1_score(np.array(y_test_ohe), y_pred_8, average=None)
c8 = confusion_matrix(y_test_ohe,y_pred_8)
c8
```

Out[37]:
```
array([[1244,   95],
       [ 100,  180]], dtype=int64)
```

In [38]:
```
recall_test_8, precision_test_8, f1_score_test_8
```

Out[38]:
```
(array([0.92905153, 0.64285714]),
 array([0.92559524, 0.65454545]),
 array([0.92732016, 0.64864865]))
```

In [39]:
```
accuracy_8 = accuracy_score(y_test_ohe,y_pred_8)
accuracy_8
```

Out[39]:
```
0.8795552810376775
```

In [40]:
```
TP, FP, FN, TN = get_data(c8)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[40]:
```
(0.5759759316092378, 0.5760099774958793)
```

In [ ]:

Case 3：做完資料處理 + PCA (n_components = 'mle')

In [41]:
```
##使用case2中分割資料集後, 且標準化的x
pca = PCA(n_components='mle', random_state=1)
pca.fit(rob_data)
x_train_pca = pca.transform(rob_data)
x_teset_pca = pca.transform(rob_data_test)

x_train_pca = pd.DataFrame(x_train_pca)
x_teset_pca = pd.DataFrame(x_teset_pca)
```

In [42]:
```
x_teset_pca
```

Out[42]:

	0	1	2	3	4	5	6	7	8	9	...	56	57	
0	-79.781358	-31.957565	-5.532115	-0.839092	-1.093855	0.305354	-0.167379	0.076485	-0.338215	0.365858	...	0.000918	-0.000739	0.
1	-78.753232	-31.872057	-5.375580	1.583489	12.084591	7.338808	-1.116839	1.244810	0.798774	-0.830700	...	-0.006095	-0.002181	-0.
2	-78.102049	-31.952286	-5.530274	-0.801392	-0.581609	-0.707539	0.922996	0.519652	-0.071354	0.142429	...	-0.004146	-0.001041	0.
3	-79.755872	-31.932347	-5.489179	1.560374	-1.192879	0.558020	-0.296269	-0.153903	1.140025	-0.802757	...	0.001333	-0.002422	0.
4	-79.434765	-31.924273	-5.455139	1.632766	0.239405	-0.379710	-1.120206	0.333178	-0.599532	-0.317446	...	0.001049	0.001383	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1614	-78.527954	194.108615	0.086340	-1.125102	0.557553	-1.206749	0.142548	-1.297034	0.356839	-0.767332	...	-0.007041	0.002382	-0.
1615	-77.702755	-31.936531	-5.486375	0.847469	-0.676716	-0.268977	-0.573102	0.222138	-0.623429	-0.269605	...	0.001892	0.002144	0.
1616	-79.637406	-31.929777	-5.473718	1.316412	-0.130438	-0.016046	-0.755882	0.010477	1.204657	-0.632886	...	-0.003936	0.005923	0.
1617	-79.606558	-31.951523	-5.526613	-0.305370	-1.129678	0.469102	0.170312	-0.095955	1.539555	0.566777	...	-0.001810	0.004001	0.
1618	-79.351679	-31.948062	-5.514157	-0.135532	-0.831739	-0.214422	-0.122102	0.230673	-0.322462	0.042037	...	-0.001487	-0.007454	-0.

1619 rows × 66 columns

In [43]:
```
xgbc_3 =XGBClassifier()
xgbc_3.fit(x_train_pca , y_train_ohe)
xgbc_3.score(x_train_pca , y_train_ohe)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:46:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[43]:
```
1.0
```

In [44]:
```
y_pred_pca = xgbc_3.predict(x_teset_pca)
recall_test_pca = recall_score(np.array(y_test_ohe), y_pred_pca, average=None)
precision_test_pca = precision_score(np.array(y_test_ohe), y_pred_pca, average=None)
f1_score_test_pca = f1_score(np.array(y_test_ohe), y_pred_pca, average=None)
c3=confusion_matrix(y_test_ohe,y_pred_pca)
c3
```

Out[44]:
```
array([[1278,   61],
       [ 125,  155]], dtype=int64)
```

In [45]:
```
recall_test_pca, precision_test_pca,f1_score_test_pca
```

Out[45]:
```
(array([0.95444361, 0.55357143]),
 array([0.9109052 , 0.71759259]),
 array([0.9321663, 0.625    ]))
```

In [46]:
```
accuracy_3 = accuracy_score(y_test_ohe,y_pred_pca)
accuracy_3
```

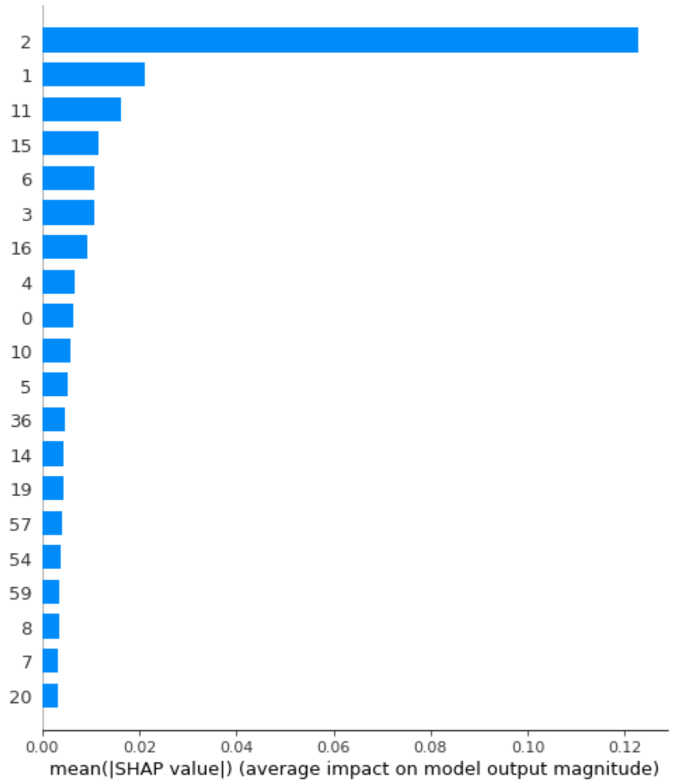
Out[46]:
```
0.8851142680667079
```

In [47]:
```
TP, FP, FN, TN = get_data(c3)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[47]:
```
(0.5584959769171224, 0.5650542760249043)
```

In [48]:
```
# 特徵重要程度
explainer_pca = shap.TreeExplainer(xgbc_3,x_train_pca, model_output='probability')
shap_values_pca = explainer_pca.shap_values(x_train_pca)
shap.summary_plot(shap_values_pca,x_train_pca,plot_type="bar")
```

```
97%|===================  | 6309/6476 [00:24<00:00]
```



In [49]:
```
shap_value_pca_1 = pd.DataFrame(shap_values_pca)
shap_value_1_0 = abs(shap_value_pca_1)
shap_value_col_1 = np.sum(shap_value_1_0 , axis = 0)/len(shap_values_pca)
shap_value_col_1 = shap_value_col_1.sort_values()
p = shap_value_col_1/sum(shap_value_col_1)*100
p.sort_values(ascending=False)
```

Out[49]:
```
2     34.341784
1      5.948335
11     4.509345
15     3.238523
6      3.009920
         ...
62     0.333105
45     0.315514
26     0.310213
32     0.306655
34     0.280365
Length: 66, dtype: float64
```

Case 4 : 做完資料處理 + PCA (n_components = 'mle') + Imbalanced data 處理

In [50]:
```
# summarize class distribution
counter = Counter(y_train_ohe)
print(counter)
# define pipeline
over = ADASYN(sampling_strategy=1.0, random_state=1)
#under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over)]
pipeline = Pipeline(steps=steps)
# transform the dataset
OU_X, OU_y = pipeline.fit_resample(x_train_pca, y_train_ohe)
# summarize the new class distribution
counter = Counter(OU_y)
print(counter)
```

```
Counter({0: 5479, 1: 997})
Counter({1: 5581, 0: 5479})
```

In [51]:
```
xgbc_4 =XGBClassifier()
xgbc_4.fit(OU_X , OU_y)
xgbc_4.score(OU_X , OU_y)
```

```
[05:46:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

Out[51]:
```
0.9999095840867993
```

In [52]:
```
xgbc_4.score(x_teset_pca , y_test_ohe)
```

Out[52]:
```
0.8733786287831995
```

In [53]:
```
y_pred_4 = xgbc_4.predict(x_teset_pca)
recall_test_4 = recall_score(np.array(y_test_ohe), y_pred_4, average=None)
precision_test_4 = precision_score(np.array(y_test_ohe), y_pred_4, average=None)
f1_score_test_4 = f1_score(np.array(y_test_ohe), y_pred_4, average=None)
c4=confusion_matrix(y_test_ohe,y_pred_4)
c4
```

Out[53]:
```
array([[1231,  108],
       [  97,  183]], dtype=int64)
```

In [54]:
```
recall_test_4, precision_test_4, f1_score_test_4
```

Out[54]:
```
(array([0.91934279, 0.65357143]),
 array([0.92695783, 0.62886598]),
 array([0.92313461, 0.64098074]))
```

In [55]:
```
accuracy_4 = accuracy_score(y_test_ohe,y_pred_4)
accuracy_4
```

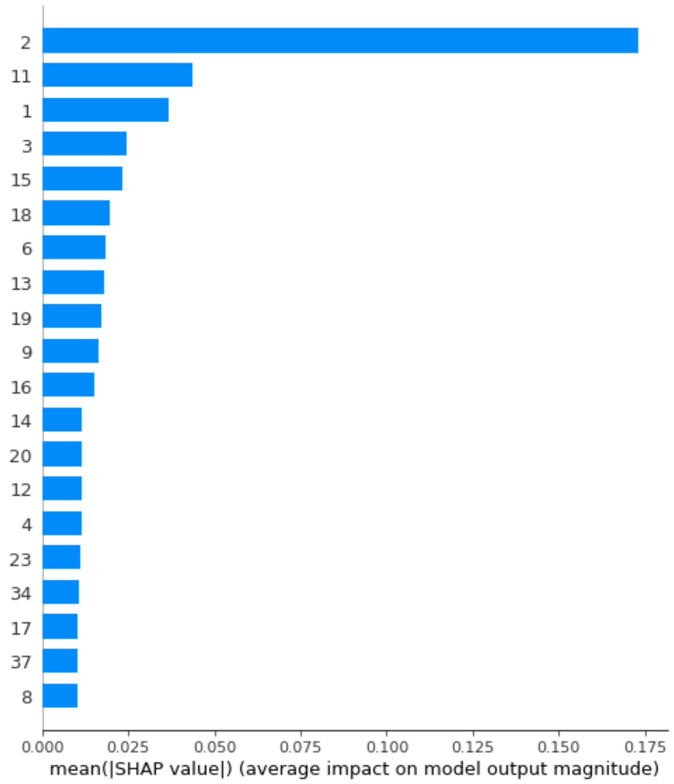
Out[55]:
```
0.8733786287831995
```

In [56]:
```
TP, FP, FN, TN = get_data(c4)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[56]:
```
(0.5641499745892586, 0.5643043203039761)
```

In [57]:
```
explainer_ada = shap.TreeExplainer(xgbc_4,OU_X, model_output='probability')
shap_values_ada = explainer_ada.shap_values(OU_X)
shap.summary_plot(shap_values_ada,OU_X,plot_type="bar")
```

```
99%|===================| 10987/11060 [00:36<00:00]
```



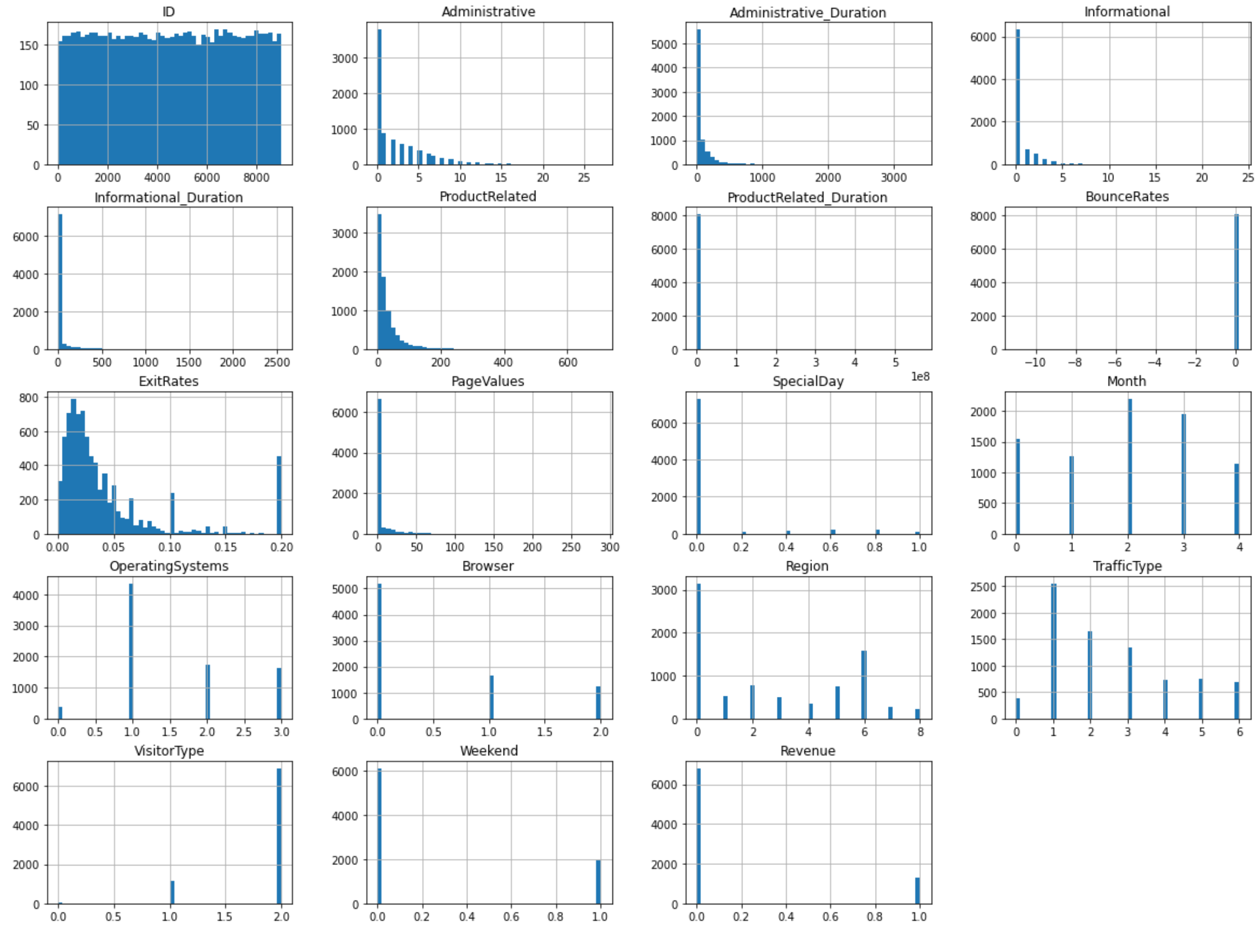
In [58]:
```
shap_value_1_ada = pd.DataFrame(shap_values_ada)
shap_value_1_0_ada = abs(shap_value_1_ada)
shap_value_col_1_ada = np.sum(shap_value_1_0_ada , axis = 0)/len(shap_values_ada)
shap_value_col_1_ada = shap_value_col_1_ada.sort_values()
p = shap_value_col_1_ada/sum(shap_value_col_1_ada)*100
p.sort_values(ascending=False)
```

Out[58]:
```
2     23.448936
11     5.942229
1      4.962965
3      3.307677
15     3.137786
        ...
38     0.373129
52     0.371985
65     0.370586
43     0.356740
46     0.269031
Length: 66, dtype: float64
```

將類別型特徵的項目做整合

In [59]:
```
data4 = data3.copy()
data4['Month'] = data3['Month'].apply(new_month)
data4['OperatingSystems'] = data3['OperatingSystems'].apply(new_OperatingSystems)
data4['Browser'] = data3['Browser'].apply(new_browser)
data4['TrafficType'] = data3['TrafficType'].apply(new_TrafficType)

#將每一數值屬性畫出直方圖
%matplotlib inline
data4.hist(bins=50, figsize=(20,15))
plt.show()
```



In [60]:
```
#尋找相關性
#使用corr()計算每一對屬性之間的標準相關性係數
corr_matrix = data4.corr()

#查看每一個屬性與是否訂房之間的相關性有多大
corr_matrix['Revenue'].sort_values(ascending=False)
```

Out[60]:
```
Revenue                    1.000000
PageValues                 0.499459
ProductRelated             0.149078
Administrative             0.142878
Administrative_Duration    0.096156
Informational              0.091202
Informational_Duration     0.068601
Month                      0.051810
Weekend                    0.023159
Browser                    0.017122
ID                        -0.001806
ProductRelated_Duration   -0.005533
Region                    -0.010480
BounceRates               -0.050035
TrafficType               -0.052108
OperatingSystems          -0.073666
SpecialDay                -0.076383
VisitorType               -0.111432
ExitRates                 -0.207819
Name: Revenue, dtype: float64
```

In [61]:
```
# 對類別型特徵做One-hot-encoding
df_str_2 = data4.astype({'Month':'category','OperatingSystems':'category','Browser':'category','Region':'category','Tr
df_str_2
df_dum_2 = pd.get_dummies(df_str_2[['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','Weekenc

df_str_2.drop(['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','Weekend'], axis=1, inplace=T

df_new_2 = pd.concat([df_dum_2,df_str_2],axis=1)
df_new_2
```

Out[61]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSystems_3	Bro
0	0	0	1	0	0	0	0	1	0	
1	0	0	0	0	1	0	1	0	0	
2	0	0	0	1	0	0	1	0	0	
3	0	0	1	0	0	0	0	0	1	
4	0	0	1	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	
8095	0	0	1	0	0	0	1	0	0	
8096	1	0	0	0	0	0	0	1	0	
8097	1	0	0	0	0	0	1	0	0	
8098	0	0	0	1	0	0	0	0	1	
8099	0	0	0	1	0	0	0	0	1	

8095 rows × 45 columns

In [62]:
```
#尋找相關性
#使用corr()計算每一對屬性之間的標準相關性係數
corr_matrix = df_new_2.corr()

#查看每一個屬性與是否訂房之間的相關性有多大
corr_matrix['Revenue'].sort_values(ascending=False)
```

Out[62]:
```
Revenue                      1.000000
PageValues                   0.499459
Month_3                      0.158018
ProductRelated               0.149078
Administrative               0.142878
TrafficType_1                0.118563
VisitorType_1.0              0.116709
Administrative_Duration      0.096156
Informational                0.091202
Informational_Duration       0.068601
TrafficType_6                0.066664
OperatingSystems_1           0.052443
TrafficType_0                0.035160
OperatingSystems_0           0.028453
Weekend_1                    0.023159
Browser_2                    0.019684
Region_0.0                   0.019631
Region_4.0                   0.016641
Region_5.0                   0.011108
VisitorType_0.0              0.010331
Region_3.0                   0.004525
Region_8.0                   0.001859
Month_0                     -0.000537
ID                          -0.001806
Browser_1                   -0.003577
ProductRelated_Duration     -0.005533
TrafficType_4               -0.006006
Region_6.0                  -0.007108
Browser_0                   -0.011869
OperatingSystems_2          -0.016142
Region_1.0                  -0.016833
Region_2.0                  -0.021571
Region_7.0                  -0.022825
Weekend_0                   -0.023159
Month_4                     -0.031019
BounceRates                 -0.050035
Month_1                     -0.059319
OperatingSystems_3          -0.063234
TrafficType_5               -0.064800
TrafficType_2               -0.067781
SpecialDay                  -0.076383
Month_2                     -0.078724
TrafficType_3               -0.089715
VisitorType_2.0             -0.116836
ExitRates                   -0.207819
Name: Revenue, dtype: float64
```

Case 9: 合併特徵類別 no PCA

In [63]:
```
y_ohe_9 = df_new_2['Revenue'].values
X_ohe_9 = df_new_2.drop(['ID','Revenue'],axis=1)
```

In [64]:
```
x_train_ohe_9, x_test_ohe_9, y_train_ohe_9, y_test_ohe_9 = train_test_split(X_ohe_9, y_ohe_9, test_size=0.2, random_st
```

In [65]:
```
#特徵標準化
rob_scaler_9 = RobustScaler()
rob_data_9 = x_train_ohe_9.copy()


rob_data_9['rob_scaled_Administrative'] = rob_scaler_9.fit_transform(rob_data_9['Administrative'].values.reshape(-1,1)
rob_data_9['rob_scaled_Administrative_Duration'] = rob_scaler_9.fit_transform(rob_data_9['Administrative_Duration'].va
rob_data_9['rob_scaled_Informational'] = rob_scaler_9.fit_transform(rob_data_9['Informational'].values.reshape(-1,1))
rob_data_9['rob_scaled_Informational_Duration'] = rob_scaler_9.fit_transform(rob_data_9['Informational_Duration'].valu
rob_data_9['rob_scaled_ProductRelated'] = rob_scaler_9.fit_transform(rob_data_9['ProductRelated'].values.reshape(-1,1)
rob_data_9['rob_scaled_ProductRelated_Duration'] = rob_scaler_9.fit_transform(rob_data_9['ProductRelated_Duration'].va
rob_data_9['rob_scaled_BounceRates'] = rob_scaler_9.fit_transform(rob_data_9['BounceRates'].values.reshape(-1,1))
rob_data_9['rob_scaled_ExitRates'] = rob_scaler_9.fit_transform(rob_data_9['ExitRates'].values.reshape(-1,1))
rob_data_9['rob_scaled_PageValues'] = rob_scaler_9.fit_transform(rob_data_9['PageValues'].values.reshape(-1,1))
rob_data_9['rob_scaled_SpecialDay'] = rob_scaler_9.fit_transform(rob_data_9['SpecialDay'].values.reshape(-1,1))




rob_data_9.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelated',

rob_data_9.describe()
```

Out[65]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	Operati
count	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	
mean	0.186998	0.157505	0.274707	0.237492	0.143298	0.046016	0.536288	0.214330	
std	0.389940	0.364304	0.446401	0.425579	0.350404	0.209536	0.498720	0.410388	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
75%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 43 columns

In [66]:
```
rob_scaler_9 = RobustScaler()
rob_data_test_9 = x_test_ohe_9.copy()


rob_data_test_9['rob_scaled_Administrative'] = rob_scaler_9.fit_transform(rob_data_test_9['Administrative'].values.res
rob_data_test_9['rob_scaled_Administrative_Duration'] = rob_scaler_9.fit_transform(rob_data_test_9['Administrative_Dur
rob_data_test_9['rob_scaled_Informational'] = rob_scaler_9.fit_transform(rob_data_test_9['Informational'].values.resha
rob_data_test_9['rob_scaled_Informational_Duration'] = rob_scaler_9.fit_transform(rob_data_test_9['Informational_Durat
rob_data_test_9['rob_scaled_ProductRelated'] = rob_scaler_9.fit_transform(rob_data_test_9['ProductRelated'].values.res
rob_data_test_9['rob_scaled_ProductRelated_Duration'] = rob_scaler_9.fit_transform(rob_data_test_9['ProductRelated_Dur
rob_data_test_9['rob_scaled_BounceRates'] = rob_scaler_9.fit_transform(rob_data_test_9['BounceRates'].values.reshape(-
rob_data_test_9['rob_scaled_ExitRates'] = rob_scaler_9.fit_transform(rob_data_test_9['ExitRates'].values.reshape(-1,1)
rob_data_test_9['rob_scaled_PageValues'] = rob_scaler_9.fit_transform(rob_data_test_9['PageValues'].values.reshape(-1,
rob_data_test_9['rob_scaled_SpecialDay'] = rob_scaler_9.fit_transform(rob_data_test_9['SpecialDay'].values.reshape(-1,




rob_data_test_9.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRela

rob_data_test_9.describe()
```

Out[66]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	Operati
count	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	
mean	0.201977	0.151328	0.261272	0.254478	0.130945	0.041384	0.544163	0.213712	
std	0.401599	0.358479	0.439464	0.435702	0.337444	0.199237	0.498200	0.410053	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
75%	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 43 columns

In [67]:
```
xgbc_9 =XGBClassifier()
xgbc_9.fit(rob_data_9 , y_train_ohe_9)
xgbc_9.score(rob_data_9 , y_train_ohe_9)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:47:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[67]:
```
0.9941321803582458
```

In [68]:
```
xgbc_9.score(rob_data_test_9 , y_test_ohe_9)
```

Out[68]:
```
0.897467572575664
```

In [69]:
```
y_pred_9 = xgbc_9.predict(rob_data_test_9)
recall_test_9 = recall_score(np.array(y_test_ohe_9), y_pred_9, average=None)
precision_test_9 = precision_score(np.array(y_test_ohe_9), y_pred_9, average=None)
f1_score_test_9 = f1_score(np.array(y_test_ohe_9), y_pred_9, average=None)
c9 = confusion_matrix(y_test_ohe_9,y_pred_9)
c9
```

Out[69]:
```
array([[1280,   59],
       [ 107,  173]], dtype=int64)
```

In [70]:
```
recall_test_9, precision_test_9, f1_score_test_9
```

Out[70]:
```
(array([0.95593727, 0.61785714]),
 array([0.92285508, 0.74568966]),
 array([0.93910492, 0.67578125]))
```

In [71]:
```
accuracy_9 = accuracy_score(y_test_ohe_9,y_pred_9)
accuracy_9
```

Out[71]:
```
0.897467572575664
```

In [72]:
```
TP, FP, FN, TN = get_data(c9)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[72]:
```
(0.6155208524079839, 0.6193603419630186)
```

Case 11: 合併特徵類別 no PCA + Imbalance data處理

In [126]:
```
# summarize class distribution
counter = Counter(y_train_ohe_9)
print(counter)
# define pipeline
over = ADASYN(sampling_strategy=1,random_state=1)
#under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over)]
pipeline = Pipeline(steps=steps)
# transform the dataset
OU_X_11, OU_y_11 = pipeline.fit_resample(rob_data_9, y_train_ohe_9)
# summarize the new class distribution
counter = Counter(OU_y_11)
print(counter)
```

```
Counter({0: 5479, 1: 997})
Counter({0: 5479, 1: 5430})
```

In [127]:
```
xgbc_11 =XGBClassifier()
xgbc_11.fit(OU_X_11 , OU_y_11)
xgbc_11.score(OU_X_11 , OU_y_11)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:54:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[127]:
```
0.9976166468053901
```

In [128]:
```
xgbc_11.score(rob_data_test_9 , y_test_ohe_9)
```
Out[128]:
```
0.8820259419394688
```
In [129]:
```
y_pred_11 = xgbc_11.predict(rob_data_test_9)
recall_test_11 = recall_score(np.array(y_test_ohe_9), y_pred_11, average=None)
precision_test_11 = precision_score(np.array(y_test_ohe_9), y_pred_11, average=None)
f1_score_test_11 = f1_score(np.array(y_test_ohe_9), y_pred_11, average=None)
c11=confusion_matrix(y_test_ohe_9,y_pred_11)
c11
```
Out[129]:
```
array([[1252,   87],
       [ 104,  176]], dtype=int64)
```
In [130]:
```
recall_test_11, precision_test_11, f1_score_test_11
```
Out[130]:
```
(array([0.93502614, 0.62857143]),
 array([0.92330383, 0.66920152]),
 array([0.92912801, 0.64825046]))
```
In [131]:
```
accuracy_11 = accuracy_score(y_test_ohe_9,y_pred_11)
accuracy_11
```
Out[131]:
```
0.8820259419394688
```
In [132]:
```
TP, FP, FN, TN = get_data(c11)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```
Out[132]:
```
(0.5774619211655054, 0.5778707270694273)
```

Case 5 : 合併特徵類別 + PCA (n_components = 'mle') + Imbalanced data 處理

Split x and y

In [73]:
```
y_ohe_2 = df_new_2['Revenue'].values
X_ohe_2 = df_new_2.drop(['ID','Revenue'],axis=1)
```

分割訓練集 / 測試集 80/20

In [74]:
```
x_train_ohe_2, x_test_ohe_2, y_train_ohe_2, y_test_ohe_2 = train_test_split(X_ohe_2, y_ohe_2, test_size=0.2, random_st
```

In [75]:
```
#特徵標準化
rob_scaler_2 = RobustScaler()
rob_data_2 = x_train_ohe_2.copy()


rob_data_2['rob_scaled_Administrative'] = rob_scaler_2.fit_transform(rob_data_2['Administrative'].values.reshape(-1,1)
rob_data_2['rob_scaled_Administrative_Duration'] = rob_scaler_2.fit_transform(rob_data_2['Administrative_Duration'].va
rob_data_2['rob_scaled_Informational'] = rob_scaler_2.fit_transform(rob_data_2['Informational'].values.reshape(-1,1))
rob_data_2['rob_scaled_Informational_Duration'] = rob_scaler_2.fit_transform(rob_data_2['Informational_Duration'].valu
rob_data_2['rob_scaled_ProductRelated'] = rob_scaler_2.fit_transform(rob_data_2['ProductRelated'].values.reshape(-1,1)
rob_data_2['rob_scaled_ProductRelated_Duration'] = rob_scaler_2.fit_transform(rob_data_2['ProductRelated_Duration'].va
rob_data_2['rob_scaled_BounceRates'] = rob_scaler_2.fit_transform(rob_data_2['BounceRates'].values.reshape(-1,1))
rob_data_2['rob_scaled_ExitRates'] = rob_scaler_2.fit_transform(rob_data_2['ExitRates'].values.reshape(-1,1))
rob_data_2['rob_scaled_PageValues'] = rob_scaler_2.fit_transform(rob_data_2['PageValues'].values.reshape(-1,1))
rob_data_2['rob_scaled_SpecialDay'] = rob_scaler_2.fit_transform(rob_data_2['SpecialDay'].values.reshape(-1,1))




rob_data_2.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelated',

rob_data_2.describe()
```

Out[75]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	Operati
count	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	
mean	0.186998	0.157505	0.274707	0.237492	0.143298	0.046016	0.536288	0.214330	
std	0.389940	0.364304	0.446401	0.425579	0.350404	0.209536	0.498720	0.410388	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
75%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 43 columns

In [76]:
```
rob_scaler_2 = RobustScaler()
rob_data_test_2 = x_test_ohe_2.copy()


rob_data_test_2['rob_scaled_Administrative'] = rob_scaler_2.fit_transform(rob_data_test_2['Administrative'].values.res
rob_data_test_2['rob_scaled_Administrative_Duration'] = rob_scaler_2.fit_transform(rob_data_test_2['Administrative_Dur
rob_data_test_2['rob_scaled_Informational'] = rob_scaler_2.fit_transform(rob_data_test_2['Informational'].values.resha
rob_data_test_2['rob_scaled_Informational_Duration'] = rob_scaler_2.fit_transform(rob_data_test_2['Informational_Durat
rob_data_test_2['rob_scaled_ProductRelated'] = rob_scaler_2.fit_transform(rob_data_test_2['ProductRelated'].values.res
rob_data_test_2['rob_scaled_ProductRelated_Duration'] = rob_scaler_2.fit_transform(rob_data_test_2['ProductRelated_Dur
rob_data_test_2['rob_scaled_BounceRates'] = rob_scaler_2.fit_transform(rob_data_test_2['BounceRates'].values.reshape(-
rob_data_test_2['rob_scaled_ExitRates'] = rob_scaler_2.fit_transform(rob_data_test_2['ExitRates'].values.reshape(-1,1)
rob_data_test_2['rob_scaled_PageValues'] = rob_scaler_2.fit_transform(rob_data_test_2['PageValues'].values.reshape(-1,
rob_data_test_2['rob_scaled_SpecialDay'] = rob_scaler_2.fit_transform(rob_data_test_2['SpecialDay'].values.reshape(-1,




rob_data_test_2.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRela

rob_data_test_2.describe()
```

Out[76]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	Operati
count	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	
mean	0.201977	0.151328	0.261272	0.254478	0.130945	0.041384	0.544163	0.213712	
std	0.401599	0.358479	0.439464	0.435702	0.337444	0.199237	0.498200	0.410053	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
75%	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 43 columns

In [77]:
```python
##特徵pca
pca_2 = PCA(n_components='mle',random_state=1)
pca_2.fit(rob_data_2)
x_train_pca_2 = pca_2.transform(rob_data_2)
x_teset_pca_2 = pca_2.transform(rob_data_test_2)

x_train_pca_2 = pd.DataFrame(x_train_pca_2)
x_teset_pca_2 = pd.DataFrame(x_teset_pca_2)
```

In [78]:
```python
x_teset_pca_2
```

Out[78]:

	0	1	2	3	4	5	6	7	8	9	...	26	27	
0	-79.781358	-31.957548	-5.532330	-0.839105	-1.095003	0.305034	-0.164587	0.073071	-0.374804	0.476172	...	-0.138461	-0.000955	0.
1	-78.753232	-31.872071	-5.375588	1.583949	12.085089	7.336106	-1.129985	1.232592	0.801437	-0.771698	...	0.728285	0.339338	0.
2	-78.102049	-31.952290	-5.530359	-0.801232	-0.573387	-0.704938	0.914887	0.520060	-0.063402	0.194679	...	0.377574	-0.312611	-0.
3	-79.755872	-31.932332	-5.489216	1.560404	-1.193652	0.556476	-0.295231	-0.156683	1.131469	-0.742277	...	0.411228	-0.301033	-0.
4	-79.434766	-31.924295	-5.455691	1.632434	0.240527	-0.378483	-1.112454	0.332263	-0.596306	-0.305902	...	0.132023	0.104600	-0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1614	-78.527954	194.108631	0.086310	-1.125068	0.556882	-1.208400	0.142966	-1.298104	0.362206	-0.759106	...	-0.108671	-0.004628	-0.
1615	-77.702754	-31.936515	-5.486412	0.847499	-0.677524	-0.270589	-0.571202	0.218649	-0.625891	-0.232929	...	-0.052826	0.053987	-0.
1616	-79.637406	-31.929781	-5.473803	1.316572	-0.122234	-0.013447	-0.765282	0.012664	1.216048	-0.619902	...	-0.420889	-0.238956	0.
1617	-79.606558	-31.951546	-5.527165	-0.305701	-1.128451	0.470482	0.176313	-0.097691	1.538483	0.519729	...	-0.467091	-0.257644	0.
1618	-79.351679	-31.948047	-5.514194	-0.135503	-0.832532	-0.216024	-0.119860	0.226409	-0.336125	0.139583	...	0.211830	0.033569	0.

1619 rows × 36 columns

In [79]:
```python
# summarize class distribution
counter = Counter(y_train_ohe_2)
print(counter)
# define pipeline
over = ADASYN(sampling_strategy=1,random_state=1)
#under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over)]
pipeline = Pipeline(steps=steps)
# transform the dataset
OU_X_2, OU_y_2 = pipeline.fit_resample(x_train_pca_2, y_train_ohe_2)
# summarize the new class distribution
counter = Counter(OU_y_2)
print(counter)
```

```
Counter({0: 5479, 1: 997})
Counter({0: 5479, 1: 5430})
```

In [80]:
```python
xgbc_5 =XGBClassifier()
xgbc_5.fit(OU_X_2 , OU_y_2)
xgbc_5.score(OU_X_2 , OU_y_2)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:47:24] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[80]:
```
0.9995416628471904
```

In [81]:
```python
xgbc_5.score(x_teset_pca_2 , y_test_ohe_2)
```

Out[81]:
```
0.871525633106856
```

In [82]:
```python
y_pred_5 = xgbc_5.predict(x_teset_pca_2)
recall_test_5 = recall_score(np.array(y_test_ohe_2), y_pred_5, average=None)
precision_test_5 = precision_score(np.array(y_test_ohe_2), y_pred_5, average=None)
f1_score_test_5 = f1_score(np.array(y_test_ohe_2), y_pred_5, average=None)
c5=confusion_matrix(y_test_ohe_2,y_pred_5)
c5
```

Out[82]:
```
array([[1227,  112],
       [  96,  184]], dtype=int64)
```

In [83]:
```
recall_test_5, precision_test_5, f1_score_test_5
```

Out[83]:
```
(array([0.91635549, 0.65714286]),
 array([0.92743764, 0.62162162]),
 array([0.92186326, 0.63888889]))
```

In [84]:
```
accuracy_5 = accuracy_score(y_test_ohe_2,y_pred_5)
accuracy_5
```

Out[84]:
```
0.871525633106856
```

In [85]:
```
TP, FP, FN, TN = get_data(c5)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[85]:
```
(0.5608254736666386, 0.5611457738918311)
```

Case10: Drop掉與預測目標不太相關的特徵 no PCA

abs(corr)<=0.01

In [86]:
```
y_ohe_10=df_new_2['Revenue'].values
X_ohe_10=df_new_2.drop(['ID','Revenue','Month_0','Region_8.0','Region_3.0','Browser_1','ProductRelated_Duration','Tra
```

In [87]:
```
x_train_ohe_10, x_test_ohe_10, y_train_ohe_10, y_test_ohe_10 = train_test_split(X_ohe_10, y_ohe_10, test_size=0.2, rar
```

In [88]:
```
#特徵標準化
rob_scaler_10 = RobustScaler()
rob_data_10 = x_train_ohe_10.copy()


rob_data_10['rob_scaled_Administrative'] = rob_scaler_10.fit_transform(rob_data_10['Administrative'].values.reshape(-1
rob_data_10['rob_scaled_Administrative_Duration'] = rob_scaler_10.fit_transform(rob_data_10['Administrative_Duration']
rob_data_10['rob_scaled_Informational'] = rob_scaler_10.fit_transform(rob_data_10['Informational'].values.reshape(-1,1
rob_data_10['rob_scaled_Informational_Duration'] = rob_scaler_10.fit_transform(rob_data_10['Informational_Duration'].v
rob_data_10['rob_scaled_ProductRelated'] = rob_scaler_10.fit_transform(rob_data_10['ProductRelated'].values.reshape(-1

rob_data_10['rob_scaled_BounceRates'] = rob_scaler_10.fit_transform(rob_data_10['BounceRates'].values.reshape(-1,1))
rob_data_10['rob_scaled_ExitRates'] = rob_scaler_10.fit_transform(rob_data_10['ExitRates'].values.reshape(-1,1))
rob_data_10['rob_scaled_PageValues'] = rob_scaler_10.fit_transform(rob_data_10['PageValues'].values.reshape(-1,1))
rob_data_10['rob_scaled_SpecialDay'] = rob_scaler_10.fit_transform(rob_data_10['SpecialDay'].values.reshape(-1,1))




rob_data_10.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelated'

rob_data_10.describe()
```

Out[88]:

	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSystems_3
count	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000
mean	0.157505	0.274707	0.237492	0.143298	0.046016	0.536288	0.214330	0.203366
std	0.364304	0.446401	0.425579	0.350404	0.209536	0.498720	0.410388	0.402534
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 36 columns

In [89]:
```
rob_scaler_10 = RobustScaler()
rob_data_test_10 = x_test_ohe_10.copy()


rob_data_test_10['rob_scaled_Administrative'] = rob_scaler_10.fit_transform(rob_data_test_10['Administrative'].values.
rob_data_test_10['rob_scaled_Administrative_Duration'] = rob_scaler_10.fit_transform(rob_data_test_10['Administrative_
rob_data_test_10['rob_scaled_Informational'] = rob_scaler_10.fit_transform(rob_data_test_10['Informational'].values.re
rob_data_test_10['rob_scaled_Informational_Duration'] = rob_scaler_10.fit_transform(rob_data_test_10['Informational_Du
rob_data_test_10['rob_scaled_ProductRelated'] = rob_scaler_10.fit_transform(rob_data_test_10['ProductRelated'].values.

rob_data_test_10['rob_scaled_BounceRates'] = rob_scaler_10.fit_transform(rob_data_test_10['BounceRates'].values.reshap
rob_data_test_10['rob_scaled_ExitRates'] = rob_scaler_10.fit_transform(rob_data_test_10['ExitRates'].values.reshape(-1
rob_data_test_10['rob_scaled_PageValues'] = rob_scaler_10.fit_transform(rob_data_test_10['PageValues'].values.reshape(
rob_data_test_10['rob_scaled_SpecialDay'] = rob_scaler_10.fit_transform(rob_data_test_10['SpecialDay'].values.reshape(




rob_data_test_10.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRel

rob_data_test_10.describe()
```

Out[89]:

	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSystems_3
count	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000
mean	0.151328	0.261272	0.254478	0.130945	0.041384	0.544163	0.213712	0.200741
std	0.358479	0.439464	0.435702	0.337444	0.199237	0.498200	0.410053	0.400679
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
75%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 36 columns

In [90]:
```
xgbc_10 =XGBClassifier()
xgbc_10.fit(rob_data_10 , y_train_ohe_10)
xgbc_10.score(rob_data_10 , y_train_ohe_10)
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[05:47:25] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[90]: 0.9918159357628166

In [91]:
```
xgbc_10.score(rob_data_test_10 , y_test_ohe_10)
```

Out[91]: 0.887584928968499

In [92]:
```
y_pred_10 = xgbc_10.predict(rob_data_test_10)
recall_test_10 = recall_score(np.array(y_test_ohe_10), y_pred_10, average=None)
precision_test_10 = precision_score(np.array(y_test_ohe_10), y_pred_10, average=None)
f1_score_test_10 = f1_score(np.array(y_test_ohe_10), y_pred_10, average=None)
c10 = confusion_matrix(y_test_ohe_10,y_pred_10)
c10
```

Out[92]:
```
array([[1278,   61],
       [ 121,  159]], dtype=int64)
```

In [93]:
```
recall_test_10, precision_test_10, f1_score_test_10
```

Out[93]:
```
(array([0.95444361, 0.56785714]),
 array([0.91350965, 0.72272727]),
 array([0.93352812, 0.636     ]))
```

In [94]:
```
accuracy_10 = accuracy_score(y_test_ohe_10,y_pred_10)
accuracy_10
```

Out[94]: 0.887584928968499

In [95]:
```
TP, FP, FN, TN = get_data(c10)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```
Out[95]: (0.5706571470202534, 0.5764607762511087)

In [ ]:

Case6: Drop掉與預測目標不太相關的特徵

abs(corr)<=0.01

Split x and y

In [96]:
```
y_ohe_drop=df_new_2['Revenue'].values
X_ohe_drop=df_new_2.drop(['ID','Revenue','Month_0','Region_8.0','Region_3.0','Browser_1','ProductRelated_Duration','Tr
```

分割訓練集 / 測試集 80/20

In [97]:
```
x_train_ohe_drop, x_test_ohe_drop, y_train_ohe_drop, y_test_ohe_drop = train_test_split(X_ohe_drop, y_ohe_drop, test_s
```

In [98]:
```
#特徵標準化
rob_scaler_3 = RobustScaler()
rob_data_3 = x_train_ohe_drop.copy()


rob_data_3['rob_scaled_Administrative'] = rob_scaler_3.fit_transform(rob_data_3['Administrative'].values.reshape(-1,1)
rob_data_3['rob_scaled_Administrative_Duration'] = rob_scaler_3.fit_transform(rob_data_3['Administrative_Duration'].va
rob_data_3['rob_scaled_Informational'] = rob_scaler_3.fit_transform(rob_data_3['Informational'].values.reshape(-1,1))
rob_data_3['rob_scaled_Informational_Duration'] = rob_scaler_3.fit_transform(rob_data_3['Informational_Duration'].valu
rob_data_3['rob_scaled_ProductRelated'] = rob_scaler_3.fit_transform(rob_data_3['ProductRelated'].values.reshape(-1,1)

rob_data_3['rob_scaled_BounceRates'] = rob_scaler_3.fit_transform(rob_data_3['BounceRates'].values.reshape(-1,1))
rob_data_3['rob_scaled_ExitRates'] = rob_scaler_3.fit_transform(rob_data_3['ExitRates'].values.reshape(-1,1))
rob_data_3['rob_scaled_PageValues'] = rob_scaler_3.fit_transform(rob_data_3['PageValues'].values.reshape(-1,1))
rob_data_3['rob_scaled_SpecialDay'] = rob_scaler_3.fit_transform(rob_data_3['SpecialDay'].values.reshape(-1,1))




rob_data_3.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelated',

rob_data_3.describe()
```

Out[98]:

	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSystems_3
count	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000
mean	0.157505	0.274707	0.237492	0.143298	0.046016	0.536288	0.214330	0.203366
std	0.364304	0.446401	0.425579	0.350404	0.209536	0.498720	0.410388	0.402534
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 36 columns

In [99]:
```python
rob_scaler_3 = RobustScaler()
rob_data_test_3 = x_test_ohe_drop.copy()


rob_data_test_3['rob_scaled_Administrative'] = rob_scaler_3.fit_transform(rob_data_test_3['Administrative'].values.res
rob_data_test_3['rob_scaled_Administrative_Duration'] = rob_scaler_3.fit_transform(rob_data_test_3['Administrative_Dur
rob_data_test_3['rob_scaled_Informational'] = rob_scaler_3.fit_transform(rob_data_test_3['Informational'].values.resha
rob_data_test_3['rob_scaled_Informational_Duration'] = rob_scaler_3.fit_transform(rob_data_test_3['Informational_Durat
rob_data_test_3['rob_scaled_ProductRelated'] = rob_scaler_3.fit_transform(rob_data_test_3['ProductRelated'].values.res

rob_data_test_3['rob_scaled_BounceRates'] = rob_scaler_3.fit_transform(rob_data_test_3['BounceRates'].values.reshape(-
rob_data_test_3['rob_scaled_ExitRates'] = rob_scaler_3.fit_transform(rob_data_test_3['ExitRates'].values.reshape(-1,1)
rob_data_test_3['rob_scaled_PageValues'] = rob_scaler_3.fit_transform(rob_data_test_3['PageValues'].values.reshape(-1,
rob_data_test_3['rob_scaled_SpecialDay'] = rob_scaler_3.fit_transform(rob_data_test_3['SpecialDay'].values.reshape(-1,



rob_data_test_3.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRela

rob_data_test_3.describe()
```

Out[99]:

	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSystems_3
count	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000
mean	0.151328	0.261272	0.254478	0.130945	0.041384	0.544163	0.213712	0.200741
std	0.358479	0.439464	0.435702	0.337444	0.199237	0.498200	0.410053	0.400679
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
75%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 36 columns

In [100]:
```python
##特徵pca
pca_3 = PCA(n_components='mle',random_state=1)
pca_3.fit(rob_data_3)
x_train_pca_3 = pca_3.transform(rob_data_3)
x_teset_pca_3 = pca_3.transform(rob_data_test_3)

x_train_pca_3 = pd.DataFrame(x_train_pca_3)
x_teset_pca_3 = pd.DataFrame(x_teset_pca_3)
```

In [101]:
```python
x_teset_pca_3
```

Out[101]:

	0	1	2	3	4	5	6	7	8	9	...	23	24	
0	-31.952648	-5.531419	-0.839619	-1.093674	0.315992	-0.169627	0.045409	-0.122938	0.552141	0.100332	...	-0.116804	-0.087052	-0.1
1	-31.867125	-5.374381	1.584225	12.092622	7.312629	-1.124364	1.282231	0.266913	-0.903372	1.959901	...	-0.035773	-0.067130	-0.2
2	-31.947477	-5.529077	-0.801711	-0.581092	-0.699905	0.913964	0.502425	0.028045	0.241316	0.230615	...	-0.421281	0.432088	0.2
3	-31.927340	-5.487923	1.560696	-1.187603	0.528167	-0.279600	-0.080566	0.564163	-0.903472	0.083312	...	-0.365763	0.402941	0.2
4	-31.919402	-5.454414	1.631957	0.232855	-0.373589	-1.116208	0.293456	-0.654457	-0.049211	0.017021	...	-0.003481	0.036184	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1614	194.113557	0.087283	-1.124627	0.562648	-1.235590	0.158810	-1.244798	-0.172846	-0.642472	-0.852997	...	-0.067052	0.098971	-0.0
1615	-31.931759	-5.485449	0.846985	-0.676558	-0.261830	-0.580145	0.188279	-0.650851	-0.018398	0.058799	...	0.094988	-0.008415	0.0
1616	-31.924751	-5.471818	1.316548	-0.125965	-0.045105	-0.729035	0.095352	0.666373	-0.765775	-0.376640	...	-0.010066	-0.163699	0.0
1617	-31.946527	-5.525718	-0.305700	-1.129842	0.439743	0.210760	-0.015422	1.413695	0.140520	0.325985	...	-0.050821	0.072521	0.1
1618	-31.943189	-5.513202	-0.136020	-0.831513	-0.207703	-0.127217	0.205560	-0.235510	0.255917	0.227280	...	-0.064516	-0.047096	-0.0

1619 rows × 33 columns

In [102]:
```
# summarize class distribution
counter = Counter(y_train_ohe_2)
print(counter)
# define pipeline
over = ADASYN(sampling_strategy=1,random_state=1)
#under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over)]
pipeline = Pipeline(steps=steps)
# transform the dataset
OU_X_3, OU_y_3 = pipeline.fit_resample(x_train_pca_3, y_train_ohe_drop)
# summarize the new class distribution
counter = Counter(OU_y_3)
print(counter)
```

```
Counter({0: 5479, 1: 997})
Counter({0: 5479, 1: 5412})
```

In [103]:
```
xgbc_6 =XGBClassifier()
xgbc_6.fit(OU_X_3 , OU_y_3)
xgbc_6.score(OU_X_3 , OU_y_3)
```

```
[05:47:26] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

Out[103]: 1.0

In [104]:
```
xgbc_6.score(x_teset_pca_3 , y_test_ohe_drop)
```

Out[104]: 0.8746139592340951

In [105]:
```
y_pred_6 = xgbc_6.predict(x_teset_pca_3)
recall_test_6 = recall_score(np.array(y_test_ohe_drop), y_pred_6, average=None)
precision_test_6 = precision_score(np.array(y_test_ohe_drop), y_pred_6, average=None)
f1_score_test_6 = f1_score(np.array(y_test_ohe_drop), y_pred_6, average=None)
c6 = confusion_matrix(y_test_ohe_drop,y_pred_6)
c6
```

Out[105]:
```
array([[1230,  109],
       [  94,  186]], dtype=int64)
```

In [106]:
```
recall_test_6, precision_test_6, f1_score_test_6
```

Out[106]:
```
(array([0.91859597, 0.66428571]),
 array([0.92900302, 0.63050847]),
 array([0.92377018, 0.64695652]))
```

In [107]:
```
accuracy_6 = accuracy_score(y_test_ohe_drop,y_pred_6)
accuracy_6
```

Out[107]: 0.8746139592340951

In [108]:
```
TP, FP, FN, TN = get_data(c6)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[108]: (0.5707897743968134, 0.5710770538242703)

Case 7:Drop掉與預測目標不太相關的特徵-2

abs(corr)<=0.05

Split x and y

In [109]:
```
y_ohe_drop_2=df_new_2['Revenue'].values
X_ohe_drop_2=df_new_2.drop(['ID','Revenue','Month_0','Region_8.0','Region_3.0','Browser_1','ProductRelated_Duration','
                            'VisitorType_0.0','Region_5.0','Region_4.0','Region_0.0','Browser_2','Weekend_1','OperatingS
                            'Browser_0','OperatingSystems_2','Region_1.0','Region_2.0','Region_7.0','Weekend_0','Month_4
```

分割訓練集 / 測試集 80/20

In [110]:
```
x_train_ohe_drop_2, x_test_ohe_drop_2, y_train_ohe_drop_2, y_test_ohe_drop_2 = train_test_split(X_ohe_drop_2, y_ohe_dr
```

In [111]:
```
#特徵標準化
rob_scaler_4 = RobustScaler()
rob_data_4 = x_train_ohe_drop_2.copy()


rob_data_4['rob_scaled_Administrative'] = rob_scaler_4.fit_transform(rob_data_4['Administrative'].values.reshape(-1,1)
rob_data_4['rob_scaled_Administrative_Duration'] = rob_scaler_4.fit_transform(rob_data_4['Administrative_Duration'].va
rob_data_4['rob_scaled_Informational'] = rob_scaler_4.fit_transform(rob_data_4['Informational'].values.reshape(-1,1))
rob_data_4['rob_scaled_Informational_Duration'] = rob_scaler_4.fit_transform(rob_data_4['Informational_Duration'].valu
rob_data_4['rob_scaled_ProductRelated'] = rob_scaler_4.fit_transform(rob_data_4['ProductRelated'].values.reshape(-1,1)

rob_data_4['rob_scaled_BounceRates'] = rob_scaler_4.fit_transform(rob_data_4['BounceRates'].values.reshape(-1,1))
rob_data_4['rob_scaled_ExitRates'] = rob_scaler_4.fit_transform(rob_data_4['ExitRates'].values.reshape(-1,1))
rob_data_4['rob_scaled_PageValues'] = rob_scaler_4.fit_transform(rob_data_4['PageValues'].values.reshape(-1,1))
rob_data_4['rob_scaled_SpecialDay'] = rob_scaler_4.fit_transform(rob_data_4['SpecialDay'].values.reshape(-1,1))




rob_data_4.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelated',

rob_data_4.describe()
```

Out[111]:

	Month_1	Month_2	Month_3	OperatingSystems_1	OperatingSystems_3	TrafficType_1	TrafficType_2	TrafficType_3	TrafficType_!
count	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.000000	6476.00000(
mean	0.157505	0.274707	0.237492	0.536288	0.203366	0.314855	0.205683	0.167542	0.09218i
std	0.364304	0.446401	0.425579	0.498720	0.402534	0.464494	0.404231	0.373488	0.28931'
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000(
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000(
50%	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.00000(
75%	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.00000(
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000(

8 rows × 21 columns

In [112]:

```
rob_scaler_4 = RobustScaler()
rob_data_test_4 = x_test_ohe_drop_2.copy()


rob_data_test_4['rob_scaled_Administrative'] = rob_scaler_4.fit_transform(rob_data_test_4['Administrative'].values.res
rob_data_test_4['rob_scaled_Administrative_Duration'] = rob_scaler_4.fit_transform(rob_data_test_4['Administrative_Dur
rob_data_test_4['rob_scaled_Informational'] = rob_scaler_4.fit_transform(rob_data_test_4['Informational'].values.resha
rob_data_test_4['rob_scaled_Informational_Duration'] = rob_scaler_4.fit_transform(rob_data_test_4['Informational_Durat
rob_data_test_4['rob_scaled_ProductRelated'] = rob_scaler_4.fit_transform(rob_data_test_4['ProductRelated'].values.res

rob_data_test_4['rob_scaled_BounceRates'] = rob_scaler_4.fit_transform(rob_data_test_4['BounceRates'].values.reshape(-
rob_data_test_4['rob_scaled_ExitRates'] = rob_scaler_4.fit_transform(rob_data_test_4['ExitRates'].values.reshape(-1,1)
rob_data_test_4['rob_scaled_PageValues'] = rob_scaler_4.fit_transform(rob_data_test_4['PageValues'].values.reshape(-1,
rob_data_test_4['rob_scaled_SpecialDay'] = rob_scaler_4.fit_transform(rob_data_test_4['SpecialDay'].values.reshape(-1,



rob_data_test_4.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRela

rob_data_test_4.describe()
```

Out[112]:

	Month_1	Month_2	Month_3	OperatingSystems_1	OperatingSystems_3	TrafficType_1	TrafficType_2	TrafficType_3	TrafficType_5
count	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000	1619.000000
mean	0.151328	0.261272	0.254478	0.544163	0.200741	0.319951	0.197653	0.157505	0.095120
std	0.358479	0.439464	0.435702	0.498200	0.400679	0.466601	0.398352	0.364388	0.293472
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	1.000000	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 21 columns

In [113]:

```
##特徵pca
pca_4 = PCA(n_components='mle',random_state=1)
pca_4.fit(rob_data_4)
x_train_pca_4 = pca_4.transform(rob_data_4)
x_teset_pca_4 = pca_4.transform(rob_data_test_4)

x_train_pca_4 = pd.DataFrame(x_train_pca_4)
x_teset_pca_4 = pd.DataFrame(x_teset_pca_4)
```

In [114]:

```
x_teset_pca_4
```

Out[114]:

	0	1	2	3	4	5	6	7	8	9	10	11	1
0	-31.952697	-5.531152	-0.839517	-1.094184	0.331764	-0.191138	0.014944	-0.117137	1.073829	-0.034399	-0.168373	-0.502126	0.46515
1	-31.866885	-5.373834	1.585956	12.107654	7.258218	-1.131360	1.299106	1.935907	0.767495	-0.117988	-0.512014	0.303544	0.06719
2	-31.947493	-5.528549	-0.801398	-0.579716	-0.687032	0.896977	0.487988	0.037299	1.002908	-0.434233	0.149955	0.162001	0.14331
3	-31.927067	-5.487116	1.562641	-1.175038	0.475974	-0.261232	-0.035762	0.131449	0.177635	-0.789705	0.561855	-0.109251	0.29278
4	-31.919518	-5.454273	1.631507	0.233821	-0.356186	-1.155529	0.263585	0.190352	-0.373476	0.424358	0.075449	-0.424127	-0.40270
...	...	...	...	...	...	...	...	...	...	...	...	...	
1614	194.113609	0.086589	-1.125780	0.573075	-1.260381	0.165643	-1.240876	-0.657280	-0.709960	0.390634	0.046075	-0.312097	-0.42337
1615	-31.931875	-5.485308	0.846534	-0.675678	-0.242526	-0.618668	0.160629	0.227886	-0.397274	0.554761	-0.191433	-0.118236	-0.77180
1616	-31.924512	-5.471272	1.318278	-0.115548	-0.097936	-0.710252	0.137205	-0.372244	0.220426	-0.268508	-0.165833	-0.084698	-0.01733
1617	-31.946382	-5.525128	-0.304291	-1.138073	0.407614	0.274769	0.083089	0.060700	0.354005	-0.052419	-0.058991	0.005685	-0.03579
1618	-31.943304	-5.513061	-0.136472	-0.830748	-0.187949	-0.163329	0.188683	0.147976	0.628873	-0.964739	-0.008929	0.188476	-0.37695

1619 rows × 20 columns

In [115]:
```
# summarize class distribution
counter = Counter(y_train_ohe_2)
print(counter)
# define pipeline
over = ADASYN(sampling_strategy=1,random_state=1)
#under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over)]
pipeline = Pipeline(steps=steps)
# transform the dataset
OU_X_4, OU_y_4 = pipeline.fit_resample(x_train_pca_4, y_train_ohe_drop_2)
# summarize the new class distribution
counter = Counter(OU_y_4)
print(counter)
```

```
Counter({0: 5479, 1: 997})
Counter({1: 5518, 0: 5479})
```

In [116]:
```
xgbc_7 =XGBClassifier()
xgbc_7.fit(OU_X_4 , OU_y_4)
xgbc_7.score(OU_X_4 , OU_y_4)
```

```
[05:47:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XG
Boost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'lo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warn
ing, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

Out[116]:
```
0.99772665272347
```

In [117]:
```
xgbc_7.score(x_teset_pca_4 , y_test_ohe_drop_2)
```

Out[117]:
```
0.8610253242742434
```

In [118]:
```
y_pred_7 = xgbc_7.predict(x_teset_pca_4)
recall_test_7 = recall_score(np.array(y_test_ohe_drop_2), y_pred_7, average=None)
precision_test_7 = precision_score(np.array(y_test_ohe_drop_2), y_pred_7, average=None)
f1_score_test_7 = f1_score(np.array(y_test_ohe_drop_2), y_pred_7, average=None)
c7 = confusion_matrix(y_test_ohe_drop_2,y_pred_7)
c7
```

Out[118]:
```
array([[1208,  131],
       [  94,  186]], dtype=int64)
```

In [119]:
```
recall_test_7, precision_test_7, f1_score_test_7
```

Out[119]:
```
(array([0.9021658 , 0.66428571]),
 array([0.92780338, 0.58675079]),
 array([0.914805  , 0.62311558]))
```

In [120]:
```
accuracy_7 = accuracy_score(y_test_ohe_drop_2,y_pred_7)
accuracy_7
```

Out[120]:
```
0.8610253242742434
```

In [121]:
```
TP, FP, FN, TN = get_data(c7)
cohen_kappa(TP, FP, FN, TN), mcc(TP, FP, FN, TN)
```

Out[121]:
```
(0.5383214431011517, 0.5398796025939093)
```

根據上述case, 採用Case 9: 合併特徵類別 no PCA

因其kappa、mcc為最高，表示模型較好

In [133]:
```
test = pd.read_csv("data-question/test.csv")
```

In [ ]:
```
```

In [136]:
```
test2 = test.copy()
test2['Month'] = test['Month'].apply(new_month)
test2['OperatingSystems'] = test['OperatingSystems'].apply(new_OperatingSystems)
test2['Browser'] = test['Browser'].apply(new_browser)
test2['TrafficType'] = test['TrafficType'].apply(new_TrafficType)
```

In [137]:
```python
# 對類別型特徵做One-hot-encoding
df_str_test = test2.astype({'Month':'category','OperatingSystems':'category','Browser':'category','Region':'category',
df_str_test
df_dum_test = pd.get_dummies(df_str_test[['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','V

df_str_test.drop(['Month','OperatingSystems','Browser','Region','TrafficType','VisitorType','Weekend'], axis=1, inplac

df_new_test = pd.concat([df_dum_test,df_str_test],axis=1)
df_new_test
```

Out[137]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSystems_3	Brow
0	0	0	0	1	0	0	0	1	0	
1	0	0	0	0	1	0	1	0	0	
2	0	0	0	1	0	0	0	1	0	
3	0	0	1	0	0	0	1	0	0	
4	0	1	0	0	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	
895	0	0	1	0	0	0	1	0	0	
896	0	0	0	1	0	0	1	0	0	
897	0	0	1	0	0	0	0	1	0	
898	0	0	0	1	0	0	0	1	0	
899	0	0	1	0	0	0	0	1	0	

900 rows × 44 columns

In [142]:
```python
df_new_test.describe()
```

Out[142]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSys
count	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900
mean	0.207778	0.158889	0.251111	0.253333	0.128889	0.046667	0.524444	0.215556	0
std	0.405942	0.365776	0.433893	0.435162	0.335263	0.211041	0.499680	0.411436	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0
75%	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

8 rows × 44 columns

In [146]:
```python
X_ohe_test_ID = df_new_test['ID'].values
X_ohe_test = df_new_test.drop(['ID'],axis=1)
```

In [147]:
```python
#特徵標準化
rob_scaler_test = RobustScaler()
rob_data_test = X_ohe_test.copy()


rob_data_test['rob_scaled_Administrative'] = rob_scaler_test.fit_transform(rob_data_test['Administrative'].values.resh
rob_data_test['rob_scaled_Administrative_Duration'] = rob_scaler_test.fit_transform(rob_data_test['Administrative_Dura
rob_data_test['rob_scaled_Informational'] = rob_scaler_test.fit_transform(rob_data_test['Informational'].values.reshap
rob_data_test['rob_scaled_Informational_Duration'] = rob_scaler_test.fit_transform(rob_data_test['Informational_Durati
rob_data_test['rob_scaled_ProductRelated'] = rob_scaler_test.fit_transform(rob_data_test['ProductRelated'].values.resh
rob_data_test['rob_scaled_ProductRelated_Duration'] = rob_scaler_test.fit_transform(rob_data_test['ProductRelated_Dura
rob_data_test['rob_scaled_BounceRates'] = rob_scaler_test.fit_transform(rob_data_test['BounceRates'].values.reshape(-1
rob_data_test['rob_scaled_ExitRates'] = rob_scaler_test.fit_transform(rob_data_test['ExitRates'].values.reshape(-1,1))
rob_data_test['rob_scaled_PageValues'] = rob_scaler_test.fit_transform(rob_data_test['PageValues'].values.reshape(-1,1
rob_data_test['rob_scaled_SpecialDay'] = rob_scaler_test.fit_transform(rob_data_test['SpecialDay'].values.reshape(-1,1




rob_data_test.drop(['Administrative','Administrative_Duration','Informational','Informational_Duration','ProductRelate

rob_data_test.describe()
```

Out[147]:

	Month_0	Month_1	Month_2	Month_3	Month_4	OperatingSystems_0	OperatingSystems_1	OperatingSystems_2	OperatingSys
count	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900.000000	900
mean	0.207778	0.158889	0.251111	0.253333	0.128889	0.046667	0.524444	0.215556	0
std	0.405942	0.365776	0.433893	0.435162	0.335263	0.211041	0.499680	0.411436	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0
75%	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

8 rows × 43 columns

In [148]:
```python
y_pred_test = xgbc_9.predict(rob_data_test)
```

In [167]:
```python
X_ohe_test_ID = pd.DataFrame(X_ohe_test_ID)
y_pred_test = pd.DataFrame(y_pred_test)
result = pd.concat([X_ohe_test_ID,y_pred_test],axis=1)
result.columns = ['ID','HasRevenue']
result
```

Out[167]:

	ID	HasRevenue
0	6162	0
1	8143	0
2	5571	1
3	3933	0
4	934	0
...	...	...
895	5887	0
896	5273	0
897	5833	0
898	2119	0
899	4448	0

900 rows × 2 columns

In [170]:
```python
from pathlib import Path
filepath = Path('output/out.csv')
filepath.parent.mkdir(parents=True, exist_ok=True)
result.to_csv(filepath, index = False)
```

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```