In [18]:

```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%precision 3
```

Out[18]:

```
'%.3f'
```

In [141]:

```
#the oversampled dataset. I have attached it with the final submission
data = pd.read_csv('master_table_final_trim_3_os.csv')
```

In [142]:

```
#one hot encoding for distributor variable
distributor_dumm = pd.get_dummies(data['DSTRBTR_KEY'], prefix = 'distributor', drop_first=T
```

In [143]:

```
distributor_dumm.head()
```

Out[143]:

	distributor_2	distributor_3	distributor_4
0	0.0	0.0	1.0
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

In [144]:

```
#convert the distributor variables to bool
distributor_dumm['distributor_2'] = distributor_dumm['distributor_2'].astype('bool')
distributor_dumm['distributor_3'] = distributor_dumm['distributor_3'].astype('bool')
distributor_dumm['distributor_4'] = distributor_dumm['distributor_4'].astype('bool')
distributor_dumm.head()
```

Out[144]:

	distributor_2	distributor_3	distributor_4
0	False	False	True
1	False	False	False
2	True	False	False
3	False	False	False
4	False	False	False

In [145]:

```
distributor_dumm.shape
```

Out[145]:

```
(13020, 3)
```

In [146]:

```
data.shape
```

Out[146]:

```
(13020, 25)
```

In [147]:

```
#merging dummy variables with the aster data
result = pd.concat([data, distributor_dumm], axis=1)
result.shape
```

Out[147]:

```
(13020, 28)
```

In [148]:

```
#splitting into test and train
msk = np.random.rand(len(result)) < 0.8
train = result[msk]
test = result[~msk]
len(train), len(test)
x_train = train[['distributor_2', 'distributor_3', 'distributor_4','city_count', 'item_cour
y_train = train['Target']
x_test = test[['distributor_2', 'distributor_3', 'distributor_4','city_count', 'item_count'
y_test = test['Target']
```

In [149]:

```
#making sure the proportion and other distributions is preserved
sum(y_train)/len(y_train), sum(y_test)/len(y_test)
```

Out[149]:

```
(0.249, 0.245)
```

In [ ]:

```
max(x_train[])
```

In [102]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model = gnb.fit(x_train, y_train)
```

In [103]:

```
y_pred = model.predict(x_test)
```

In [104]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_pred, y_test)
```

Out[104]:

```
array([[1221,  315],
       [ 717,  358]])
```

In [105]:

```
#recall
358/sum(y_test)
```

Out[105]:

```
0.892
```

In [108]:

```
#precision
358/(358+315)
```

Out[108]:

```
0.532
```

In [107]:

```
#accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_pred, y_test)
```

Out[107]:

```
0.605
```

In [ ]:

```
```

In [157]:

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=100)
clf.fit(x_train,y_train)
y_out = model.predict(x_test)
```

In [158]:

```
y_pred = []
for i in y_out:
    if(i<0.55):
        y_pred.append(0)
    else:
        y_pred.append(1)

y_pred = np.array(y_pred)
```

In [159]:

```
confusion_matrix(y_pred, y_test)
```

Out[159]:

```
array([[1211,  256],
       [ 756,  384]])
```

In [ ]:

```
```

In [109]:

```
import keras
```

In [160]:

```
#importing original master table. I have attached it with the final submission
data = pd.read_csv('master_table_final_trim_3.csv')
```

In [161]:

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)
```

In [162]:

```
# create model
model = Sequential()
model.add(Dense(12, input_dim=6, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

In [163]:

```
#compiling
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [164]:

```
#using more wights for Delayed orders
class_weight = {1: 0.75,0: 0.25}
model.fit(x_train, y_train, epochs=10, batch_size=32, class_weight=class_weight)
```

```
Epoch 1/10
10413/10413 [==============================] - 1s 92us/step - loss: 0.2873 -
acc: 0.6796
Epoch 2/10
10413/10413 [==============================] - 1s 49us/step - loss: 0.2518 -
acc: 0.5893
Epoch 3/10
10413/10413 [==============================] - 1s 48us/step - loss: 0.2494 -
acc: 0.5455
Epoch 4/10
10413/10413 [==============================] - 1s 53us/step - loss: 0.2482 -
acc: 0.5397
Epoch 5/10
10413/10413 [==============================] - 1s 60us/step - loss: 0.2475 -
acc: 0.5408
Epoch 6/10
10413/10413 [==============================] - 1s 53us/step - loss: 0.2473 -
acc: 0.5300
Epoch 7/10
10413/10413 [==============================] - 1s 60us/step - loss: 0.2469 -
acc: 0.5265
Epoch 8/10
10413/10413 [==============================] - 1s 56us/step - loss: 0.2470 -
acc: 0.5269
Epoch 9/10
10413/10413 [==============================] - 1s 60us/step - loss: 0.2465 -
acc: 0.5356
Epoch 10/10
10413/10413 [==============================] - 1s 51us/step - loss: 0.2460 -
acc: 0.5328
```

Out[164]:

```
<keras.callbacks.History at 0x1ee7a20748>
```

In [165]:

```
y_out = model.predict(x_test)
```

In [166]:

```
y_pred = []
for i in y_out:
    if(i<0.55):  #0.55 cut off gave the best F1 score
        y_pred.append(0)
    else:
        y_pred.append(1)

y_pred = np.array(y_pred)
```

In [167]:

```
confusion_matrix(y_pred, y_test)
```

Out[167]:

```
array([[1095,  230],
       [ 872,  410]])
```

In [ ]:

```

```