Importing libraries and datasets

In [10]:
```
from google.colab import drive
drive.mount("")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-5249ad3b76d4> in <module>()
      1 from google.colab import drive
----> 2 drive.mount("/content/sample_data")

/usr/local/lib/python3.7/dist-packages/google/colab/drive.py in mount(mountpoint, force_remoun
t, timeout_ms, use_metadata_server)
    111       timeout_ms=timeout_ms,
    112       use_metadata_server=use_metadata_server,
--> 113       ephemeral=ephemeral)
    114
    115

/usr/local/lib/python3.7/dist-packages/google/colab/drive.py in _mount(mountpoint, force_remoun
t, timeout_ms, use_metadata_server, ephemeral)
    192       raise ValueError('Mountpoint must not be a symlink')
    193     if _os.path.isdir(mountpoint) and _os.listdir(mountpoint):
--> 194       raise ValueError('Mountpoint must not already contain files')
    195     if not _os.path.isdir(mountpoint) and _os.path.exists(mountpoint):
    196       raise ValueError('Mountpoint must either be a directory or not exist')

ValueError: Mountpoint must not already contain files
```

In [71]:
```
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

from sklearn.metrics import precision_score, recall_score, f1_score, auc, roc_auc_score, accura
from xgboost import plot_importance
```

In [174…
```
df_response = pd.read_csv('/content/sample_data/Retail_Data_Response.csv')
df_transactions = pd.read_csv('/content/sample_data/Retail_Data_Transactions.csv', parse_dates=
```

In [175…
```
df_response.head()
```

Out[175…

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

In [176…
```
df_transactions.head()
```

Out[176…

	customer_id	trans_date	tran_amount

	customer_id	trans_date	tran_amount
0	CS5295	2013-02-11	35.0
1	CS4768	2015-03-15	39.0
2	CS2122	2013-02-26	52.0
3	CS1217	2011-11-16	99.0
4	CS1850	2013-11-20	78.0

In [177…
```
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 [178…
```
## since the last date of the data is 16 March 2015, the campaign date is assumed to be 17 Marc
## 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()
```

Out[178…

	customer_id	trans_date	tran_amount	recent
0	CS5295	2013-02-11	35.0	764.0
1	CS4768	2015-03-15	39.0	2.0
2	CS2122	2013-02-26	52.0	749.0
3	CS1217	2011-11-16	99.0	1217.0
4	CS1850	2013-11-20	78.0	482.0

In [179…
```
## 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()})


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

In [180…
```
df_rfm = df_rfm.reset_index()
df_rfm.head()
```

Out[180…

	customer_id	recency	frequency	monetary_value
0	CS1112	62.0	15	1012.0
1	CS1113	36.0	20	1490.0
2	CS1114	33.0	19	1432.0
3	CS1115	12.0	22	1659.0
4	CS1116	204.0	13	857.0

In [181…
```
## create data set with CLV variables
```

```
df_clv = df_transactions.groupby('customer_id').agg({'recent': lambda x:x.min(),
                                                     'customer_id': lambda x: len(x),
                                                     'tran_amount': lambda x: x.sum(),
                                                     'trans_date': lambda x: (x.max() - x.min()

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 [182…

```
df_clv = df_clv.reset_index()
df_clv.head()
```

Out[182…

	customer_id	recency	frequency	monetary_value	AOU	ticket_size
0	CS1112	62.0	15	1012.0	1309	67.466667
1	CS1113	36.0	20	1490.0	1354	74.500000
2	CS1114	33.0	19	1432.0	1309	75.368421
3	CS1115	12.0	22	1659.0	1303	75.409091
4	CS1116	204.0	13	857.0	1155	65.923077

Calculating response rate

In [183…

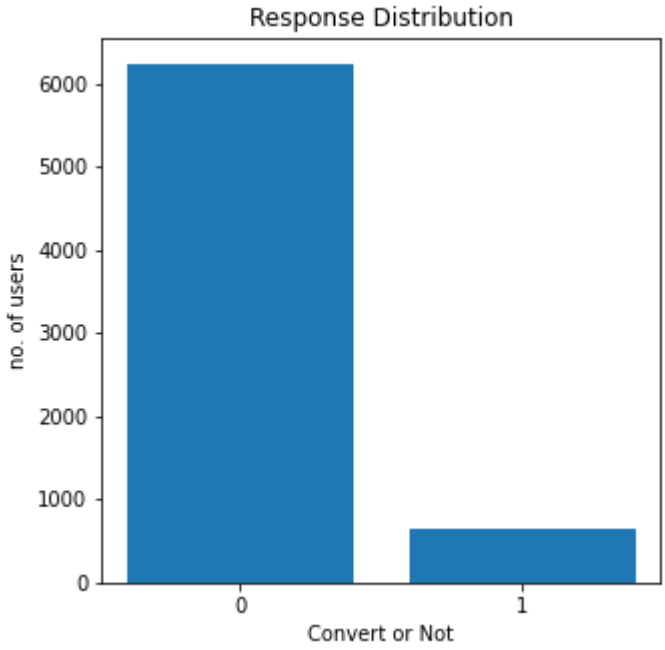
```
response_rate = df_response.groupby('response').agg({'customer_id': lambda x: len(x)}).reset_ir
response_rate.head()
```

Out[183…

	response	customer_id
0	0	6237
1	1	647

In [184…

```
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 [185…
```
## merging two data sets - RFM

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

Out[185…

	customer_id	response	recency	frequency	monetary_value
0	CS1112	0	62.0	15	1012.0
1	CS1113	0	36.0	20	1490.0
2	CS1114	1	33.0	19	1432.0
3	CS1115	1	12.0	22	1659.0
4	CS1116	1	204.0	13	857.0

In [186…
```
## merging two data sets - CLV

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

Out[186…

	customer_id	response	recency	frequency	monetary_value	AOU	ticket_size
0	CS1112	0	62.0	15	1012.0	1309	67.466667
1	CS1113	0	36.0	20	1490.0	1354	74.500000
2	CS1114	1	33.0	19	1432.0	1309	75.368421
3	CS1115	1	12.0	22	1659.0	1303	75.409091
4	CS1116	1	204.0	13	857.0	1155	65.923077

Creating train and test dataset

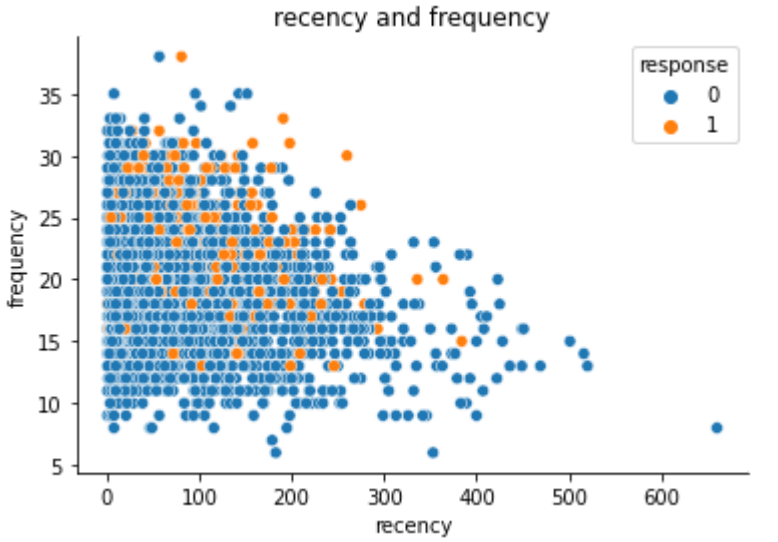
In [187…
```
## spliting dataframe into X and y

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']
```

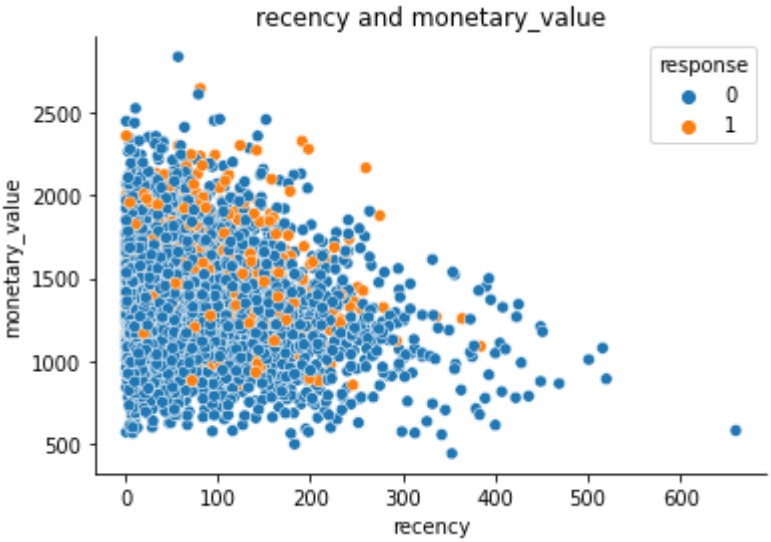
In [202…
```
## creating train and test dataset
```

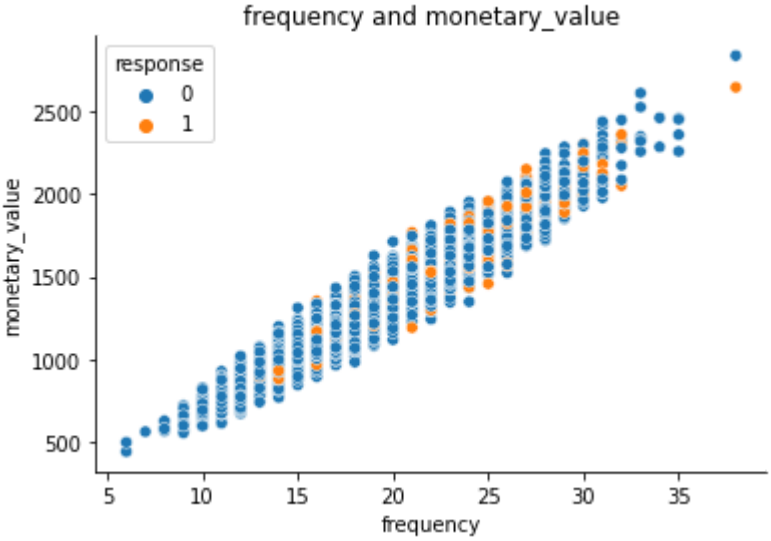
```
X_train_rfm, X_test_rfm, y_train_rfm, y_test_rfm = train_test_split(X_rfm, y_rfm, test_size=0.0
X_train_clv, X_test_clv, y_train_clv, y_test_clv = train_test_split(X_clv, y_clv, test_size=0.0
```

In [203…

```
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()
```



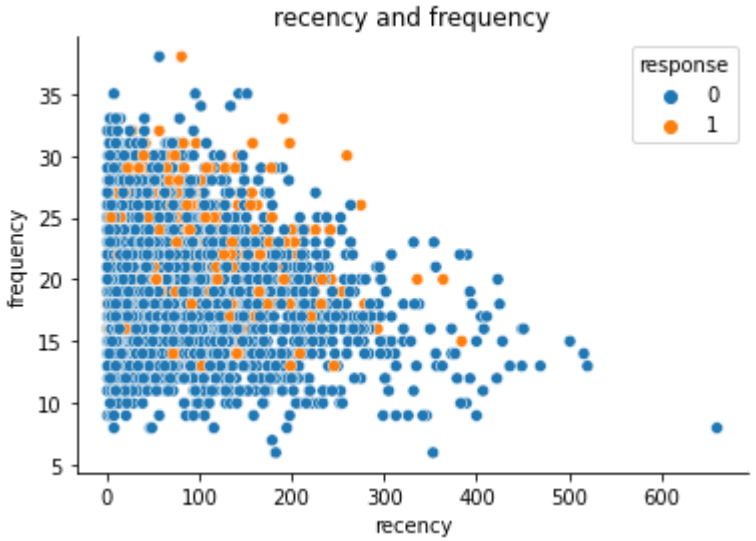


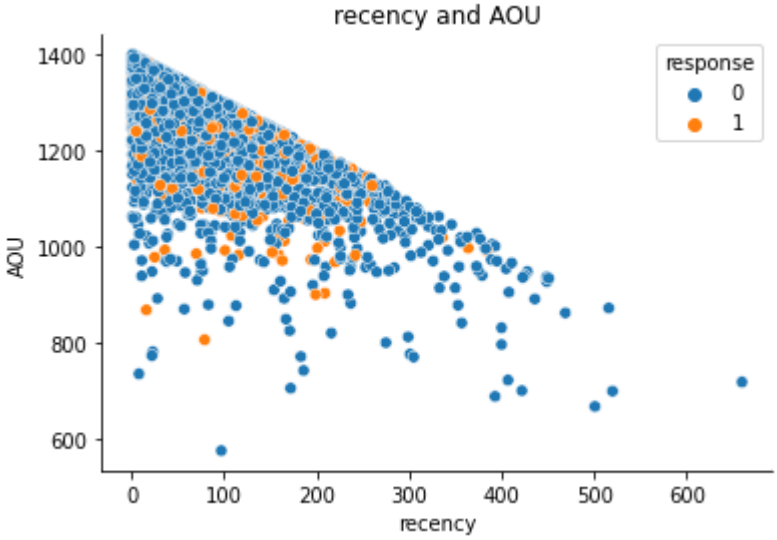


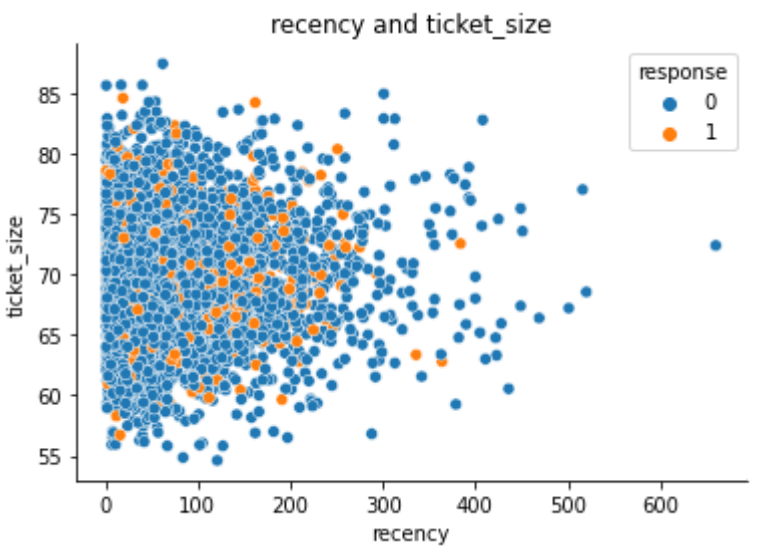
In [204…

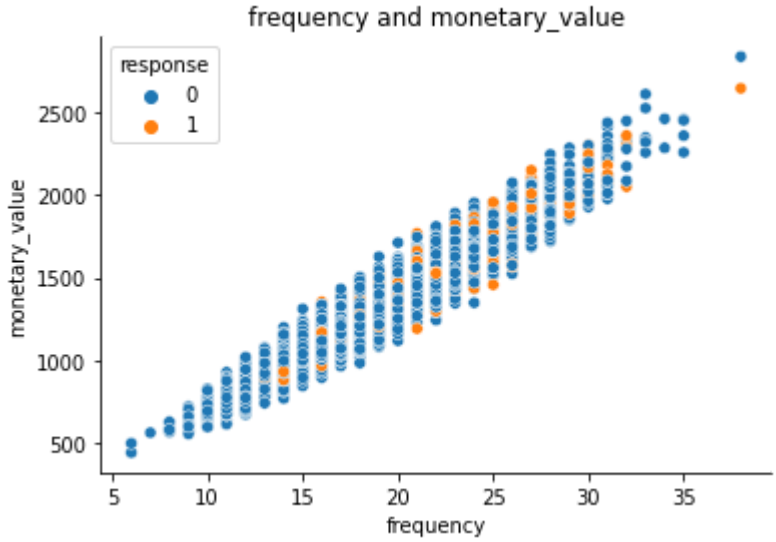
```
for i, col_i in enumerate(df_modeling_clv[['recency', 'frequency', 'monetary_value', 'AOU', 'ti
  for j, col_j in enumerate(df_modeling_clv[['recency', 'frequency', 'monetary_value', 'AOU',
    if i < j :
      plt.title(col_i + ' and ' + col_j)
```

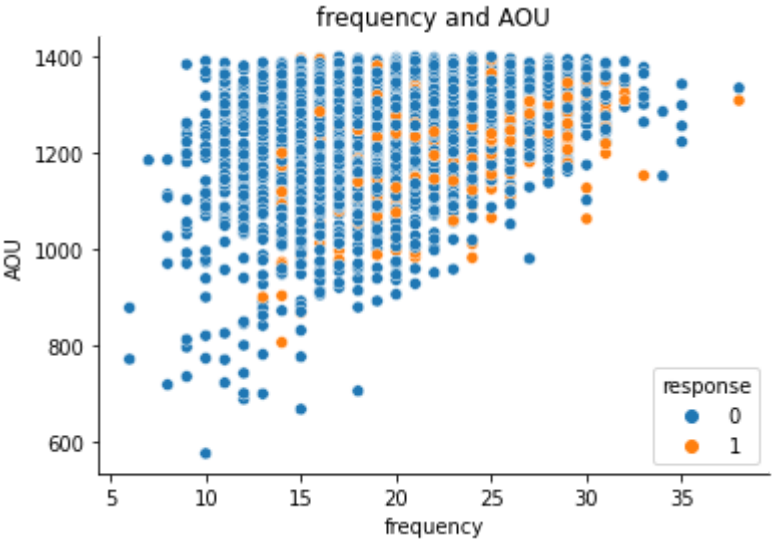
```
        sns.scatterplot(data=df_modeling_clv, x=col_i, y=col_j, hue='response')
        sns.despine()
        plt.show()
```

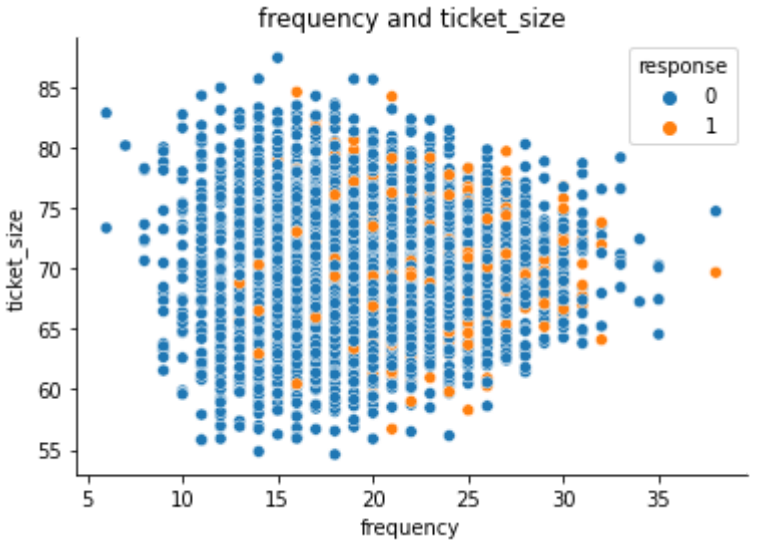


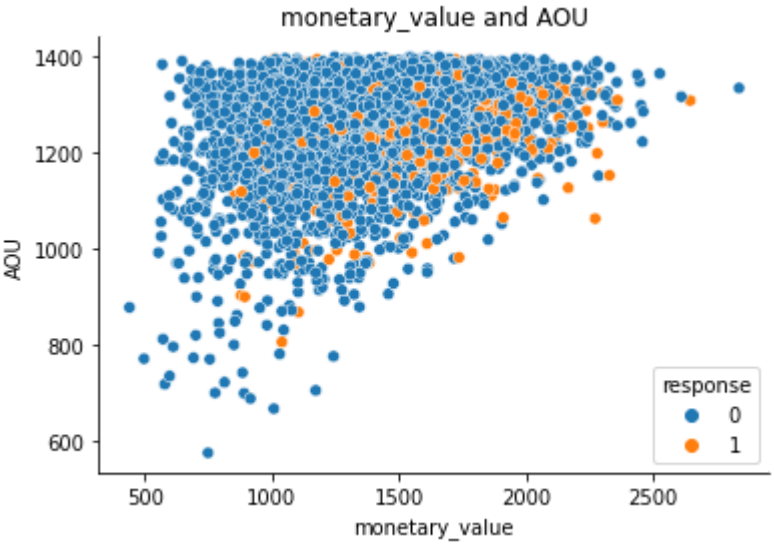


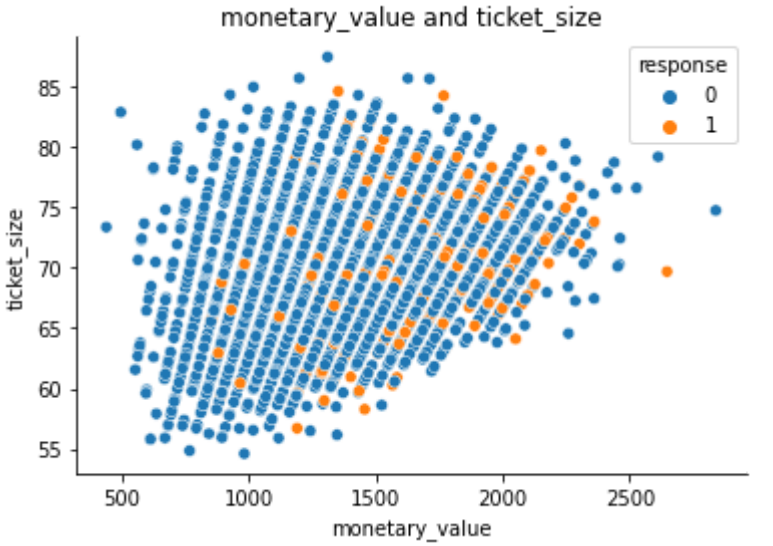

recency and ticket_size
response
0
1
ticket_size
recency
85
80
75
70
65
60
55
0
100
200
300
400
500
600

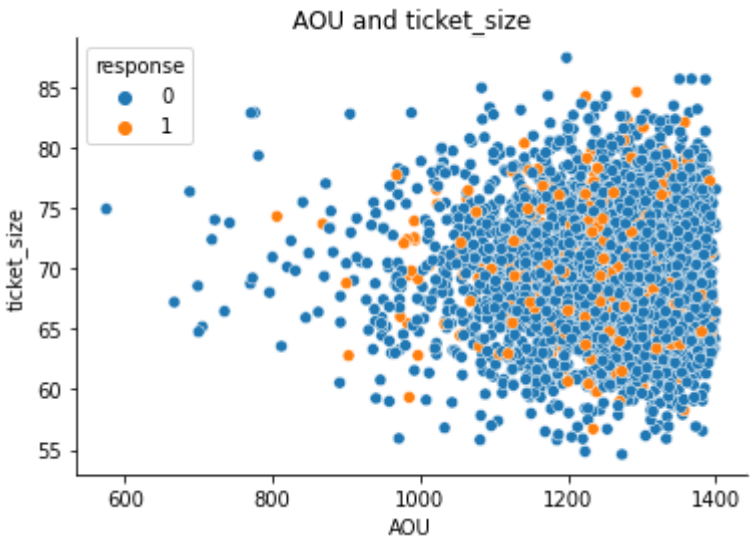

frequency and monetary_value
response
0
1
monetary_value
frequency
2500
2000
1500
1000
500
5
10
15
20
25
30
35


frequency and AOU
response
0
1
AOU
frequency
1400
1200
1000
800
600
5
10
15
20
25
30
35


frequency and ticket_size
response
0
1
ticket_size
frequency
85
80
75
70
65
60
55
5
10
15
20
25
30
35


monetary_value and AOU
response
0
1
AOU
monetary_value
1400
1200
1000
800
600
500
1000
1500
2000
2500


monetary_value and ticket_size
response
0
1
ticket_size
monetary_value
85
80
75
70
65
60
55
500
1000
1500
2000
2500



Fixing imbalanced with SMOTE

In [205…

```
sm = SMOTE(random_state = 101)

# from imblearn.over_sampling import SMOTENC
# sm = SMOTENC([1],random_state=101)

# from imblearn.over_sampling import SVMSMOTE
# svmsmote = SVMSMOTE(random_state = 2002)
# sm = svmsmote

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 deprecated; 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 deprecated; safe_indexing is deprecated in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

In [206…

```
# # Scaler
# from sklearn.preprocessing import StandardScaler
# from sklearn.preprocessing import MinMaxScaler

# transfer = MinMaxScaler()
# X_SMOTE_rfm = transfer.fit_transform(X_SMOTE_rfm)
# X_SMOTE_clv = transfer.fit_transform(X_SMOTE_clv)
# print(X_SMOTE_rfm)
# print(X_SMOTE_clv)
# print(y_SMOTE_rfm)
# print(y_SMOTE_clv)
```

Logistic Regression

In [207…

```
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       0.60      0.62      0.61      4175
           1       0.61      0.59      0.60      4175

    accuracy                           0.61      8350
   macro avg       0.61      0.61      0.61      8350
weighted avg       0.61      0.61      0.61      8350

test set
              precision    recall  f1-score   support

           0       0.96      0.59      0.73       228
           1       0.15      0.73      0.24        22

    accuracy                           0.60       250
   macro avg       0.55      0.66      0.49       250
weighted avg       0.89      0.60      0.69       250
```

In [208…

```
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       0.60      0.62      0.61      4175
           1       0.61      0.58      0.60      4175

    accuracy                           0.60      8350
   macro avg       0.60      0.60      0.60      8350
weighted avg       0.60      0.60      0.60      8350

test set
              precision    recall  f1-score   support
```

```
           0       0.95      0.57      0.72       228
           1       0.13      0.68      0.22        22

    accuracy                           0.58       250
   macro avg       0.54      0.63      0.47       250
weighted avg       0.88      0.58      0.67       250
```

XGBoost

In [242…

```
print('XGBoost model - SMOTE RFM')

xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='auc',
 learning_rate =0.01,
 n_estimators=100,
 max_depth=4,
 gamma=0.0,
 colsample_bytree=0.6)

predicted_y = []
expected_y = []

print(X_SMOTE_rfm)
print(y_SMOTE_rfm)

xgb_model_SMOTE_rfm = xgb_model.fit(X_SMOTE_rfm, y_SMOTE_rfm, early_stopping_rounds=5, eval_set
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)

print('test AUC : ' )
roc_auc_score(expected_y, predicted_y, average=None)
```

```
XGBoost model - SMOTE RFM
[[  45.           11.          834.        ]
 [ 274.           20.         1474.        ]
 [  80.           16.         1007.        ]
 ...
 [  27.82123284   25.29952583 1657.43962067]
 [  63.13164079   24.24911456 1858.25265632]
 [  20.1678666    21.48046274 1516.48046274]]
[0 0 0 ... 1 1 1]
[0]     validation_0-auc:0.683612
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]     validation_0-auc:0.737241
[2]     validation_0-auc:0.729067
[3]     validation_0-auc:0.727273
[4]     validation_0-auc:0.733852
[5]     validation_0-auc:0.737241
[6]     validation_0-auc:0.73126
Stopping. Best iteration:
[1]     validation_0-auc:0.737241

training set
              precision    recall  f1-score   support

           0       0.65      0.62      0.63      4175
           1       0.64      0.67      0.65      4175
```

```
    accuracy                           0.64      8350
   macro avg       0.64      0.64      0.64      8350
weighted avg       0.64      0.64      0.64      8350

test set
              precision    recall  f1-score   support

           0       0.95      0.65      0.77       228
           1       0.16      0.68      0.26        22

    accuracy                           0.65       250
   macro avg       0.56      0.67      0.51       250
weighted avg       0.88      0.65      0.73       250

test AUC :
```

Out[242…

```
0.6654704944178628
```

In [289…

```
print('XGBoost model - SMOTE CLV')

xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='auc',
 learning_rate =0.5,
 n_estimators=5000,
 max_depth=3,
 gamma=0.0,
 colsample_bytree=0.1,
 sampling_method = 'uniform',
 max_delta_step = 10)

predicted_y = []
expected_y = []

xgb_model_SMOTE_clv = xgb_model.fit(X_SMOTE_clv, y_SMOTE_clv, early_stopping_rounds=5, eval_set
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)
print('test AUC : ' )
roc_auc_score(expected_y, predicted_y, average=None)
```

```
XGBoost model - SMOTE CLV
[0]     validation_0-auc:0.555423
Will train until validation_0-auc hasn't improved in 5 rounds.
[1]     validation_0-auc:0.674541
[2]     validation_0-auc:0.754984
[3]     validation_0-auc:0.770036
[4]     validation_0-auc:0.766647
[5]     validation_0-auc:0.763457
[6]     validation_0-auc:0.731061
[7]     validation_0-auc:0.711822
[8]     validation_0-auc:0.723983
Stopping. Best iteration:
[3]     validation_0-auc:0.770036

training set
              precision    recall  f1-score   support

           0       0.64      0.68      0.66      4175
           1       0.66      0.62      0.64      4175
```

```
    accuracy                           0.65      8350
   macro avg       0.65      0.65      0.65      8350
weighted avg       0.65      0.65      0.65      8350

test set
              precision    recall  f1-score   support

           0       0.97      0.68      0.80       228
           1       0.19      0.77      0.31        22

    accuracy                           0.69       250
   macro avg       0.58      0.73      0.55       250
weighted avg       0.90      0.69      0.76       250

test AUC :
```

Out[289… 0.7284688995215312

In [212…
```
# ## 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 [286…
```
# ## 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__colsample_bytree': [0.1,0.3,0.5,0.7,0.9],
#     '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)
# # xgb_model_clv_GS = grid.fit(X_train_rfm, y_train_rfm)
```

In [287…
```
# 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)
```

In [232…

In [288…

```
# print(xgb_model_clv_GS.best_params_)
```

In [37]: