Importing libraries and datasets

```
import numpy as np
import pandas as pd
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score,
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from xgboost import plot_importance
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold # import KFold
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
```

```
df_response = pd.read_csv('Retail_Data_Response.csv')
df_transactions = pd.read_csv('Retail_Data_Transactions.csv', parse_dates=['trans_
```

```
df_response.head()
```

	customer_id	response
0	CS1112	0
1	CS1113	0
2	CS1114	1
3	CS1115	1
4	CS1116	1

```
df_transactions.head()
```

	customer_id	trans_date	tran_amount
0	CS5295	2013-02-11	35
1	CS4768	2015-03-15	39
2	CS2122	2013-02-26	52
3	CS1217	2011-11-16	99
4	CS1850	2013-11-20	78

```
print(df_transactions['trans_date'].min())
print(df_transactions['trans_date'].max())
```

```
    2011-05-16 00:00:00
    2015-03-16 00:00:00
```

Data Preparation

```
## since the last date of the data is 16 March 2015, the campaign date is assumed t
## RFM model will be used to predict campaign response. Recency is calculated

campaign_date = dt.datetime(2015,3,17)
df_transactions['recent']= campaign_date - df_transactions['trans_date']
df_transactions['recent'].astype('timedelta64[D]')
df_transactions['recent']=df_transactions['recent'] / np.timedelta64(1, 'D')
df_transactions.head()
```

	customer_id	trans_date	tran_amount	recent
0	CS5295	2013-02-11	35	764.0
1	CS4768	2015-03-15	39	2.0
2	CS2122	2013-02-26	52	749.0
3	CS1217	2011-11-16	99	1217.0
4	CS1850	2013-11-20	78	482.0

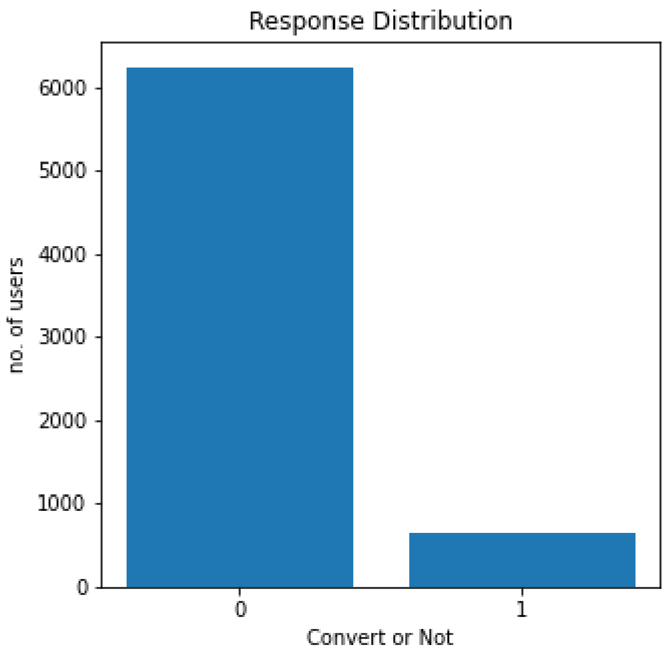
```
## create data set with RFM variables

df_rfm = df_transactions.groupby('customer_id').agg({'recent': lambda x:x.min(),
                                                      'customer_id': lambda x: len(x
                                                      'tran_amount': lambda x: x.sur

df_rfm.rename(columns={'recent': 'recency',
                        'customer_id': 'frequency',
                        'tran_amount': 'monetary_value'}, inplace=True)


df_rfm = df_rfm.reset_index()
df_rfm.head()
```

Calculating response rate

```
response_rate = df_response.groupby('response').agg({'customer_id': lambda x: len(
response_rate.head()
```

	response	customer_id
0	0	6237
1	1	647

```
plt.figure(figsize=(5,5))
x=range(2)
plt.bar(x,response_rate['customer_id'])
plt.xticks(response_rate.index)
plt.title('Response Distribution')
plt.xlabel('Convert or Not')
plt.ylabel('no. of users')
plt.show()

## data is imbalanced
```



```
## merging two data sets

df_modeling = pd.merge(df_response,df_rfm)
df_modeling.head()
```

	customer_id	response	recency	frequency	monetary_value
0	CS1112	0	62.0	15	1012
1	CS1113	0	36.0	20	1490

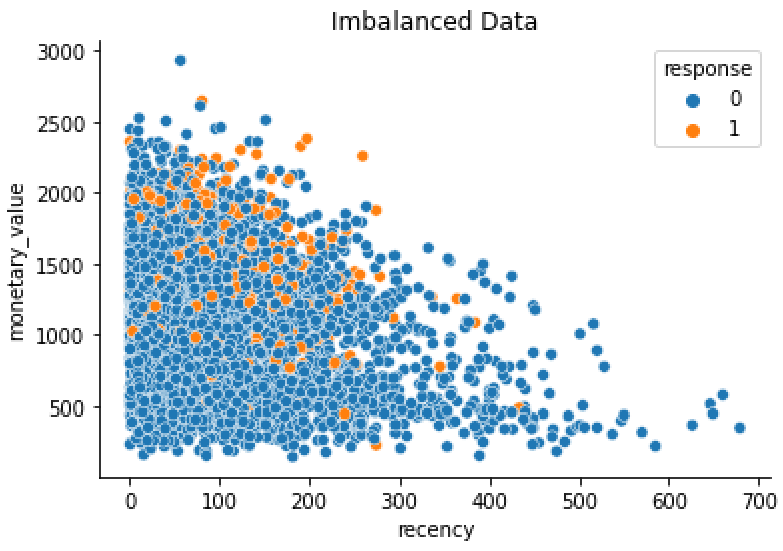
Creating train and test dataset

```
## spliting dataframe into X and y

X = df_modeling.drop(columns=['response','customer_id'])
y = df_modeling['response']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
    Number transactions X_train dataset:  (4818, 3)
    Number transactions y_train dataset:  (4818,)
    Number transactions X_test dataset:  (2066, 3)
    Number transactions y_test dataset:  (2066,)
```

```
sns.scatterplot(data=df_modeling, x='recency', y='monetary_value', hue='response')
sns.despine()
plt.title("Imbalanced Data")
```

```
    Text(0.5, 1.0, 'Imbalanced Data')
```



Fixing imbalanced with Undersampling

[ ] ↳ 1 cell hidden

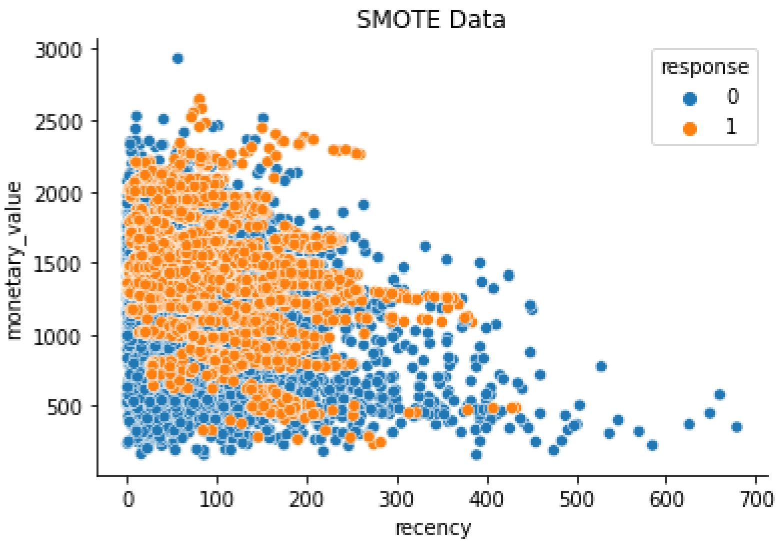
Fixing imbalanced with Oversampling

[ ] ↳ *1 cell hidden*

Fixing imbalanced with SMOTE

```
sm = SMOTE(random_state=0)
sm.fit(X_train, y_train)
X_SMOTE, y_SMOTE = sm.fit_sample(X_train, y_train)
df_SMOTE = pd.concat([pd.DataFrame(data=X_SMOTE),pd.DataFrame(data=y_SMOTE)], axis=
df_SMOTE.columns= ['recency', 'frequency', 'monetary_value', 'response']

sns.scatterplot(data=df_SMOTE, x='recency', y='monetary_value', hue='response')
sns.despine()
plt.title("SMOTE Data")
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futur
  warnings.warn(msg, category=FutureWarning)
Text(0.5, 1.0, 'SMOTE Data')
```



Logistic Regression Model

[ ] ↳ *9 cells hidden*

XGBoost

[ ] ↳ *10 cells hidden*

Improve

Import File and Create Feature

ลองเพิ่ม Feature Age of Transaction แต่พบว่า ไม่ได้ทำให้ AUC เพิ่มขึ้น

```
df_response = pd.read_csv('Retail_Data_Response.csv')
df_transactions = pd.read_csv('Retail_Data_Transactions.csv', parse_dates=['trans_c

campaign_date = dt.datetime(2015,3,17)
df_transactions['recent']= campaign_date - df_transactions['trans_date']
df_transactions['recent'].astype('timedelta64[D]')
df_transactions['recent']=df_transactions['recent'] / np.timedelta64(1, 'D')
df_transactions.head()


#Add More feature
campaign_date = dt.datetime(2015,3,17)
df_transactions['first_transaction']= campaign_date - df_transactions['trans_date']
df_transactions['first_transaction'].astype('timedelta64[D]')
df_transactions['first_transaction']=df_transactions['first_transaction'] / np.time
df_transactions['age_of_T']= campaign_date - df_transactions['trans_date']
df_transactions['age_of_T'].astype('timedelta64[D]')
df_transactions['age_of_T']=df_transactions['age_of_T'] / np.timedelta64(1, 'D')
df_transactions.head()
```

	customer_id	trans_date	tran_amount	recent	first_transaction	age_of_T
0	CS5295	2013-02-11	35	764.0	764.0	764.0
1	CS4768	2015-03-15	39	2.0	2.0	2.0
2	CS2122	2013-02-26	52	749.0	749.0	749.0
3	CS1217	2011-11-16	99	1217.0	1217.0	1217.0
4	CS1850	2013-11-20	78	482.0	482.0	482.0

```
## create data set with RFM variables

df_rfm = df_transactions.groupby('customer_id').agg({'recent': lambda x:x.min(),
                                                      'customer_id': lambda x: len(x
                                                      'tran_amount': lambda x: x.sum
                                                      'first_transaction': lambda x:
                                                      'age_of_T': lambda x:x.max()-x
df_rfm.rename(columns={'recent': 'recency',
                         'customer_id': 'frequency',
                         'tran_amount': 'monetary_value',
                         'first_transaction': 'first_transaction',
                         'age_of_T': 'age_of_T'
                       }, inplace=True)


df_rfm['rfm'] = df_rfm['recency'] * df_rfm['frequency'] * df_rfm['monetary_value']
```

```
df_rfm.head(2)
```

	recency	frequency	monetary_value	first_transaction	age_of_T
customer_id					
CS1112	62.0	15	1012	1371.0	1309.0
CS1113	36.0	20	1490	1390.0	1354.0

```
df_rfm = df_rfm.reset_index()
df_rfm.head()
```

	customer_id	recency	frequency	monetary_value	first_transaction	age_of_
0	CS1112	62.0	15	1012	1371.0	1309.
1	CS1113	36.0	20	1490	1390.0	1354.
2	CS1114	33.0	19	1432	1342.0	1309.
3	CS1115	12.0	22	1659	1315.0	1303.
4	CS1116	204.0	13	857	1359.0	1155.

```
df_modeling = pd.merge(df_response,df_rfm)
df_modeling.head()
```

	customer_id	response	recency	frequency	monetary_value	first_transactio
0	CS1112	0	62.0	15	1012	1371.
1	CS1113	0	36.0	20	1490	1390.
2	CS1114	1	33.0	19	1432	1342.
3	CS1115	1	12.0	22	1659	1315.
4	CS1116	1	204.0	13	857	1359.

```
## spliting dataframe into X and y

X = df_modeling.drop(columns=['first_transaction','age_of_T','rfm','response','cust
# X = df_modeling[['age_of_T','rfm']]
y = df_modeling['response']
```

```
df_modeling.describe()
```

	response	recency	frequency	monetary_value	first_transaction
count	6884.000000	6884.000000	6884.000000	6884.000000	6884.000000
mean	0.093986	81.024985	18.153544	1179.892214	1322.488379
std	0.291831	83.251016	5.184476	465.421365	84.553118
min	0.000000	1.000000	4.000000	149.000000	507.000000
max	1.000000	679.000000	39.000000	2933.000000	1401.000000

```
df_modeling.columns
```

```
Index(['customer_id', 'response', 'recency', 'frequency', 'monetary_value',
       'first_transaction', 'age_of_T', 'rfm'],
      dtype='object')
```

Explore

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (4818, 3)
Number transactions y_train dataset:  (4818,)
Number transactions X_test dataset:  (2066, 3)
Number transactions y_test dataset:  (2066,)
```

```
sm = SMOTE(random_state=0)
sm.fit(X_train, y_train)
X_SMOTE, y_SMOTE = sm.fit_sample(X_train, y_train)
df_SMOTE_imp = pd.concat([pd.DataFrame(data=X_SMOTE),pd.DataFrame(data=y_SMOTE)], a
df_SMOTE_imp.columns= ['recency', 'frequency', 'monetary_value', 'response']
# df_SMOTE_imp.columns= ['age_of_T','rfm', 'response']
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futur
  warnings.warn(msg, category=FutureWarning)
```

```
df_SMOTE_imp.head(5)
```

	recency	frequency	monetary_value	response
0	115.0	12.0	564.0	0

```
df_SMOTE_imp['response'].unique()
```

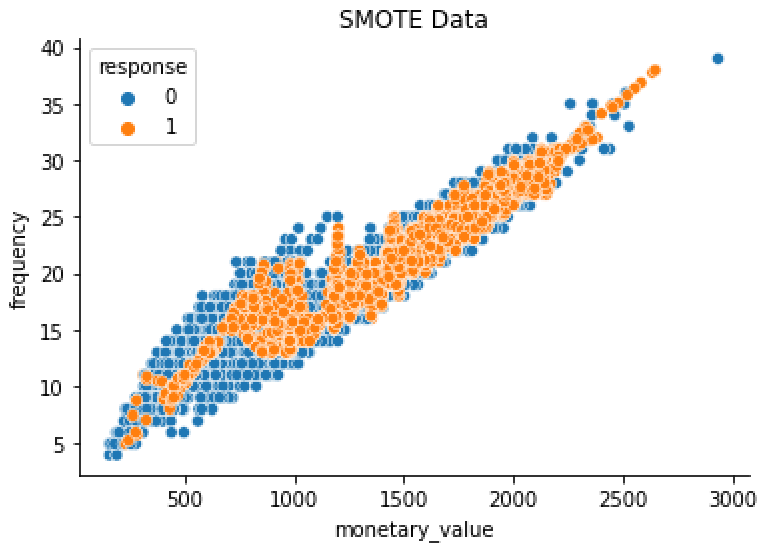
```
array([0, 1])
```

```
df_SMOTE_imp.describe()
```

	recency	frequency	monetary_value	response
count	8778.000000	8778.000000	8778.000000	8778.000000
mean	82.329037	19.546411	1311.604887	0.500000
std	75.408961	4.969402	439.465126	0.500028
min	1.000000	4.000000	157.000000	0.000000
25%	27.395891	16.000000	1048.000000	0.000000
50%	62.762745	20.000000	1376.754041	0.500000
75%	115.295107	23.000000	1607.000000	1.000000
max	679.000000	39.000000	2933.000000	1.000000

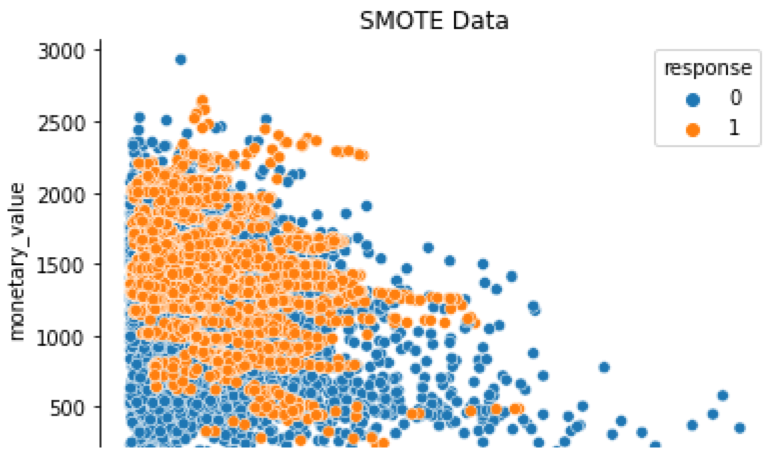
```
sns.scatterplot(data=df_SMOTE_imp, x='monetary_value', y='frequency', hue='response
sns.despine()
plt.title("SMOTE Data")
```

```
Text(0.5, 1.0, 'SMOTE Data')
```



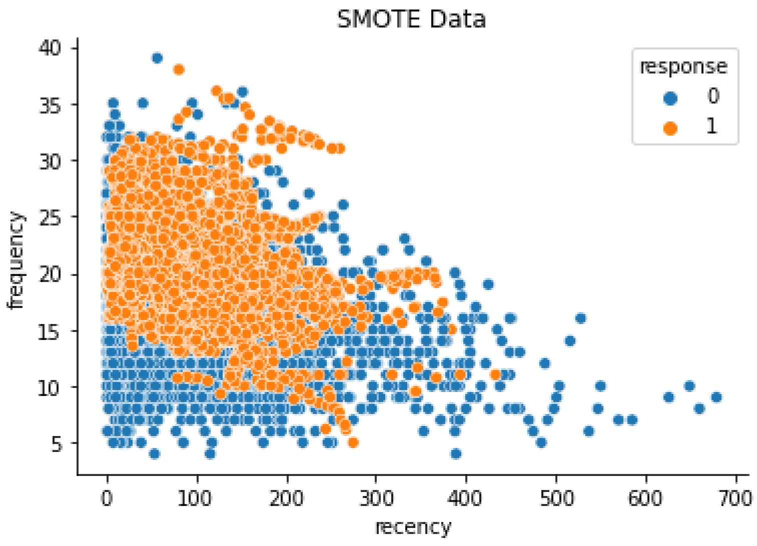
```
sns.scatterplot(data=df_SMOTE_imp, x='recency', y='monetary_value', hue='response'
sns.despine()
plt.title("SMOTE Data")
```

```
Text(0.5, 1.0, 'SMOTE Data')
```



```
sns.scatterplot(data=df_SMOTE_imp, x='recency', y='frequency', hue='response')
sns.despine()
plt.title("SMOTE Data")
```

```
Text(0.5, 1.0, 'SMOTE Data')
```



```
#Check Imbalance
df_SMOTE_imp.response.value_counts()
```

```
1    4389
0    4389
Name: response, dtype: int64
```

Tuning Model

ลองแบ่ง ข้อมูลด้วย KFold พบว่า n_splits = 12 จะได้ AUC ดีที่สุด

```
#K-Fold Cross validation
kf = KFold(n_splits=12,shuffle=True,random_state=101)
for train_index, test_index in kf.split(X):
  X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
  y_train, y_test = y[train_index], y[test_index]
  print('Train Shape X: {} Y : {}'.format(X_train.shape,y_train.shape))
  print('Test Shape X: {} Y : {}'.format(X_test.shape,y_test.shape))
```

```
#Fix Imbalance with SMOTE
  sm = SMOTE(random_state=0)
  sm.fit(X_train, y_train)
  X_SMOTE, y_SMOTE = sm.fit_sample(X_train, y_train)

#XGb Model
  xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric="auc",
    base_score=0.5,
    learning_rate =0.10,
    n_estimators=5000,
    max_depth=5,
    min_child_weight=2,
    gamma=0.1,
    subsample=0.4,
    colsample_bytree=0.4,
    nthread=4)
  predicted_y = []
  expected_y = []

  xgb_model_SMOTE = xgb_model.fit(X_SMOTE, y_SMOTE, early_stopping_rounds=5, eval_s
  predictions =  xgb_model_SMOTE.predict(X_SMOTE)
  predicted_y.extend(predictions)
  expected_y.extend(y_SMOTE)
  report_train = classification_report(expected_y, predicted_y)
  print('training set')
  print(report_train)

  predicted_y = []
  expected_y = []
  predictions = xgb_model_SMOTE.predict(X_test.to_numpy())
  predicted_y.extend(predictions)
  expected_y.extend(y_test)
  report_test = classification_report(expected_y, predicted_y)
  print('test set')
  print(report_test)
```

```
[5]     validation_0-auc:0.766772

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futu
  warnings.warn(msg, category=FutureWarning)

training set
              precision    recall  f1-score   support

           0       0.83      0.71      0.77      5715
           1       0.75      0.86      0.80      5715

    accuracy                           0.78     11430
   macro avg       0.79      0.78      0.78     11430
weighted avg       0.79      0.78      0.78     11430

test set
              precision    recall  f1-score   support

           0       0.95      0.74      0.83       522
           1       0.19      0.63      0.29        51

    accuracy                           0.73       573
    accuracy                           0.73       573
   macro avg       0.57      0.68      0.56       573
weighted avg       0.89      0.73      0.79       573

Train Shape X: (6311, 3) Y : (6311,)
Test Shape X: (573, 3) Y : (573,)
[0]     validation_0-auc:0.697561
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]     validation_0-auc:0.749126
[2]     validation_0-auc:0.739911
[3]     validation_0-auc:0.762914
[4]     validation_0-auc:0.771904
[5]     validation_0-auc:0.783966
[6]     validation_0-auc:0.765066
[7]     validation_0-auc:0.760806
[8]     validation_0-auc:0.766052
[9]     validation_0-auc:0.759775
[10]    validation_0-auc:0.753991
Stopping. Best iteration:
[5]     validation_0-auc:0.783966

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futu
  warnings.warn(msg, category=FutureWarning)
training set
              precision    recall  f1-score   support

           0       0.73      0.75      0.74      5706
           1       0.74      0.72      0.73      5706

    accuracy                           0.74     11412
   macro avg       0.74      0.74      0.74     11412
weighted avg       0.74      0.74      0.74     11412

test set
              precision    recall  f1-score   support

           0       0.97      0.72      0.82       531
           1       0.17      0.71      0.27        42
```

Result

Train Shape X: (6311, 3) Y : (6311,) Test Shape X: (573, 3) Y : (573,)

[5] validation_0-auc:0.783966

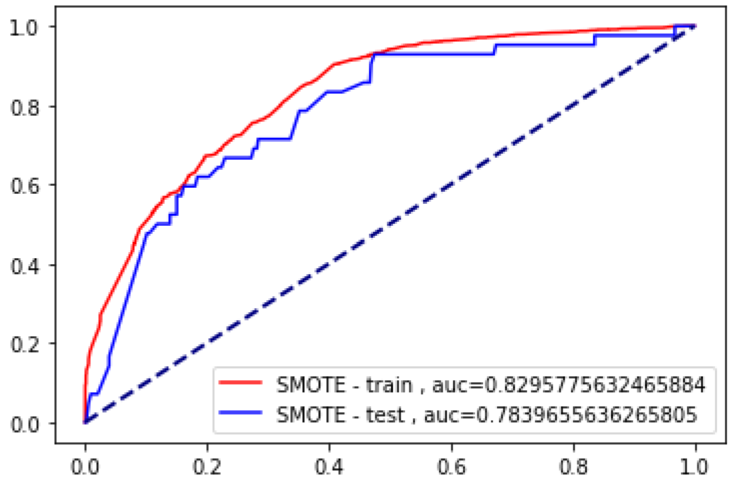
training set precision recall f1-score support

```
       0       0.73      0.75      0.74      5706

       1       0.74      0.72      0.73      5706


accuracy                           0.74     11412

macro avg       0.74      0.74      0.74      11412

weighted avg       0.74      0.74      0.74      11412
```

test set precision recall f1-score support

```
           0       0.97      0.72      0.82       531
           1       0.17      0.71      0.27        42

    accuracy                           0.72       573
   macro avg       0.57      0.71      0.55       573
weighted avg       0.91      0.72      0.78       573
```

```
y_score_train = xgb_model_SMOTE.predict_proba(X_SMOTE)
fpr_train, tpr_train, _ = roc_curve(y_SMOTE,  y_score_train[:,1])
auc_train = roc_auc_score(y_SMOTE, y_score_train[:,1])
plt.plot(fpr_train,tpr_train, color='red', label='SMOTE - train , auc='+str(auc_tra

y_score_test = xgb_model_SMOTE.predict_proba(X_test.to_numpy())
fpr_test, tpr_test, _ = roc_curve(y_test,  y_score_test[:,1])
auc_test = roc_auc_score(y_test, y_score_test[:,1])
plt.plot(fpr_test,tpr_test, color='Blue', label='SMOTE - test , auc='+str(auc_test

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.legend(loc=4)
plt.show()
```



✓ 0s completed at 17:36 ● ×