Importing libraries and datasets

```
In [32]: import numpy as np
import pandas as pd
import datetime as dt

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
import xgboost as xgb
import tensorflow as tf

from sklearn.metrics import precision_score, recall_score, f1_score, auc, roc_auc_score, accuracy_score,
 classification_report, confusion_matrix, roc_curve
from xgboost import plot_importance
```

```
In [3]: df_response = pd.read_csv('Retail_Data_Response.csv')
df_transactions = pd.read_csv('Retail_Data_Transactions.csv', parse_dates=['trans_date'])
```

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

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

```
In [5]: 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

```
In [6]: 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

```
In [7]: ## since the last date of the data is 16 March 2015, the campaign date is assumed to be 17 March 2015
## 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

```
In [8]: ## create data set with RFM variables

df_rfm = df_transactions.groupby('customer_id').agg({'recent': lambda x:x.min(),                      #
 Recency
                                                    'customer_id': lambda x: len(x),                   #
 Frequency
                                                    'tran_amount': lambda x: x.sum()})                 #
 Monetary Value


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

```
In [9]: df_rfm = df_rfm.reset_index()
df_rfm.head()
```

	customer_id	recency	frequency	monetary_value
0	CS1112	62.0	15	1012
1	CS1113	36.0	20	1490
2	CS1114	33.0	19	1432
3	CS1115	12.0	22	1659
4	CS1116	204.0	13	857

```
In [10]: ## create data set with CLV variables

df_clv = df_transactions.groupby('customer_id').agg({'recent': lambda x:x.min(),                      #
 Recency
                                                    'customer_id': lambda x: len(x),                   #
 Frequency
                                                    'tran_amount': lambda x: x.sum(),                  #
 Monetary Value
                                                    'trans_date': lambda x: (x.max() - x.min()).days})
      # AOU

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

df_clv['ticket_size'] = df_clv['monetary_value'] / df_clv['frequency']
```

```
In [11]: df_clv = df_clv.reset_index()
df_clv.head()
```

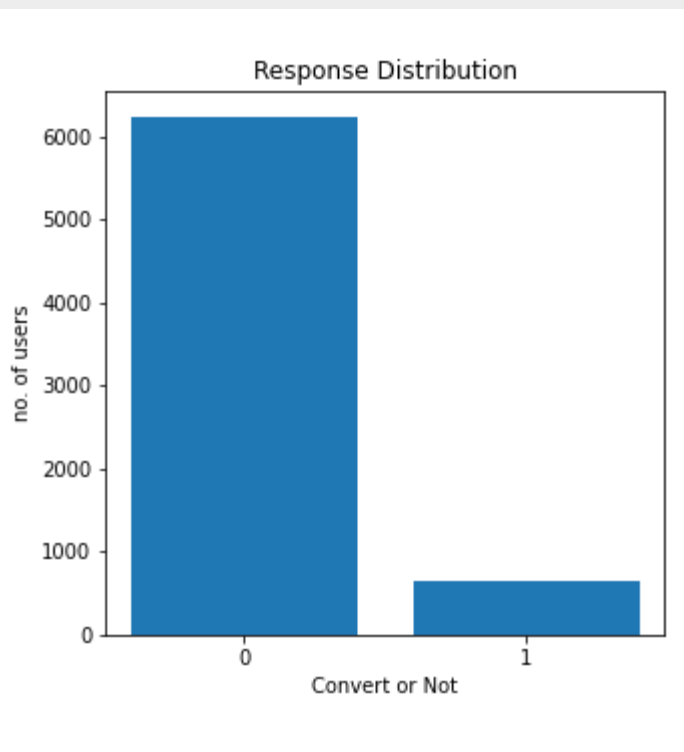
	customer_id	recency	frequency	monetary_value	AOU	ticket_size
0	CS1112	62.0	15	1012	1309	67.466667
1	CS1113	36.0	20	1490	1354	74.500000
2	CS1114	33.0	19	1432	1309	75.368421
3	CS1115	12.0	22	1659	1303	75.409091
4	CS1116	204.0	13	857	1155	65.923077

Calculating response rate

```
In [12]: response_rate = df_response.groupby('response').agg({'customer_id': lambda x: len(x)}).reset_index()
response_rate.head()
```

	response	customer_id
0	0	6237
1	1	647

```
In [13]: 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
```




```
In [14]: ## merging two data sets - RFM

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

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

```
In [15]: ## merging two data sets - CLV

df_modeling_clv = pd.merge(df_response,df_clv)
df_modeling_clv.head()
```

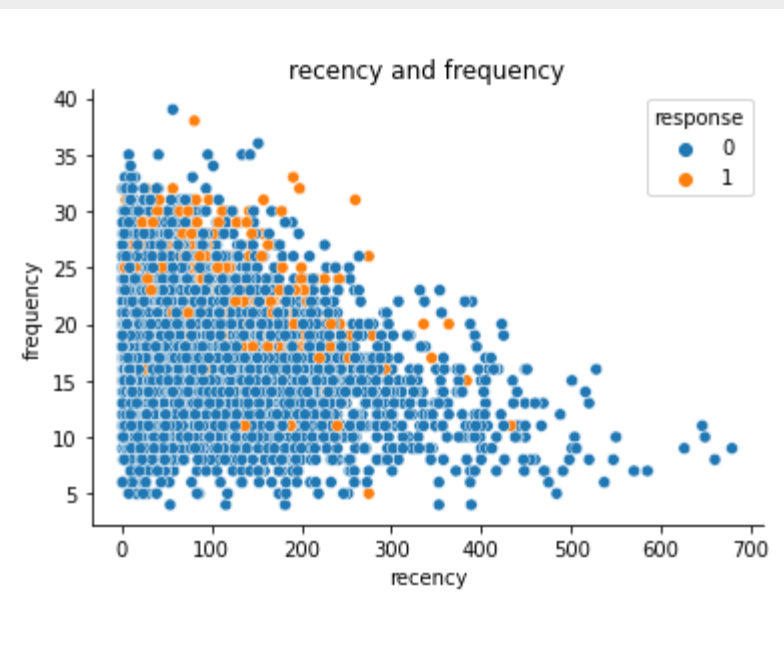
	customer_id	response	recency	frequency	monetary_value	AOU	ticket_size
0	CS1112	0	62.0	15	1012	1309	67.466667
1	CS1113	0	36.0	20	1490	1354	74.500000
2	CS1114	1	33.0	19	1432	1309	75.368421
3	CS1115	1	12.0	22	1659	1303	75.409091
4	CS1116	1	204.0	13	857	1155	65.923077

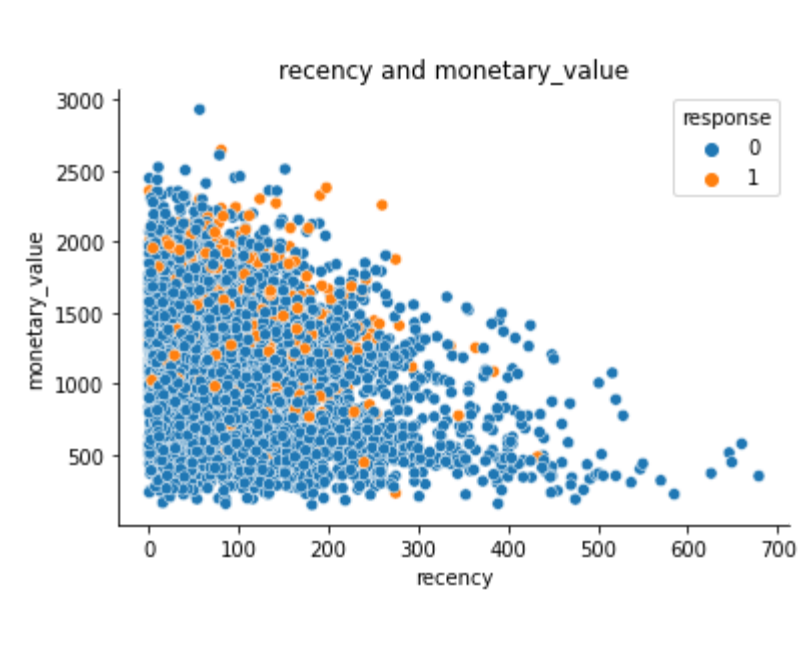
Creating train and test dataset

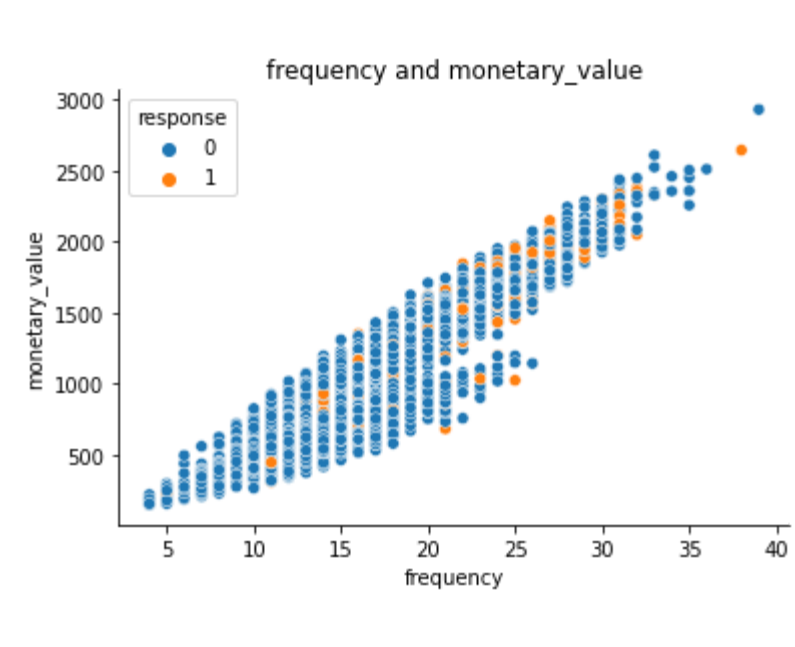
```
In [16]: ## spliting dataframe into X and y

X_rfm = df_modeling_rfm.drop(columns=['response','customer_id'])
y_rfm = df_modeling_rfm['response']

X_clv = df_modeling_clv.drop(columns=['response','customer_id'])
y_clv = df_modeling_clv['response']
```

```
In [17]: ## creating train and test dataset

X_train_rfm, X_test_rfm, y_train_rfm, y_test_rfm = train_test_split(X_rfm, y_rfm, test_size=0.3, random_
state=0)
X_train_clv, X_test_clv, y_train_clv, y_test_clv = train_test_split(X_clv, y_clv, test_size=0.3, random_
state=0)
```

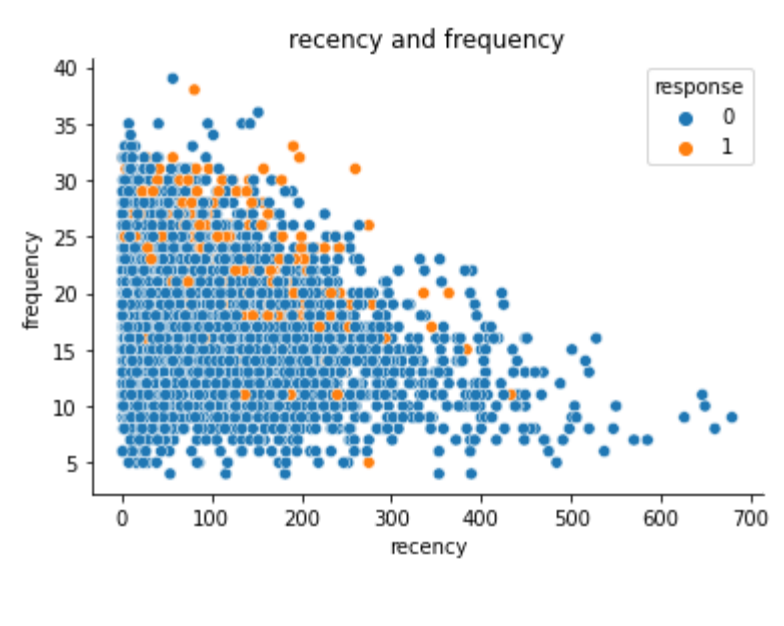
```
In [18]: for i, col_i in enumerate(df_modeling_rfm[['recency', 'frequency', 'monetary_value']].columns):
  for j, col_j in enumerate(df_modeling_rfm[['recency', 'frequency', 'monetary_value']].columns):
    if i < j :
      plt.title(col_i + ' and ' + col_j)
      sns.scatterplot(data=df_modeling_rfm, x=col_i, y=col_j, hue='response')
      sns.despine()
      plt.show()
```

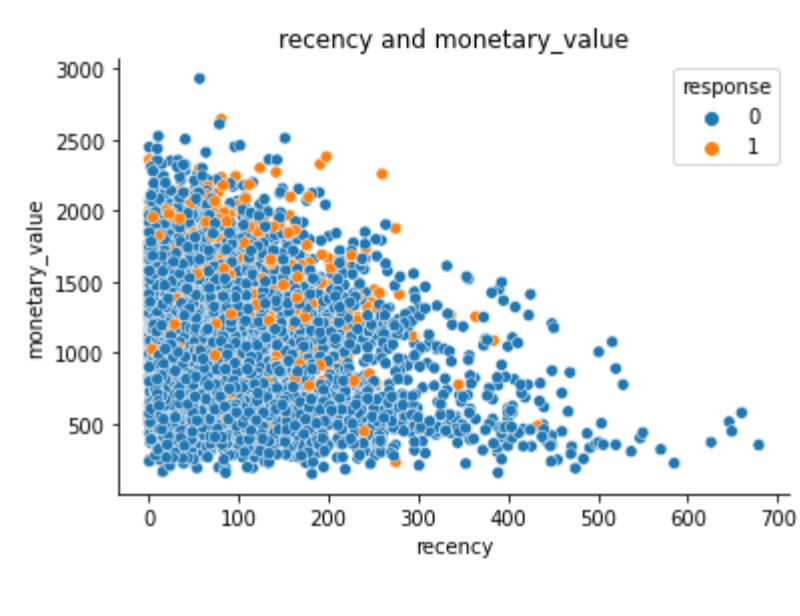


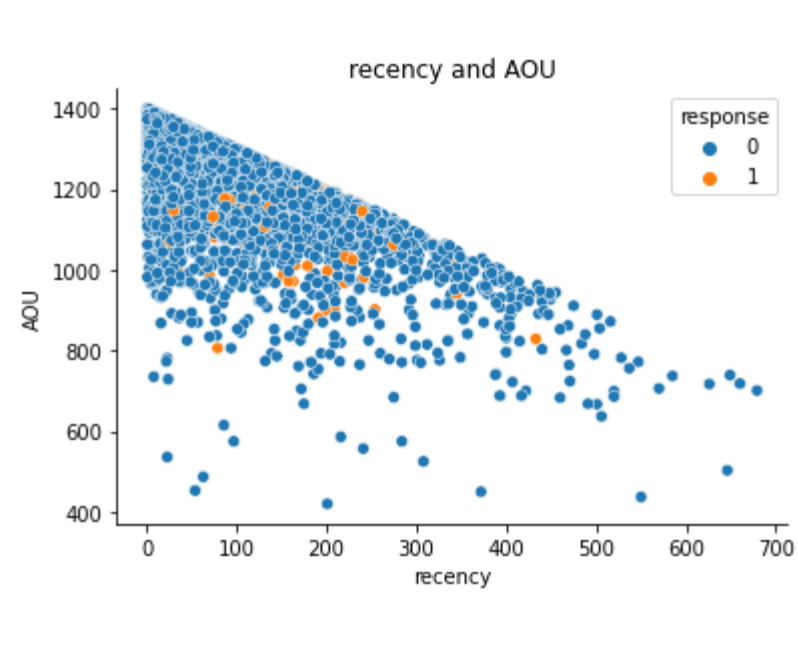


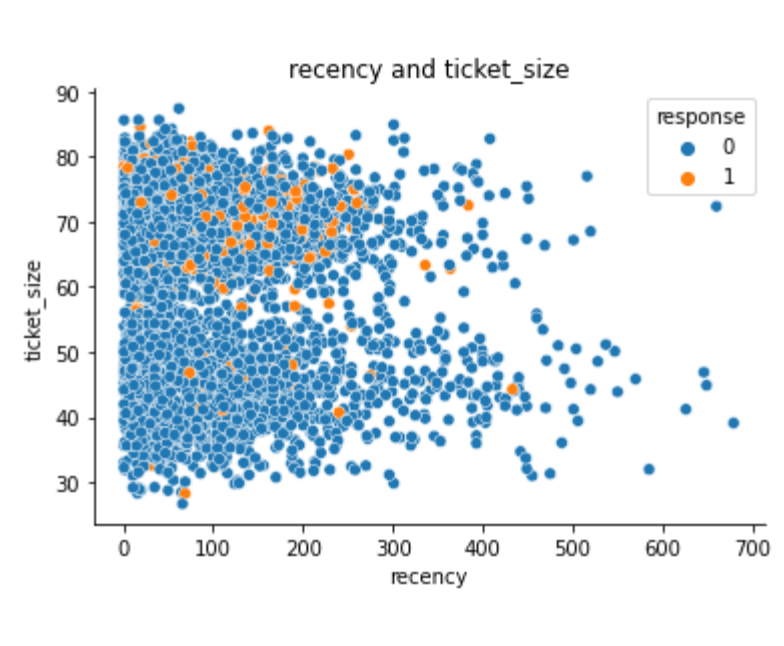


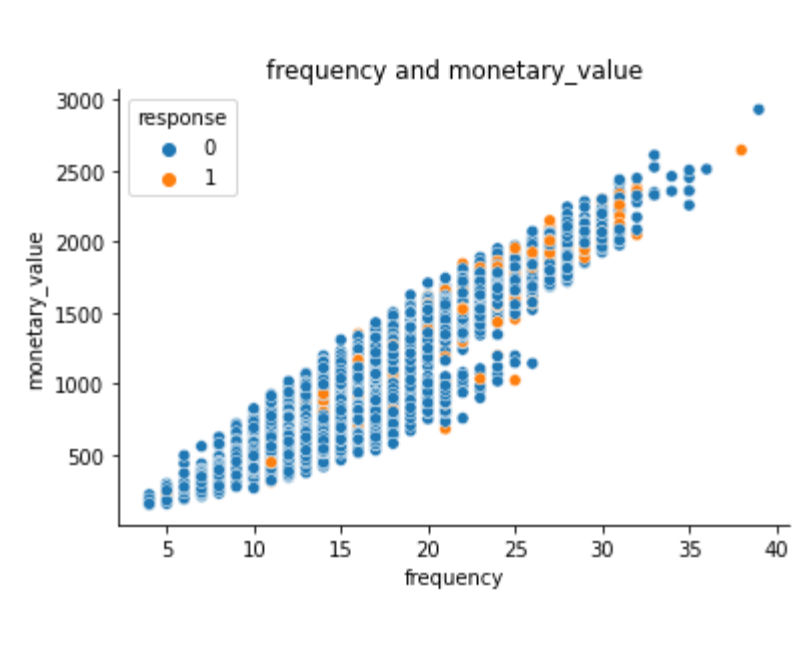

```
In [19]: for i, col_i in enumerate(df_modeling_clv[['recency', 'frequency', 'monetary_value', 'AOU', 'ticket_siz
e']].columns):
  for j, col_j in enumerate(df_modeling_clv[['recency', 'frequency', 'monetary_value', 'AOU', 'ticket_si
ze']].columns):
    if i < j :
      plt.title(col_i + ' and ' + col_j)
      sns.scatterplot(data=df_modeling_clv, x=col_i, y=col_j, hue='response')
      sns.despine()
      plt.show()
```

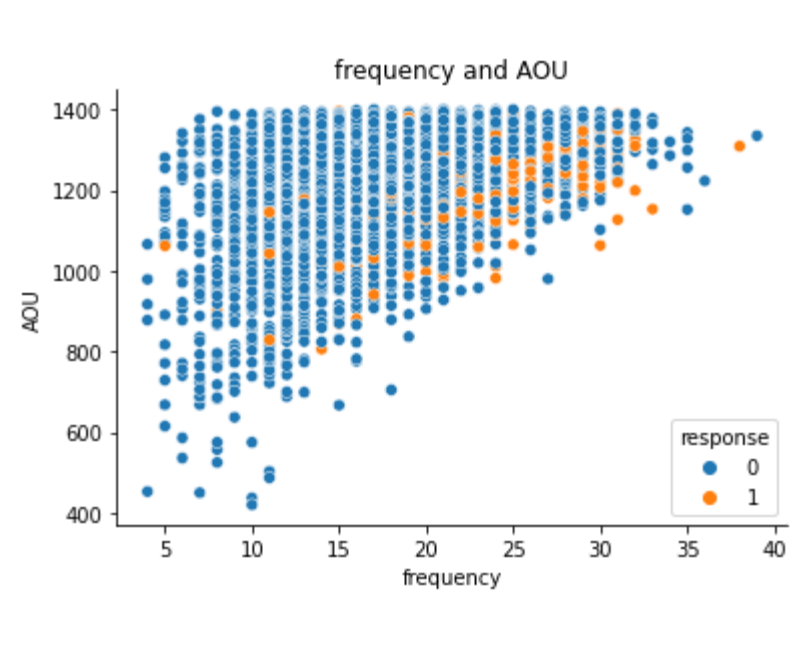


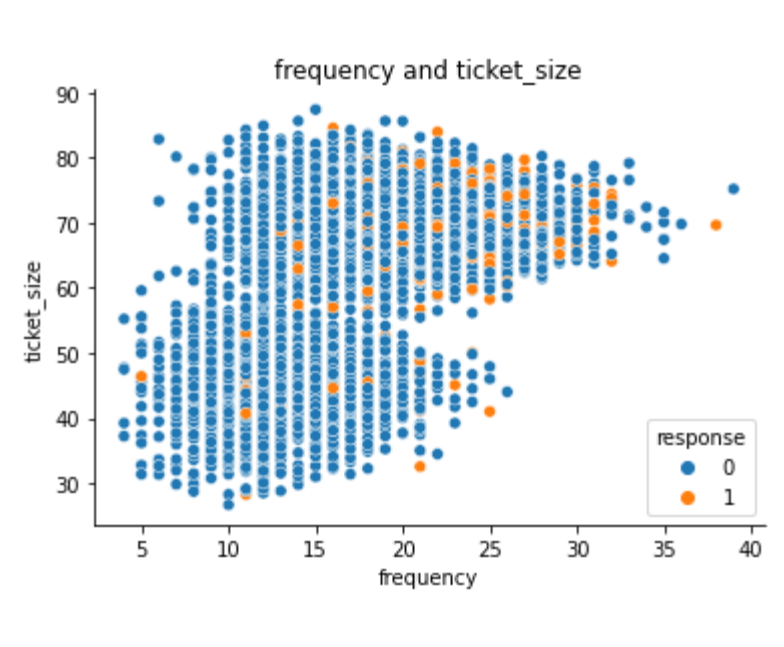


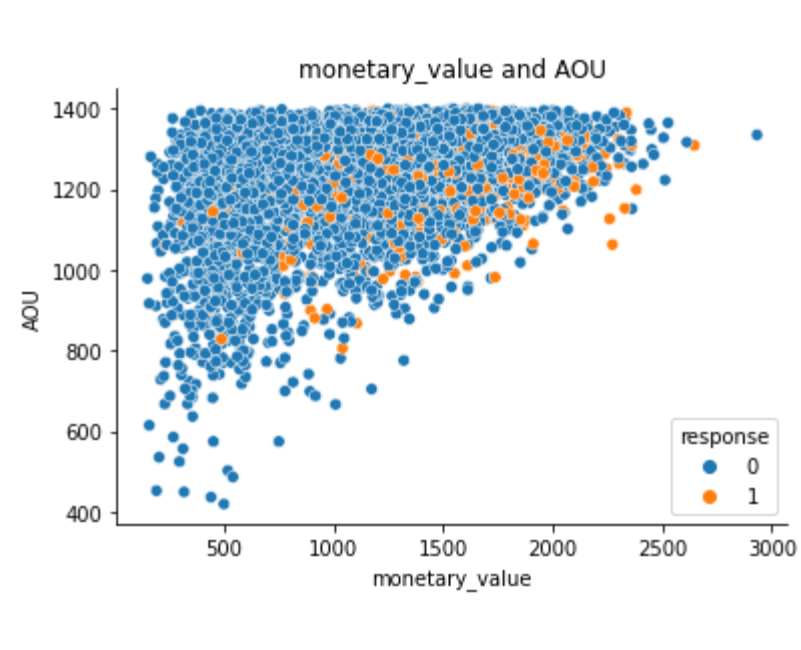


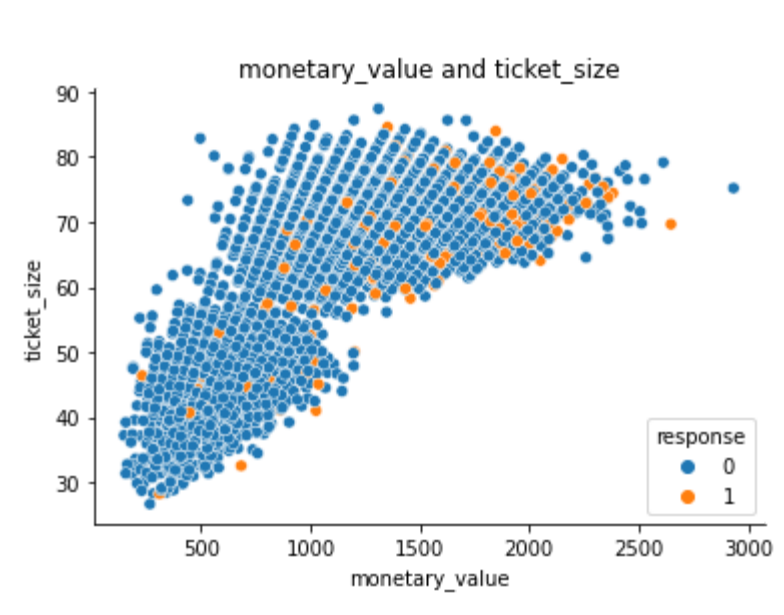


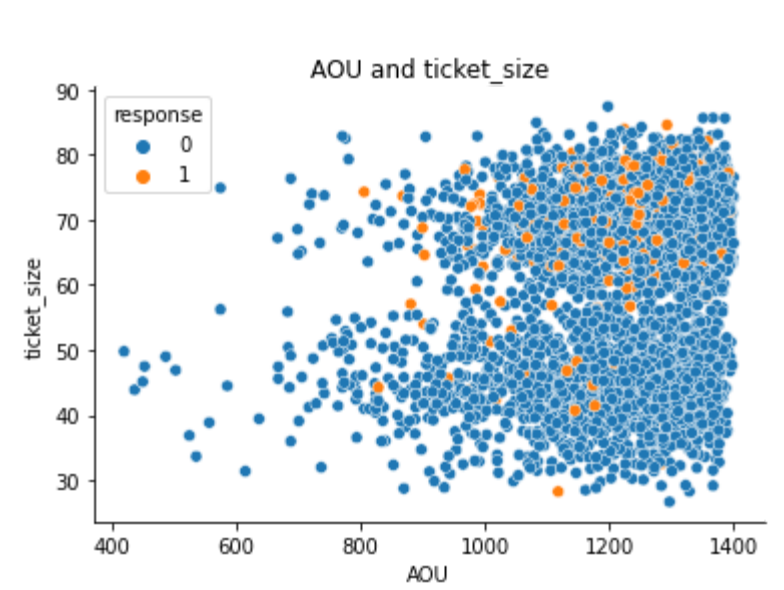












Fixing imbalanced with SMOTE

```
In [138]: sm = SMOTE(random_state=0)

sm.fit(X_train_rfm, y_train_rfm)
X_SMOTE_rfm, y_SMOTE_rfm = sm.fit_sample(X_train_rfm, y_train_rfm)

sm.fit(X_train_clv, y_train_clv)
X_SMOTE_clv, y_SMOTE_clv = sm.fit_sample(X_train_clv, y_train_clv)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprecat
ed; safe_indexing is deprecated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprecat
ed; safe_indexing is deprecated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

```
In [30]: X_SMOTE_rfm
```

```
array([[ 115.        ,   12.        ,  564.        ],
       [   8.        ,   28.        , 1976.        ],
       [ 425.        ,   19.        , 1413.        ],
       ...,
       [  27.07492082,   29.56916403, 2002.87752375],
       [  26.16799539,   17.94400154, 1254.44798772],
       [  32.49393568,   25.        , 1792.43030409]])
```

```
In [31]: y_SMOTE_rfm
```

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

```
In [80]: X_train_clv.head()
```

	recency	frequency	monetary_value	AOU	ticket_size
6590	115.0	12	564	1265	47.000000
567	8.0	28	1976	1385	70.571429
2468	425.0	19	1413	940	74.368421
1149	249.0	18	1183	1120	65.722222
2752	139.0	25	1880	1247	75.200000

```
In [90]: X_train_clv
```

	recency	frequency	monetary_value	AOU	ticket_size
6590	115.0	12	564	1265	47.000000
567	8.0	28	1976	1385	70.571429
2468	425.0	19	1413	940	74.368421
1149	249.0	18	1183	1120	65.722222
2752	139.0	25	1880	1247	75.200000
...	...	...	...	...	...
4931	31.0	25	1702	1331	68.080000
3264	152.0	20	1353	1237	67.650000
1653	13.0	24	1639	1283	68.291667
2607	7.0	18	1430	1386	79.444444
2732	95.0	22	1544	1272	70.181818

4818 rows × 5 columns

```
In [94]: X_train_clv_nor = tf.keras.utils.normalize(X_train_clv.values,axis=-1, order=2 )
X_train_clv_normalized = pd.DataFrame(data=X_train_clv_nor, columns=["recency", "frequency","monetary_va
lue","AOU","ticket_size"])
X_train_clv_normalized
```

	recency	frequency	monetary_value	AOU	ticket_size
0	0.082695	0.008629	0.405567	0.909649	0.033797
1	0.003314	0.011598	0.818472	0.573676	0.029231
2	0.242691	0.010850	0.806877	0.536776	0.042467
3	0.150964	0.010913	0.717229	0.679033	0.039846
4	0.061460	0.011054	0.831256	0.551370	0.033250
...	...	...	...	...	...
4813	0.014338	0.011563	0.787205	0.615611	0.031488
4814	0.082569	0.010864	0.734972	0.671959	0.036749
4815	0.006242	0.011523	0.786943	0.616015	0.032789
4816	0.003512	0.009031	0.717463	0.695388	0.039859
4817	0.047403	0.010978	0.770427	0.634704	0.035019

4818 rows × 5 columns

```
In [100]: X_test_clv_nor = tf.keras.utils.normalize(X_test_clv.values,axis=-1, order=2 )
X_test_clv_normalized = pd.DataFrame(data=X_test_clv_nor, columns=["recency", "frequency","monetary_valu
e","AOU","ticket_size"])
X_test_clv_normalized
```

	recency	frequency	monetary_value	AOU	ticket_size
0	0.017981	0.011559	0.506030	0.861792	0.028113
1	0.064925	0.012738	0.862518	0.500910	0.027823
2	0.060520	0.008742	0.466679	0.881580	0.035898
3	0.010592	0.010034	0.644409	0.763703	0.035801
4	0.011311	0.011311	0.786785	0.616217	0.031471
...	...	...	...	...	...
2061	0.101866	0.010502	0.786573	0.603845	0.078657
2062	0.018660	0.011107	0.809043	0.586456	0.032362
2063	0.038258	0.010340	0.744989	0.664855	0.037249
2064	0.115401	0.011480	0.760682	0.637426	0.040036
2065	0.008540	0.009760	0.656943	0.752709	0.041059

2066 rows × 5 columns

```
In [76]: x_test_clv
```

	recency	frequency	monetary_value	AOU	ticket_size
5546	28.0	18	788	1342	43.777778
3027	158.0	31	2099	1219	67.709677
6234	90.0	13	694	1311	53.384615
4308	19.0	18	1156	1370	64.222222
3035	25.0	25	1739	1362	69.560000
...	...	...	...	...	...
2266	97.0	10	749	575	74.900000
1303	42.0	25	1821	1320	72.840000
474	74.0	20	1441	1286	72.050000
1789	191.0	19	1259	1055	66.263158
3617	14.0	16	1077	1234	67.312500

2066 rows × 5 columns

```
In [77]: y_train_clv
```

```
6590    0
567     0
2468    0
1149    0
2752    0
       ..
4931    0
3264    0
1653    0
2607    0
2732    0
Name: response, Length: 4818, dtype: int64
```

```
In [78]: y_test_clv
```

```
5546    0
3027    1
6234    0
4308    0
3035    0
       ..
2266    0
1303    0
474     0
1789    0
3617    0
Name: response, Length: 2066, dtype: int64
```

```
In [ ]:
```

TF

```
In [95]: inputs = tf.keras.Input(shape=(X_train_clv_normalized.shape[1],))
x = tf.keras.layers.Dense(128, activation='relu')(inputs)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)
```

```
In [126]: model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.AUC(name='auc')
    ]
)

history = model.fit(
    X_train_clv_normalized,
    y_train_clv,
    validation_split=0.2,
    batch_size=32,
    epochs=100,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True
        )
    ]
)
```

```
Epoch 1/100
121/121 [==============================] - 1s 5ms/step - loss: 0.2768 - accuracy: 0.9102 - auc: 0.7113 - val_loss: 0.2714
- val_accuracy: 0.9139 - val_auc: 0.7227
Epoch 2/100
121/121 [==============================] - 0s 3ms/step - loss: 0.2769 - accuracy: 0.9102 - auc: 0.7106 - val_loss: 0.2707
- val_accuracy: 0.9139 - val_auc: 0.7246
Epoch 3/100
121/121 [==============================] - 0s 2ms/step - loss: 0.2760 - accuracy: 0.9102 - auc: 0.7124 - val_loss: 0.2704
- val_accuracy: 0.9139 - val_auc: 0.7273
Epoch 4/100
121/121 [==============================] - 0s 3ms/step - loss: 0.2760 - accuracy: 0.9102 - auc: 0.7129 - val_loss: 0.2699
- val_accuracy: 0.9139 - val_auc: 0.7279
Epoch 5/100
121/121 [==============================] - 0s 3ms/step - loss: 0.2763 - accuracy: 0.9102 - auc: 0.7106 - val_loss: 0.2699
- val_accuracy: 0.9139 - val_auc: 0.7292
Epoch 6/100
121/121 [==============================] - 0s 3ms/step - loss: 0.2776 - accuracy: 0.9102 - auc: 0.7040 - val_loss: 0.2702
- val_accuracy: 0.9139 - val_auc: 0.7292
Epoch 7/100
121/121 [==============================] - 0s 2ms/step - loss: 0.2772 - accuracy: 0.9102 - auc: 0.7063 - val_loss: 0.2702
- val_accuracy: 0.9139 - val_auc: 0.7289
```

```
In [127]: results = model.evaluate(X_test_clv_normalized, y_test_clv, verbose=0)

print("    Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
print("     Test AUC: {:.5f}".format(results[2]))
```

```
    Test Loss: 0.30694
Test Accuracy: 89.45%
     Test AUC: 0.72945
```

```
In [ ]:
```

```
In [139]: inputs = tf.keras.Input(shape=(X_train_clv.shape[1],))
x = tf.keras.layers.Dense(128, activation='relu')(inputs)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)
```

```
In [141]: model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.AUC(name='auc')
    ]
)

history = model.fit(
    X_train_clv,
    y_train_clv,
    validation_split=0.2,
    batch_size=32,
    epochs=100,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True
        )
    ]
)
```

```
Epoch 1/100
121/121 [==============================] - 1s 5ms/step - loss: 1.2110 - accuracy: 0.8511 - auc: 0.5459 - val_loss: 1.0285
- val_accuracy: 0.9139 - val_auc: 0.5000
Epoch 2/100
121/121 [==============================] - 0s 3ms/step - loss: 1.1635 - accuracy: 0.8516 - auc: 0.5388 - val_loss: 0.6774
- val_accuracy: 0.9139 - val_auc: 0.5029
Epoch 3/100
121/121 [==============================] - 0s 2ms/step - loss: 1.0992 - accuracy: 0.8524 - auc: 0.5522 - val_loss: 0.5700
- val_accuracy: 0.9139 - val_auc: 0.5248
Epoch 4/100
121/121 [==============================] - 0s 2ms/step - loss: 0.5887 - accuracy: 0.8648 - auc: 0.5696 - val_loss: 0.5648
- val_accuracy: 0.9139 - val_auc: 0.5226
Epoch 5/100
121/121 [==============================] - 0s 3ms/step - loss: 0.6568 - accuracy: 0.8640 - auc: 0.6081 - val_loss: 0.7252
- val_accuracy: 0.9139 - val_auc: 0.5266
Epoch 6/100
121/121 [==============================] - 0s 3ms/step - loss: 0.7966 - accuracy: 0.8594 - auc: 0.5354 - val_loss: 2.8885
- val_accuracy: 0.3641 - val_auc: 0.7136
Epoch 7/100
121/121 [==============================] - 0s 3ms/step - loss: 0.9385 - accuracy: 0.8485 - auc: 0.5482 - val_loss: 0.5152
- val_accuracy: 0.9129 - val_auc: 0.5727
Epoch 8/100
121/121 [==============================] - 0s 3ms/step - loss: 0.5225 - accuracy: 0.8718 - auc: 0.6038 - val_loss: 0.3418
- val_accuracy: 0.9129 - val_auc: 0.7451
Epoch 9/100
121/121 [==============================] - 0s 3ms/step - loss: 0.6058 - accuracy: 0.8646 - auc: 0.5722 - val_loss: 0.6780
- val_accuracy: 0.6556 - val_auc: 0.7394
Epoch 10/100
121/121 [==============================] - 0s 2ms/step - loss: 0.6167 - accuracy: 0.8573 - auc: 0.5805 - val_loss: 0.3045
- val_accuracy: 0.9077 - val_auc: 0.7194
Epoch 11/100
121/121 [==============================] - 0s 3ms/step - loss: 1.0240 - accuracy: 0.8630 - auc: 0.5783 - val_loss: 2.2940
- val_accuracy: 0.9139 - val_auc: 0.5000
Epoch 12/100
121/121 [==============================] - 0s 2ms/step - loss: 1.0538 - accuracy: 0.8591 - auc: 0.5547 - val_loss: 0.6086
- val_accuracy: 0.9139 - val_auc: 0.5371
Epoch 13/100
121/121 [==============================] - 0s 2ms/step - loss: 0.4942 - accuracy: 0.8682 - auc: 0.6165 - val_loss: 0.6391
- val_accuracy: 0.9139 - val_auc: 0.4977
```

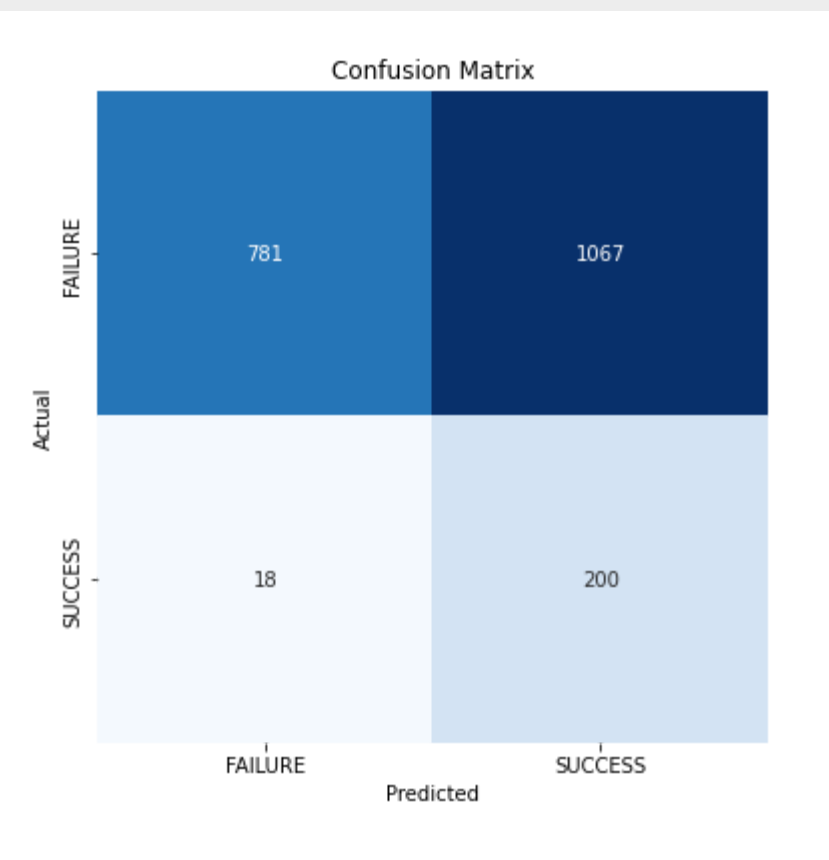
```
In [69]: results = model.evaluate(X_test_clv, y_test_clv, verbose=0)

print("    Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
print("     Test AUC: {:.5f}".format(results[2]))
```

```
    Test Loss: 0.29974
Test Accuracy: 89.45%
     Test AUC: 0.75225
```

```
In [142]: results = model.evaluate(X_train_clv, y_train_clv, verbose=0)

print("    Train Loss: {:.5f}".format(results[0]))
print("Train Accuracy: {:.2f}%".format(results[1] * 100))
print("     Train AUC: {:.5f}".format(results[2]))
```

```
    Train Loss: 0.28952
Train Accuracy: 90.45%
     Train AUC: 0.72425
```

```
In [74]: y_pred = np.array(model.predict(X_test_clv) >= 0.05, dtype=np.int)

cm = confusion_matrix(y_test_clv, y_pred)
clr = classification_report(y_test_clv, y_pred, target_names=["FAILURE", "SUCCESS"])

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
plt.xticks(ticks=np.arange(2) + 0.5, labels=["FAILURE", "SUCCESS"])
plt.yticks(ticks=np.arange(2) + 0.5, labels=["FAILURE", "SUCCESS"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

print("Classification Report:\n----------------------\n", clr)
```




```
Classification Report:
----------------------
              precision    recall  f1-score   support

     FAILURE       0.98      0.42      0.59      1848
     SUCCESS       0.16      0.92      0.27       218

    accuracy                           0.47      2066
   macro avg       0.57      0.67      0.43      2066
weighted avg       0.89      0.47      0.56      2066
```

```
In [63]:
```

Logistic Regression

```
In [21]: print('logistic regression model - SMOTE RFM')
logreg = LogisticRegression(solver='liblinear', class_weight='balanced')
predicted_y = []
expected_y = []

logreg_model_SMOTE_rfm = logreg.fit(X_SMOTE_rfm, y_SMOTE_rfm)
predictions = logreg_model_SMOTE_rfm.predict(X_SMOTE_rfm)
predicted_y.extend(predictions)
expected_y.extend(y_SMOTE_rfm)
report_train = classification_report(expected_y, predicted_y)
print('training set')
print(report_train)

predicted_y = []
expected_y = []
predictions = logreg_model_SMOTE_rfm.predict(X_test_rfm)
predicted_y.extend(predictions)
expected_y.extend(y_test_clv)
report_test = classification_report(expected_y, predicted_y)
print('test set')
print(report_test)
```

```
logistic regression model - SMOTE RFM
training set
              precision    recall  f1-score   support

           0       0.68      0.62      0.65      4389
           1       0.65      0.71      0.68      4389

    accuracy                           0.67      8778
   macro avg       0.67      0.67      0.67      8778
weighted avg       0.67      0.67      0.67      8778

test set
              precision    recall  f1-score   support

           0       0.95      0.63      0.75      1848
           1       0.18      0.71      0.29       218

    accuracy                           0.64      2066
   macro avg       0.57      0.67      0.52      2066
weighted avg       0.87      0.64      0.71      2066
```

```
In [22]: print('logistic regression model - SMOTE CLV')
logreg = LogisticRegression(solver='liblinear', class_weight='balanced')
predicted_y = []
expected_y = []

logreg_model_SMOTE_clv = logreg.fit(X_SMOTE_clv, y_SMOTE_clv)
predictions = logreg_model_SMOTE_clv.predict(X_SMOTE_clv)
predicted_y.extend(predictions)
expected_y.extend(y_SMOTE_clv)
report_train = classification_report(expected_y, predicted_y)
print('training set')
print(report_train)

predicted_y = []
expected_y = []
predictions = logreg_model_SMOTE_clv.predict(X_test_clv)
predicted_y.extend(predictions)
expected_y.extend(y_test_clv)
report_test = classification_report(expected_y, predicted_y)
print('test set')
print(report_test)
```

```
logistic regression model - SMOTE CLV
training set
              precision    recall  f1-score   support

           0       0.68      0.62      0.65      4389
           1       0.65      0.71      0.68      4389

    accuracy                           0.67      8778
   macro avg       0.67      0.67      0.67      8778
weighted avg       0.67      0.67      0.67      8778

test set
              precision    recall  f1-score   support

           0       0.95      0.62      0.75      1848
           1       0.18      0.72      0.29       218

    accuracy                           0.63      2066
   macro avg       0.57      0.67      0.52      2066
weighted avg       0.87      0.63      0.70      2066
```

XGBoost

```
In [23]: print('XGBoost model - SMOTE RFM')

xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='auc',
 learning_rate =0.01,
 n_estimators=100,
 max_depth=2,
 gamma=0.0,
 colsample_bytree=0.6)

predicted_y = []
expected_y = []

xgb_model_SMOTE_rfm = xgb_model.fit(X_SMOTE_rfm, y_SMOTE_rfm, early_stopping_rounds=5, eval_set=[(X_test
_rfm.to_numpy(), y_test_rfm)])
predictions =  xgb_model_SMOTE_rfm.predict(X_SMOTE_rfm)
predicted_y.extend(predictions)
expected_y.extend(y_SMOTE_rfm)
report_train = classification_report(expected_y, predicted_y)
print('training set')
print(report_train)

predicted_y = []
expected_y = []
predictions = xgb_model_SMOTE_rfm.predict(X_test_rfm.to_numpy())
predicted_y.extend(predictions)
expected_y.extend(y_test_rfm)
report_test = classification_report(expected_y, predicted_y)
print('test set')
print(report_test)
```

```
XGBoost model - SMOTE RFM
[0]	validation_0-auc:0.567615
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]	validation_0-auc:0.713189
[2]	validation_0-auc:0.709773
[3]	validation_0-auc:0.69828
[4]	validation_0-auc:0.711449
[5]	validation_0-auc:0.713189
[6]	validation_0-auc:0.708296
Stopping. Best iteration:
[1]	validation_0-auc:0.713189

training set
              precision    recall  f1-score   support

           0       0.75      0.57      0.65      4389
           1       0.65      0.81      0.72      4389

    accuracy                           0.69      8778
   macro avg       0.70      0.69      0.69      8778
weighted avg       0.70      0.69      0.69      8778

test set
              precision    recall  f1-score   support

           0       0.96      0.57      0.71      1848
           1       0.18      0.79      0.29       218

    accuracy                           0.59      2066
   macro avg       0.57      0.68      0.50      2066
weighted avg       0.88      0.59      0.67      2066
```

```
In [24]: print('XGBoost model - SMOTE CLV')

xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='auc',
 learning_rate =0.01,
 n_estimators=100,
 max_depth=2,
 gamma=0.0,
 colsample_bytree=0.6)

predicted_y = []
expected_y = []

xgb_model_SMOTE_clv = xgb_model.fit(X_SMOTE_clv, y_SMOTE_clv, early_stopping_rounds=5, eval_set=[(X_test
_clv.to_numpy(), y_test_clv)])
predictions =  xgb_model_SMOTE_clv.predict(X_SMOTE_clv)
predicted_y.extend(predictions)
expected_y.extend(y_SMOTE_clv)
report_train = classification_report(expected_y, predicted_y)
print('training set')
print(report_train)

predicted_y = []
expected_y = []
predictions = xgb_model_SMOTE_clv.predict(X_test_clv.to_numpy())
predicted_y.extend(predictions)
expected_y.extend(y_test_clv)
report_test = classification_report(expected_y, predicted_y)
print('test set')
print(report_test)
```

```
XGBoost model - SMOTE CLV
[0]	validation_0-auc:0.68482
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]	validation_0-auc:0.717808
[2]	validation_0-auc:0.724408
[3]	validation_0-auc:0.725875
[4]	validation_0-auc:0.723803
[5]	validation_0-auc:0.726727
[6]	validation_0-auc:0.72539
[7]	validation_0-auc:0.724427
[8]	validation_0-auc:0.723888
[9]	validation_0-auc:0.724839
[10]	validation_0-auc:0.724921
Stopping. Best iteration:
[5]	validation_0-auc:0.726727

training set
              precision    recall  f1-score   support

           0       0.77      0.55      0.64      4389
           1       0.65      0.84      0.73      4389

    accuracy                           0.70      8778
   macro avg       0.71      0.70      0.69      8778
weighted avg       0.71      0.70      0.69      8778

test set
              precision    recall  f1-score   support

           0       0.96      0.55      0.70      1848
           1       0.17      0.80      0.28       218

    accuracy                           0.58      2066
   macro avg       0.57      0.67      0.49      2066
weighted avg       0.88      0.58      0.66      2066
```

```
In [25]: ## building pipeline for hyperparameter tuning

from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2

# Create a pipeline
pipe = Pipeline([
  ('fs', SelectKBest()),
  ('clf', xgb.XGBClassifier(objective='binary:logistic', scale_pos_weight=9))
])
```

```
In [26]: ## hyper parameter tuning - grid search

from sklearn.model_selection import KFold, GridSearchCV
from sklearn.metrics import accuracy_score, make_scorer
# Define our search space for grid search
search_space = [
  {
    'clf__n_estimators': [100, 300],
    'clf__learning_rate': [0.01, 0.1],
    'clf__max_depth': range(2, 5),
    'clf__colsample_bytree': [i/10.0 for i in range(4, 7)],
    'clf__gamma': [i/10.0 for i in range(3)],
    'fs__score_func': [chi2],
    'fs__k': [2],
  }
]
# Define cross validation
kfold = KFold(n_splits=5, random_state=42)
# AUC and F1 as score
scoring = {'AUC':'roc_auc','F1 score': 'f1_micro'}
# Define grid search
grid = GridSearchCV(
  pipe,
  param_grid=search_space,
  cv=kfold,
  scoring=scoring,
  refit='AUC',
  verbose=1,
  n_jobs=-1
)

# Fit grid search
xgb_model_clv_GS = grid.fit(X_train_clv, y_train_clv)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state has n
o effect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), o
r set shuffle=True.
  FutureWarning
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:    9.8s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:   37.9s
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 540 out of 540 | elapsed:  1.7min finished
```

```
In [27]: predicted_y = []
expected_y = []
predictions = xgb_model_clv_GS.predict(X_test_clv)
print('Best AUC Score: {}'.format(xgb_model_clv_GS.best_score_))
print(confusion_matrix(y_test_clv,predictions))


predicted_y.extend(predictions)
expected_y.extend(y_test_clv)
report_test = classification_report(expected_y, predicted_y)
print('test set')
print(report_test)
```

```
Best AUC Score: 0.7089054675375633
[[1089  759]
 [  51  167]]
test set
              precision    recall  f1-score   support

           0       0.96      0.59      0.73      1848
           1       0.18      0.77      0.29       218

    accuracy                           0.61      2066
   macro avg       0.57      0.68      0.51      2066
weighted avg       0.87      0.61      0.68      2066
```

```
In [28]: print(xgb_model_clv_GS.best_params_)
```

```
{'clf__colsample_bytree': 0.4, 'clf__gamma': 0.0, 'clf__learning_rate': 0.01, 'clf__max_depth': 2, 'clf__n_estimators': 1
00, 'fs__k': 2, 'fs__score_func': <function chi2 at 0x7fdd1c696a70>}
```

```
In [28]:
```