Campaign response model

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

```
min 2013-03-17 00:00:00
max 2015-03-16 00:00:00
```

Data cover 3 year period

11 features are used

- Recency – days from last purchase
- Frequency_2y – number of transaction over the last 2 years
- monetary_value_2y – amount purchased over the last 2 years
- median_purchase_2y – median purchase over the last 2 years
- std_amount_2y - standard deviation of amount purchase over the last 2 years
- frequency_1y – frequency of purchase over 1 year
- Freq_Q2 – portion of frequency in Q2 over the last 2 years
- Freq_Q3 – portion of frequency in Q3 over the last 2 years
- Freq_Q4 - portion of frequency in Q4 over the last 2 years
- last_purchase – last purchase amount
- days_between_trans - average days between transactions

Data preparation & feature engineering

```
#Time between purchase (days)
df_trans2 = df_transactions.copy()
df_trans2.sort_values(['customer_id','trans_date'],inplace=True)
df_trans2['previous_trans_date'] = df_trans2.groupby(['customer_id'])['trans_date'].shift(1)
df_trans2= df_trans2[~(df_trans2['previous_trans_date'].isna())]
df_trans2['days_between_trans'] = df_trans2['trans_date']-df_trans2['previous_trans_date']
df_trans2['days_between_trans'].astype('timedelta64[D]')
df_trans2['days_between_trans']=df_trans2['days_between_trans'] / np.timedelta64(1, 'D')
df_tbp = df_trans2.groupby(['customer_id']).agg({'days_between_trans': lambda x: x.mean()}).reset_index()
df_tbp.rename(columns={'days_between_trans':'days_between_trans'},inplace=True)
```

```
df_trans3 = df_transactions.copy()
df_trans3.sort_values(['customer_id','trans_date'],inplace=True)
df_trans3['rank']=df_trans3.groupby('customer_id')['trans_date'].rank(method='dense',ascending=False)
df_trans3 = df_trans3[df_trans3['rank']==1]
df_trans3.rename(columns={'tran_amount':'last_purchase'},inplace=True)
df_trans3 = df_trans3[['customer_id','last_purchase']]
df_trans3 = df_trans3.groupby('customer_id').sum().reset_index()
```

```
#Trans Quarter
df_transq = df_transactions.copy()
df_transq['Quarter'] =  df_transq['trans_date'].dt.quarter
df_transqt = df_transq.groupby(['customer_id','Quarter']).agg({'tran_amount':'sum','recent':'count'}).reset_index()
df_transqt.rename(columns={'recent':'Freq'},inplace=True)
```

```
df_transqt=df_transqt.pivot(index='customer_id',columns='Quarter',values=['Freq'])
df_transqt.index.name = 'customer_id'
df_transqt.columns = ['%s%s' % (a, '_Q%s' % b if b else '') for a, b in df_transqt.columns]
df_transqt.reset_index(inplace=True)
```

```
df_transqt.drop(['Freq_Q1'],axis=1,inplace=True)
df_transqt.fillna(0,inplace=True)
```

Data preparation & feature engineering

```
## create data set with RFM variables
df_transactions = df_transactions[df_transactions['trans_date']>=dt.datetime(2013,3,17)]
df_rfm1 = 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_rfm1.rename(columns={'recent': 'recency',
                        'customer_id': 'frequency_2y',
                        'tran_amount': 'monetary_value_2y'}, inplace=True)
df_rfm1.reset_index(inplace=True)

df_rfm2 = df_transactions.groupby('customer_id').agg({'tran_amount':lambda x: x.mean()}).reset_index()
df_rfm2.rename(columns={'tran_amount':'median_purchase_2y'}, inplace=True)

df_rfm3 = df_transactions.groupby('customer_id').agg({'tran_amount':lambda x: x.std()}).reset_index()
df_rfm3.rename(columns={'tran_amount':'std_amount_2y'}, inplace=True)
```

```
df_rfm3.fillna(0,inplace=True)
```

```
## create data set with RFM variables 1yr
df_transaction_1y = df_transactions.copy()
df_transaction_1y = df_transaction_1y[df_transaction_1y['trans_date']>=dt.datetime(2014,3,17)]
df_rfm_1y = df_transaction_1y.groupby('customer_id').agg({'customer_id': lambda x: len(x)})
df_rfm_1y.rename(columns={'customer_id': 'frequency_1y'}, inplace=True)
df_rfm_1y.reset_index(inplace=True)
```

```
#Merge data
df_rfm = pd.merge(df_rfm1,df_rfm2)
df_rfm = pd.merge(df_rfm,df_rfm3)
df_rfm = pd.merge(df_rfm,df_rfm_1y)
df_rfm = pd.merge(df_rfm,df_transqt)
df_rfm = pd.merge(df_rfm,df_trans3)
df_rfm.head()
```

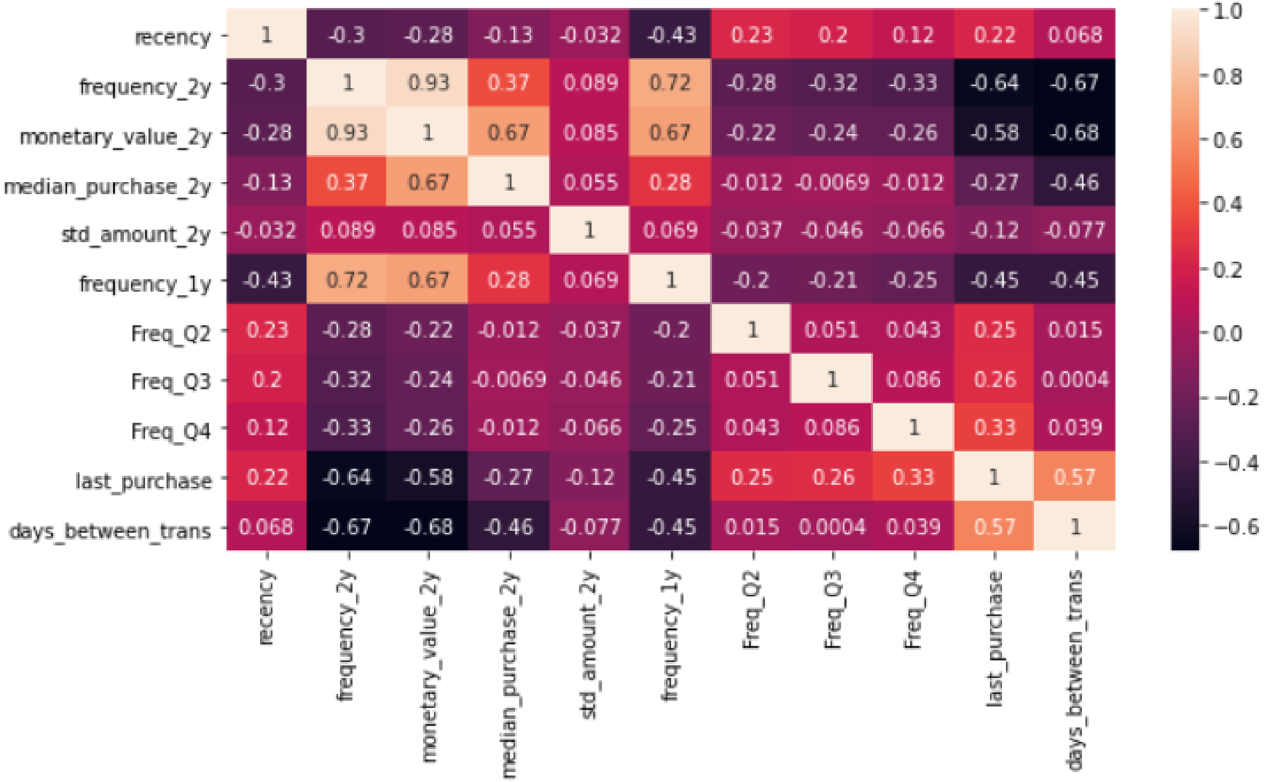
Data preparation & feature engineering

```
## merging two data sets

df_modeling1 = pd.merge(df_response,df_rfm)
df_modeling = pd.merge(df_modeling1,df_tbp)
df_modeling['Freq_Q2'] = df_modeling['Freq_Q2']/df_modeling['frequency_2y']
df_modeling['Freq_Q3'] = df_modeling['Freq_Q3']/df_modeling['frequency_2y']
df_modeling['Freq_Q4'] = df_modeling['Freq_Q4']/df_modeling['frequency_2y']
df_modeling['last_purchase'] = df_modeling['last_purchase']/df_modeling['monetary_value_2y']
df_modeling.head()
```

recency	frequency_2y	monetary_value_2y	median_purchase_2y	std_amount_2y	frequency_1y	Freq_Q2	Freq_Q3	Freq_Q4	last_purchase	days_between_trans
62.0	6	358	59.666667	20.235283	4	0.666667	0.833333	0.666667	0.108939	93.50000
36.0	11	775	70.454545	23.888757	6	0.454545	0.454545	0.363636	0.227097	71.26315
33.0	11	804	73.090909	24.010225	5	0.181818	0.818182	0.363636	0.098259	72.72222
12.0	11	831	75.545455	15.577956	3	0.545455	0.454545	0.272727	0.066185	62.04761
204.0	5	333	66.600000	25.234896	4	0.800000	1.000000	0.000000	0.270270	96.25000

Correlation heatmap



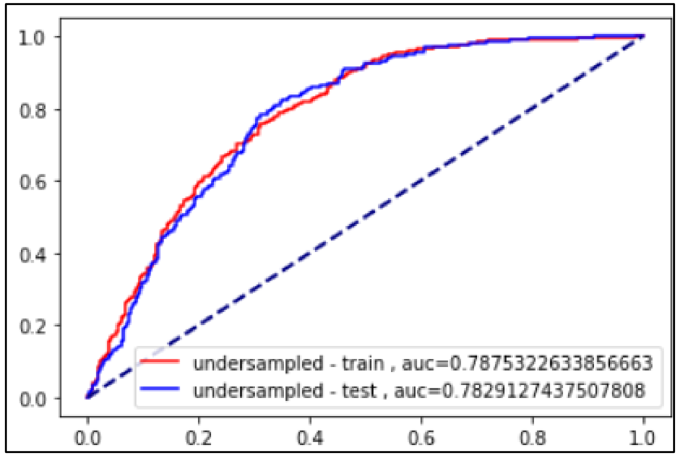
**There might be a problem in monetary_2y & frequency_2y where correlation=0.93 (multi collinearity) but both are important feature so I will keep it in the model

Result using logistic regression

Undersampled

test set

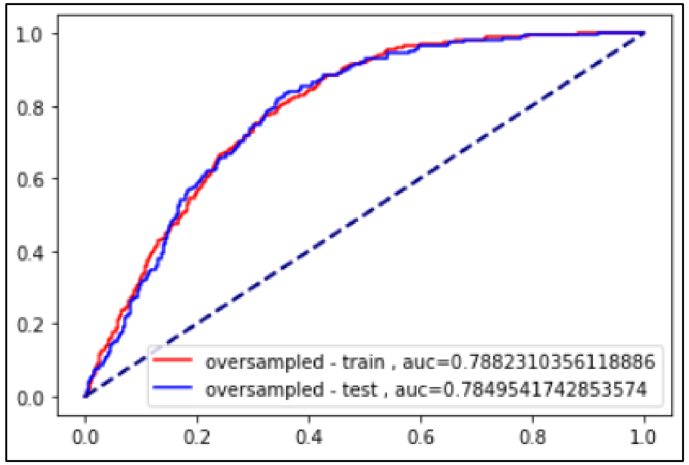
	precision	recall	f1-score	support
0	0.97	0.68	0.80	1844
1	0.20	0.79	0.32	191
accuracy			0.69	2035
macro avg	0.59	0.73	0.56	2035
weighted avg	0.90	0.69	0.75	2035



Oversampled

test set

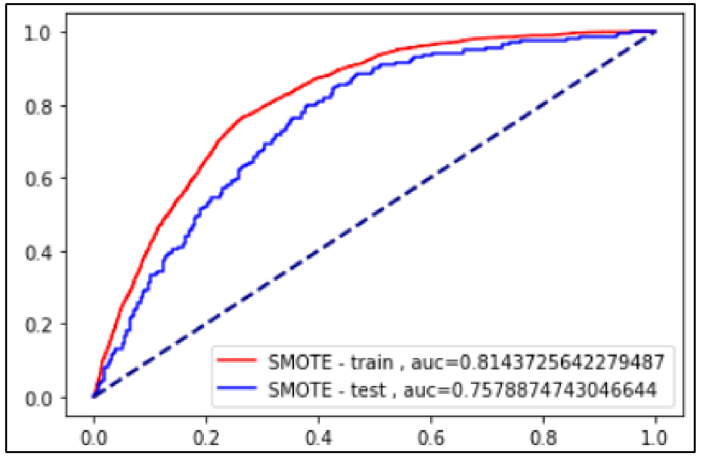
	precision	recall	f1-score	support
0	0.97	0.67	0.79	1844
1	0.20	0.80	0.32	191
accuracy			0.68	2035
macro avg	0.58	0.73	0.55	2035
weighted avg	0.90	0.68	0.75	2035



Smote

test set

	precision	recall	f1-score	support
0	0.95	0.71	0.82	1844
1	0.19	0.66	0.30	191
accuracy			0.71	2035
macro avg	0.57	0.69	0.56	2035
weighted avg	0.88	0.71	0.77	2035

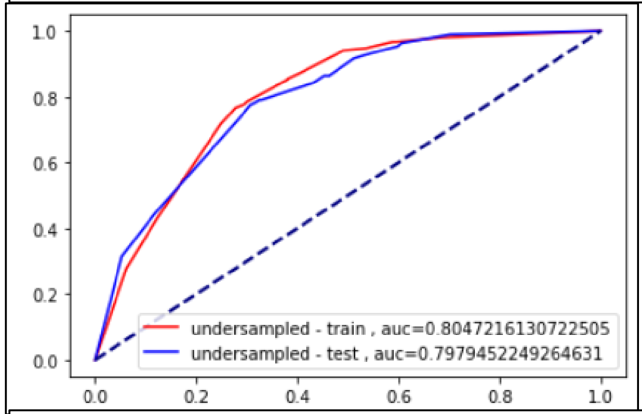


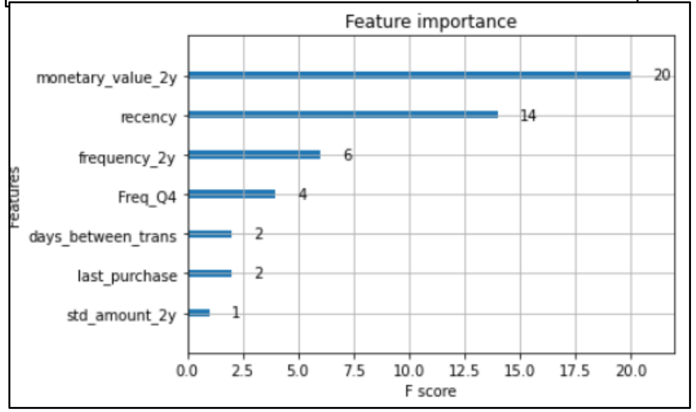
Result using XGBoost - logistic regression

Undersampled

test set

	precision	recall	f1-score	support
0	0.97	0.67	0.79	1844
1	0.20	0.79	0.32	191
accuracy			0.68	2035
macro avg	0.58	0.73	0.56	2035
weighted avg	0.90	0.68	0.75	2035

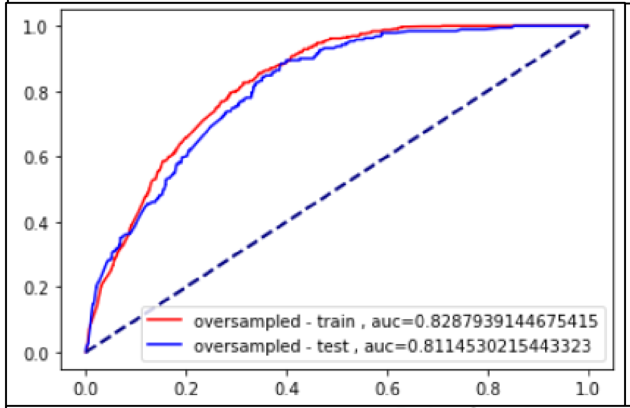


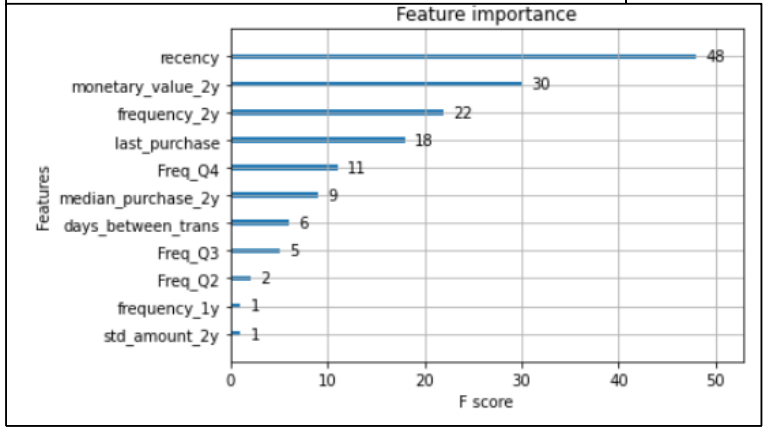


Oversampled

test set

	precision	recall	f1-score	support
0	0.98	0.65	0.78	1844
1	0.20	0.84	0.32	191
accuracy			0.67	2035
macro avg	0.59	0.75	0.55	2035
weighted avg	0.90	0.67	0.74	2035

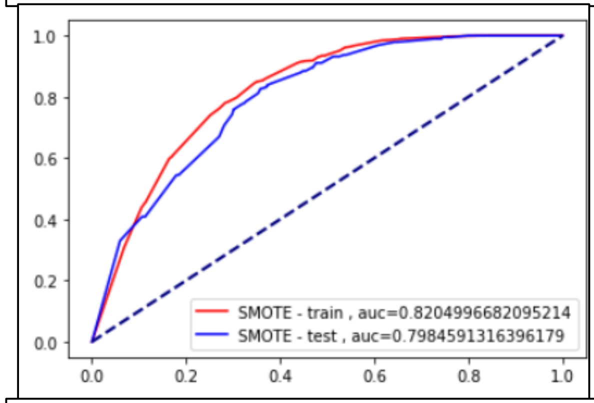


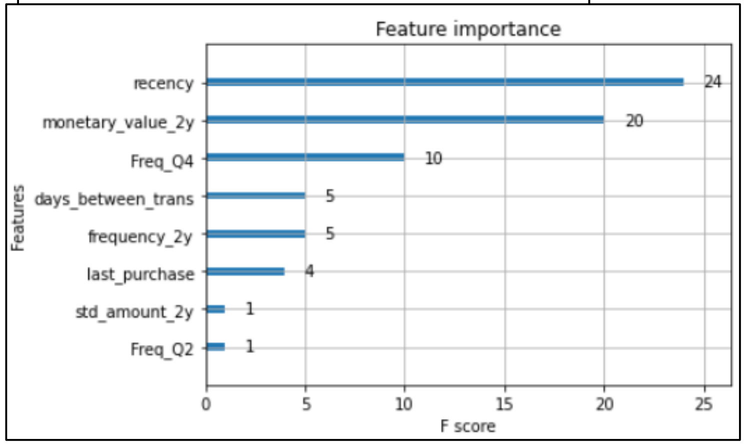


Smote

test set

	precision	recall	f1-score	support
0	0.97	0.64	0.77	1844
1	0.19	0.83	0.31	191
accuracy			0.66	2035
macro avg	0.58	0.73	0.54	2035
weighted avg	0.90	0.66	0.73	2035





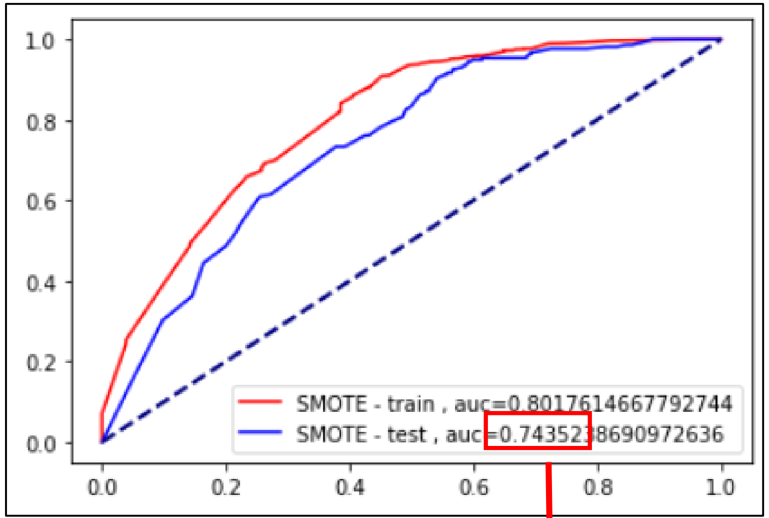
Improvement from original result

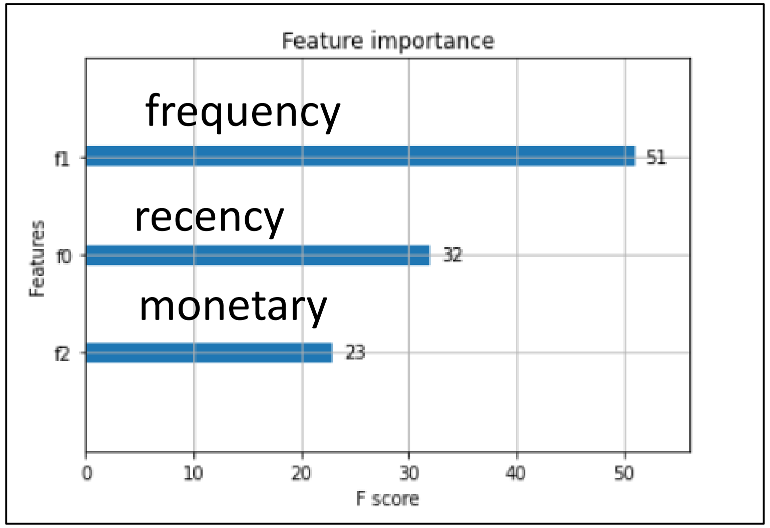
F1-score increase from 0.69 to 0.74
AUC for test set increase from 0.74 to 0.81

Original- XGBoost Smote

test set

	precision	recall	f1-score	support
0	0.95	0.60	0.74	1848
1	0.18	0.74	0.29	218
accuracy			0.62	2066
macro avg	0.57	0.67	0.51	2066
weighted avg	0.87	0.62	0.69	2066





Best result- XGBoost Oversampling

test set

	precision	recall	f1-score	support
0	0.98	0.65	0.78	1844
1	0.20	0.84	0.32	191
accuracy			0.67	2035
macro avg	0.59	0.75	0.55	2035
weighted avg	0.90	0.67	0.74	2035