In [...

```
from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here to
```

Out[265]: Click here to toggle on/off the raw code.

In [...

```
import pandas as pd
import numpy as np
import os
import warnings
import pickle

import matplotlib.pyplot as plt
import seaborn as sns

import catboost as cab
import xgboost as xgb

from sklearn.metrics import log_loss, recall_score, precision_score,roc_auc_score,
from sklearn.metrics import roc_curve, auc, precision_recall_curve
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV,
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
import shap
import scikitplot as skplt
import jm_mp
```

In [68]:

```
warnings.filterwarnings('ignore')
df = pd.read_csv(r'D:\z7z8\DS_MiniProject_ANON\processed_data.csv')
df.drop(['agegrp',
       'tensuregrp'],axis =1,inplace = True)
```

In [177]:

```
jp.wide_view(df.head())
```

	DATE_FOR	RTD_ST_CD	CustomerSegment	Tenure	Age	MART_STATUS	GENDER	CHANNEL1_6M
0	5/19/2014	ST_S0	1	16.175222	78.403833	MS_S0	F	0.0
1	5/17/2014	ST_S0	1	15.931554	70.989733	MS_S1	F	0.0

CATBoost

In [144]:

```
float_features  = df.select_dtypes(include = [np.float]).columns.tolist()
X = df.copy()
X[float_features] = X[float_features].fillna(-9)
X[float_features] = X[float_features].astype(np.int64)
print(X.dtypes)
```

```
DATE_FOR                  object
RTD_ST_CD                 object
CustomerSegment           object
Tenure                     int64
Age                        int64
MART_STATUS               object
GENDER                    object
CHANNEL1_6M                int64
CHANNEL2_6M                int64
CHANNEL3_6M                int64
CHANNEL4_6M                int64
CHANNEL5_6M                int64
METHOD1_6M                 int64
RECENT_PAYMENT             int64
PAYMENTS_6M                int64
CHANNEL1_3M                int64
CHANNEL2_3M                int64
CHANNEL3_3M                int64
CHANNEL4_3M                int64
CHANNEL5_3M                int64
METHOD1_3M                 int64
PAYMENTS_3M                int64
NOT_DI_3M                  int64
NOT_DI_6M                  int64
EVENT1_30_FLAG             int64
EVENT2_90_SUM              int64
LOGINS                     int64
POLICYPURCHASECHANNEL      int64
Call_Flag                  int64
Weekday                    int64
dtype: object
```

In [145]:

```
X = X[[col for col in X.columns if X[col].nunique() >1]]
y = X['Call_Flag']
X.drop('Call_Flag',axis = 1, inplace =True)

str_features = X_train.select_dtypes(exclude = [np.number]).columns.tolist()
num_features = X_train.select_dtypes(include = [np.number]).columns.tolist()
```

Tunning Hyperparameters

In [1...

```
X_tun = pd.concat([X,y],axis =1).sample(frac = 0.1,random_state =99).reset_index(
y_tun = X_tun['Call_Flag']
X_tun.drop('Call_Flag',axis =1,inplace =True)

params_dict  = {'depth':[3,4,5],
```

In [1...

```
%%time
rand_cv = RandomizedSearchCV(cbc,param_distributions = params_dict,cv =3,n_iter =
rand_cv.fit(X_tun,y_tun)
```

```
0:	learn: 0.6337039	total: 97.9ms	remaining: 39.1s
1:	learn: 0.5810603	total: 131ms	remaining: 26.1s
2:	learn: 0.5339764	total: 217ms	remaining: 28.7s
3:	learn: 0.4916205	total: 336ms	remaining: 33.3s
4:	learn: 0.4537154	total: 444ms	remaining: 35.1s
5:	learn: 0.4211456	total: 465ms	remaining: 30.5s
6:	learn: 0.3922660	total: 485ms	remaining: 27.2s
7:	learn: 0.3662087	total: 593ms	remaining: 29.1s
8:	learn: 0.3435400	total: 615ms	remaining: 26.7s
9:	learn: 0.3231779	total: 709ms	remaining: 27.6s
10:	learn: 0.3053513	total: 738ms	remaining: 26.1s
11:	learn: 0.2872145	total: 868ms	remaining: 28.1s
12:	learn: 0.2707427	total: 948ms	remaining: 28.2s
13:	learn: 0.2585529	total: 975ms	remaining: 26.9s
14:	learn: 0.2476974	total: 1s	remaining: 25.7s
15:	learn: 0.2362808	total: 1.11s	remaining: 26.6s
16:	learn: 0.2258844	total: 1.25s	remaining: 28.3s
17:	learn: 0.2159832	total: 1.38s	remaining: 29.3s
18:	learn: 0.2094058	total: 1.4s	remaining: 28.1s
19:	learn: 0.2017904	total: 1.46s	remaining: 27.8s
20:	learn: 0.1938533	total: 1.56s	remaining: 28.1s
21:	learn: 0.1879904	total: 1.66s	remaining: 28.5s
22:	learn: 0.1828934	total: 1.76s	remaining: 28.9s
23:	learn: 0.1784035	total: 1.86s	remaining: 29.2s
24:	learn: 0.1744390	total: 1.9s	remaining: 28.5s
25:	learn: 0.1695801	total: 2s	remaining: 28.8s
26:	learn: 0.1650295	total: 2.1s	remaining: 29s
27:	learn: 0.1618002	total: 2.18s	remaining: 29s
28:	learn: 0.1590692	total: 2.28s	remaining: 29.2s
29:	learn: 0.1560175	total: 2.34s	remaining: 28.9s
30:	learn: 0.1532236	total: 2.44s	remaining: 29.1s
31:	learn: 0.1512864	total: 2.54s	remaining: 29.2s
32:	learn: 0.1488181	total: 2.64s	remaining: 29.4s
33:	learn: 0.1464251	total: 2.74s	remaining: 29.5s
34:	learn: 0.1445460	total: 2.85s	remaining: 29.7s
35:	learn: 0.1428142	total: 2.95s	remaining: 29.8s
36:	learn: 0.1416559	total: 3.05s	remaining: 29.9s
37:	learn: 0.1401232	total: 3.16s	remaining: 30.1s
38:	learn: 0.1388164	total: 3.26s	remaining: 30.2s
39:	learn: 0.1381658	total: 3.4s	remaining: 30.6s
40:	learn: 0.1371285	total: 3.53s	remaining: 30.9s
41:	learn: 0.1358706	total: 3.67s	remaining: 31.3s
42:	learn: 0.1349021	total: 3.82s	remaining: 31.7s
43:	learn: 0.1339503	total: 3.94s	remaining: 31.9s
44:	learn: 0.1330671	total: 4.04s	remaining: 31.9s
45:	learn: 0.1325346	total: 4.14s	remaining: 31.9s
46:	learn: 0.1317178	total: 4.25s	remaining: 31.9s
47:	learn: 0.1309995	total: 4.38s	remaining: 32.1s
48:	learn: 0.1302276	total: 4.53s	remaining: 32.5s
49:	learn: 0.1296209	total: 4.67s	remaining: 32.7s
```

Model Training

In [17...

```
%%time
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify=y,random_state = 9
```

```
Wall time: 145 ms
```

In [173]:

```
%%time
cbc = rand_cv.best_estimator_.fit(X_train,y_train)

y_pred_test = cbc.predict_proba(X_test)
y_pred_train = cbc.predict_proba(X_train)
```

```
0:	learn: 0.6338401	total: 128ms	remaining: 51.1s
1:	learn: 0.5815081	total: 302ms	remaining: 1m
2:	learn: 0.5350264	total: 505ms	remaining: 1m 6s
3:	learn: 0.4930737	total: 704ms	remaining: 1m 9s
4:	learn: 0.4564222	total: 887ms	remaining: 1m 10s
5:	learn: 0.4229821	total: 1.11s	remaining: 1m 13s
6:	learn: 0.3941156	total: 1.31s	remaining: 1m 13s
7:	learn: 0.3683716	total: 1.51s	remaining: 1m 14s
8:	learn: 0.3457390	total: 1.63s	remaining: 1m 10s
9:	learn: 0.3257548	total: 1.78s	remaining: 1m 9s
10:	learn: 0.3080773	total: 1.83s	remaining: 1m 4s
11:	learn: 0.2923674	total: 1.87s	remaining: 1m
12:	learn: 0.2753166	total: 2.06s	remaining: 1m 1s
13:	learn: 0.2606070	total: 2.18s	remaining: 1m
14:	learn: 0.2486164	total: 2.34s	remaining: 1m
15:	learn: 0.2366941	total: 2.55s	remaining: 1m 1s
16:	learn: 0.2268603	total: 2.76s	remaining: 1m 2s
17:	learn: 0.2181408	total: 2.95s	remaining: 1m 2s
18:	learn: 0.2098228	total: 3.14s	remaining: 1m 3s
19:	learn: 0.2007004	total: 3.38s	remaining: 1m 4s
20:	learn: 0.1926257	total: 3.59s	remaining: 1m 4s
21:	learn: 0.1855125	total: 3.84s	remaining: 1m 5s
22:	learn: 0.1792658	total: 4.06s	remaining: 1m 6s
23:	learn: 0.1749750	total: 4.36s	remaining: 1m 8s
24:	learn: 0.1700350	total: 4.63s	remaining: 1m 9s
25:	learn: 0.1656193	total: 4.98s	remaining: 1m 11s
26:	learn: 0.1625861	total: 5.3s	remaining: 1m 13s
27:	learn: 0.1584914	total: 5.51s	remaining: 1m 13s
28:	learn: 0.1559789	total: 5.74s	remaining: 1m 13s
29:	learn: 0.1531886	total: 5.98s	remaining: 1m 13s
30:	learn: 0.1504161	total: 6.28s	remaining: 1m 14s
31:	learn: 0.1483425	total: 6.5s	remaining: 1m 14s
32:	learn: 0.1461592	total: 6.75s	remaining: 1m 15s
33:	learn: 0.1443461	total: 6.99s	remaining: 1m 15s
34:	learn: 0.1430543	total: 7.16s	remaining: 1m 14s
35:	learn: 0.1414344	total: 7.37s	remaining: 1m 14s
36:	learn: 0.1400778	total: 7.55s	remaining: 1m 14s
37:	learn: 0.1384448	total: 7.78s	remaining: 1m 14s
38:	learn: 0.1368896	total: 8.03s	remaining: 1m 14s
39:	learn: 0.1353416	total: 8.26s	remaining: 1m 14s
40:	learn: 0.1342950	total: 8.51s	remaining: 1m 14s
41:	learn: 0.1333604	total: 8.76s	remaining: 1m 14s
```

XGBoost

In [...

```
dummy_lst =[]
X2 = X.copy()
X2['Weekday'] = X2['Weekday'].map(dict(zip(list(np.arange(7)),['mon','tue','wed','t
X2['CustomerSegment'] =X2['CustomerSegment'].apply(lambda x: 'seg_'+ str(x))
for col in ['Weekday']+str_features:
    df_dummy = pd.get_dummies(X2[col],drop_first = True)
    print('{} shape:'.format(col),df_dummy.shape)
    dummy_lst.append(df_dummy)
    del df_dummy
X_dummies = pd.concat(dummy_lst,axis =1)
X2 = pd.concat([X2[num_features[:-1]],X_dummies],axis =1)
print(X_dummies.shape,X.shape,X2.shape)
del dummy_lst,X_dummies
```

```
Weekday shape: (130086, 6)
DATE_FOR shape: (130086, 7)
RTD_ST_CD shape: (130086, 50)
CustomerSegment shape: (130086, 3)
MART_STATUS shape: (130086, 4)
GENDER shape: (130086, 1)
(130086, 71) (130086, 30) (130086, 93)
```

Tunning Hyperparameters

In [157]:

```
X_tun2 = pd.concat([X2,y],axis =1).sample(frac = 0.1,random_state =99)
y_tun2 = X_tun2['Call_Flag']
X_tun2.drop('Call_Flag',axis =1,inplace =True)

params_dict2  = {'max_depth':[3,4,5],
                'n_estimators':[100,200,300,400,500],
                'learning_rate':np.arange(0.01,0.2,0.03),
                'reg_lambda':np.arange(0.1,6,0.5),
                'gamma':np.arange(0,6,1)}
```

In [...

```
%%time
xgc= xgb.XGBClassifier(objective = 'binary:logistic',early_stopping_rounds=100,eva
rand_cv2 = RandomizedSearchCV(xgc,param_distributions = params_dict2,cv =3,n_iter
rand_cv2.fit(X_tun2,y_tun2)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[21:24:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/le
arner.cc:573:
Parameters: { "early_stopping_rounds" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings bu
t
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


Wall time: 1min 4s
```

Model Training

In [16...

```
%%time
X_train2,X_test2,y_train2,y_test2 = train_test_split(X2,y,stratify=y,random_state
```

```
Wall time: 172 ms
```

In [171]:

```
%%time
warnings.filterwarnings('ignore')
xgc= rand_cv2.best_estimator_.fit(X_train2,y_train2)

y_pred_test2 = xgc.predict_proba(X_test2)
y_pred_train2 = xgc.predict_proba(X_train2)
```

```
Wall time: 442 ms
```

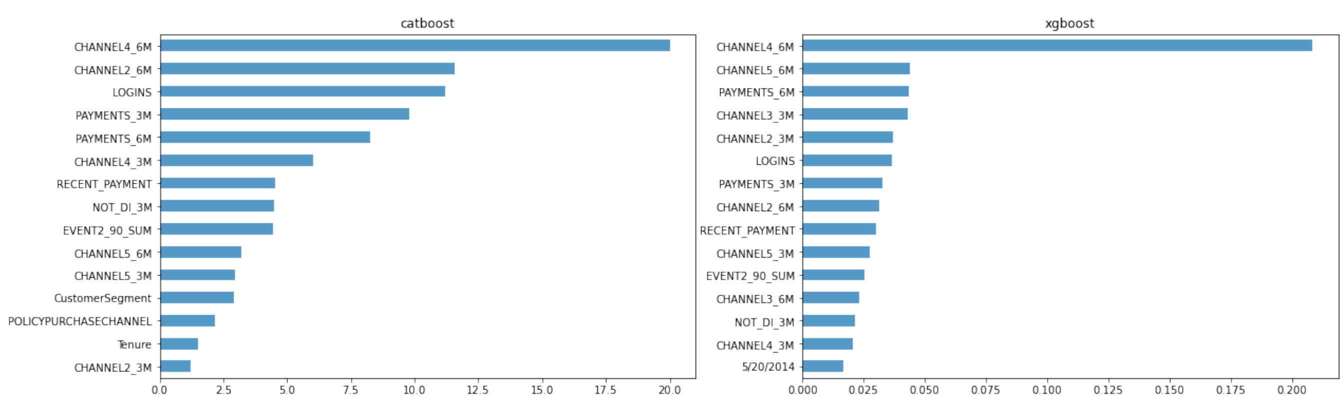
Out[171]:

	KS(t=0.041)	Roc	Log Loss	Brier Loss	Recall(t=0.041)	Precision(t=0.041)
Test Set	0.573	0.870	0.118	0.030	0.768	0.126
Training Set	0.598	0.881	0.114	0.030	0.796	0.133
Difference	0.026	0.011	0.004	0.001	0.027	0.007

Evaluations

In [...

```
fig,ax = plt.subplots(1,2,figsize = (20,6))
feat_importances = pd.Series(cbc.get_feature_importance(), index=X_train.columns).
feat_importances2 = pd.Series(xgc.feature_importances_, index=X_train2.columns).so
feat_importances[:15][::-1].plot(kind ='barh',alpha =0.75,ax= ax[0],title ='catboo
feat_importances2[:15][::-1].plot(kind ='barh',alpha =0.75,ax= ax[1],title ='xgboo
plt.show()
```



In [320]:

```
exp1 = shap.TreeExplainer(cbc)
exp2 = shap.TreeExplainer(xgc)
shap_values1 = exp1.shap_values(X_test[feat_importances.index.tolist()])
shap_values2 = exp2.shap_values(X_test2[feat_importances2.index.tolist()])
shap.summary_plot(shap_values1, X_test[feat_importances.index.tolist()])
shap.summary_plot(shap_values2, X_test2[feat_importances2.index.tolist()])
```

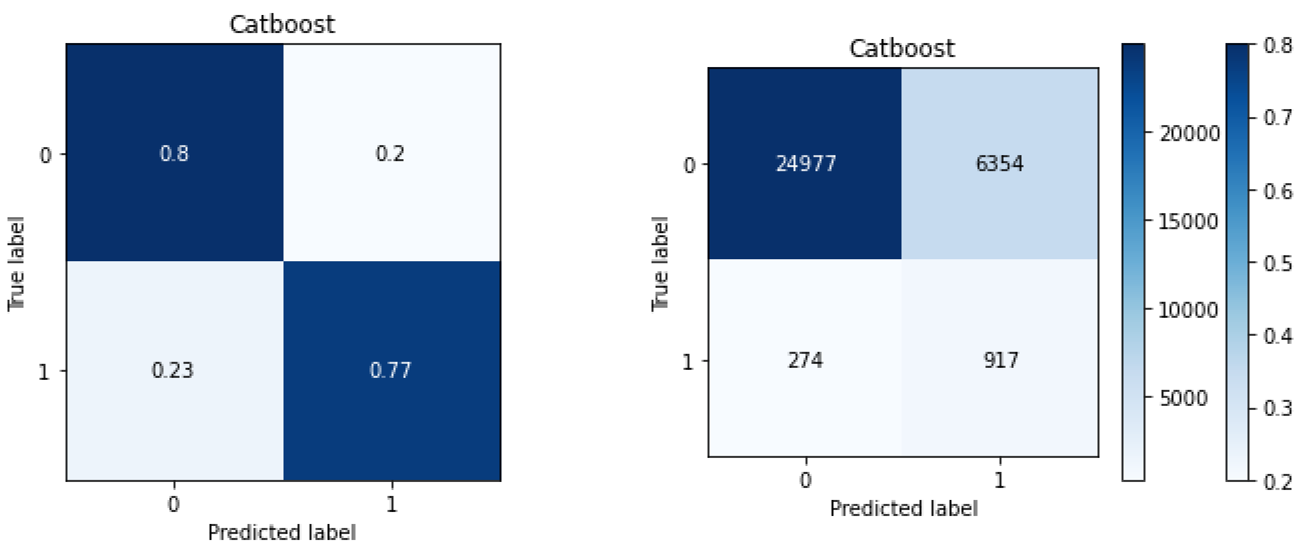
In [...

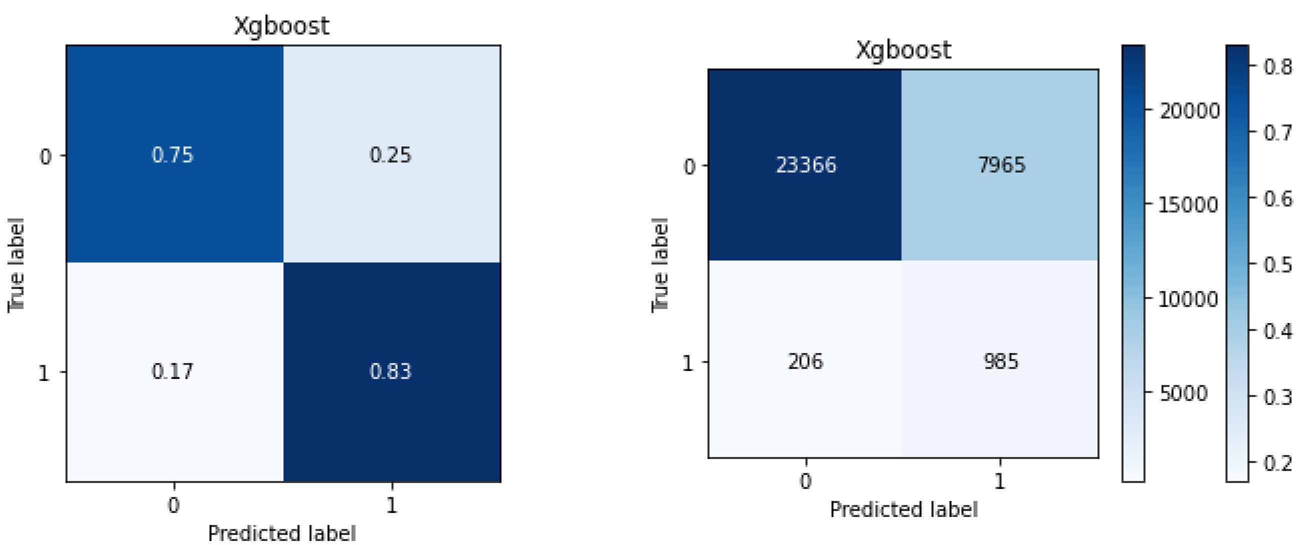
```
cat_accuracy = accuracy_score(y_test,jm_mp.prob_thresh(y_pred_test[:,1],jm_mp.ks_s
xgb_accuracy = accuracy_score(y_test2,jm_mp.prob_thresh(y_pred_test2[:,1],jm_mp.ks_
print('{} accuracy = {}%'.format('Catboost',round(100*cat_accuracy,4)))
print('{} accuracy = {}%'.format('Xgbboost',round(100*xgb_accuracy,4)))
```

```
Catboost accuracy = 79.6199%
Xgbboost accuracy = 74.8755%
```

In...

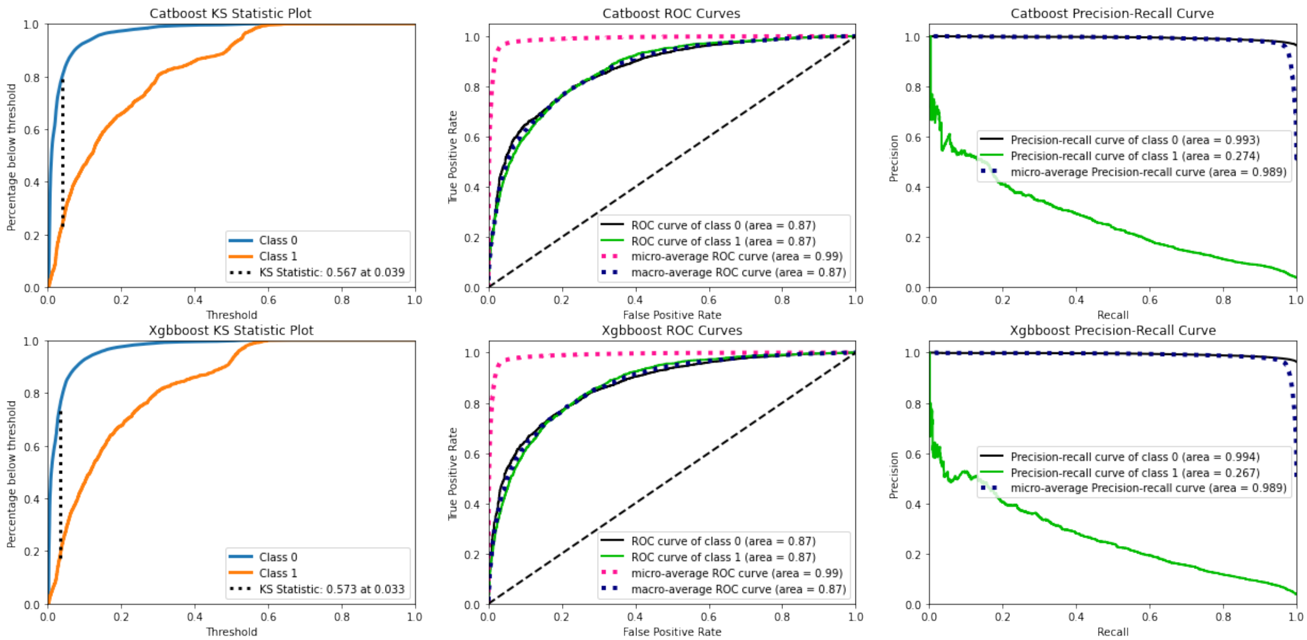
```
fig,ax = plt.subplots(1,2,figsize = (12,4))
skplt.metrics.plot_confusion_matrix(y_test,jm_mp.prob_thresh(y_pred_test[:,1],jm_mp.
skplt.metrics.plot_confusion_matrix(y_test,jm_mp.prob_thresh(y_pred_test[:,1],jm_mp.
fig,ax2 = plt.subplots(1,2,figsize = (12,4))
skplt.metrics.plot_confusion_matrix(y_test2,jm_mp.prob_thresh(y_pred_test2[:,1],jm_m
skplt.metrics.plot_confusion_matrix(y_test2,jm_mp.prob_thresh(y_pred_test2[:,1],jm_m
plt.show()
```





In [...

```
fig,ax = plt.subplots(2,3,figsize = (21,10))
skplt.metrics.plot_ks_statistic(y_test,y_pred_test,ax = ax[0,0],title = 'Catboost
```



In [...

```
jm_mp.display_sides(jm_mp.model_metrics_df(y_test, y_pred_test[:,1], y_train, y_pr
jm_mp.model_metrics_df(y_test2, y_pred_test2[:,1], y_train2, y_pred_train2[:,1],na
```

	KS(t=0.039)	Roc	Log Loss	Brier Loss	Recall(t=0.039)	Precision(t=0.039)
Catboost						
Test Set	0.567	0.867	0.118	0.030	0.770	0.126
Training Set	0.575	0.874	0.115	0.030	0.769	0.129
Difference	0.008	0.006	0.003	0.001	-0.001	0.002

	KS(t=0.033)	Roc	Log Loss	Brier Loss	Recall(t=0.033)	Precision(t=0.033)
Xgbboost						
Test Set	0.573	0.870	0.118	0.030	0.827	0.110
Training Set	0.598	0.881	0.114	0.030	0.842	0.113
Difference	0.026	0.011	0.004	0.001	0.015	0.003

In [...

```
df_p1 = pd.DataFrame(dict(zip(['y','score'],[y_test,y_pred_test[:,1]])))
df_p2 = pd.DataFrame(dict(zip(['y','score'],[y_test2,y_pred_test2[:,1]])))
fig,ax =jm_mp.create_accuracy_plots(df_p1,df_p2,show_log_odd = True,num_bins = 20,1
```

Saving Models

In [264]:

```
out_put_path = r'D:\z7z8\DS_MiniProject_ANON'
with open(os.path.join(out_put_path,'catboost_model.pkl'),'wb') as f:
    pickle.dump(cbc,f)
with open(os.path.join(out_put_path,'catboost_features.pkl'),'wb') as f:
    pickle.dump(X_train.columns,f)
with open(os.path.join(out_put_path,'xgboost_model.pkl'),'wb') as f:
    pickle.dump(xgc,f)
with open(os.path.join(out_put_path,'xgboost_features.pkl'),'wb') as f:
    pickle.dump(X_train2.columns,f)
```

Functions

In ...

```
def roc_func(ground_truth, predictions):
    """Return Roc index. Takes input like y_test, model.predict_proba(X_test)[:, 1]
    return roc_auc_score(ground_truth, predictions)



def ks_stat_func(ground_truth, predictions):
    """Return KS stat. Takes input like y_test, model.predict_proba(X_test)[:, 1]""
    thresholds, pct1, pct2, ks_statistic, max_distance_at, classes = \
        skplt.helpers.binary_ks_curve(np.array(ground_truth), np.array(predictions)
    return ks_statistic, max_distance_at



def prob_thresh(probas, thresh):
    """Convert probability lists to 0s and 1s based on given threshold value. Stric
    return [int(i) for i in (probas > thresh)]

def recall_precision_stat_func(ground_truth,probas,thresh):
    """The thresh is set when KS statistic is at the critical threshold value."""
    return (recall_score(ground_truth,prob_thresh(probas,thresh)),
            precision_score(ground_truth,prob_thresh(probas,thresh)))



def model_metrics_df(y_test, y_preda, y_train, y_preda_train, round_decimals=3):

    ks_test,ks_thresh    = ks_stat_func(y_test, y_preda)
    ks_train,ks_thresh   = ks_stat_func(y_train, y_preda_train)
    roc_test             = roc_func(y_test, y_preda)
    roc_train            = roc_func(y_train, y_preda_train)
    logloss_test         = log_loss(y_test, y_preda)
    logloss_train        = log_loss(y_train, y_preda_train)
    brierloss_test       = brier_score_loss(y_test, y_preda)
    brierloss_train      = brier_score_loss(y_train, y_preda_train)
    recall_test,precision_test    = recall_precision_stat_func(y_test, y_preda, ks_
    recall_train,precision_train  = recall_precision_stat_func(y_train, y_preda_tra

    df_metrics = pd.DataFrame({f'KS(t={round(ks_thresh, round_decimals)})': [ks_tes
                               'Roc': [roc_test, roc_train, roc_train - roc_test],
                               'Log Loss': [logloss_test, logloss_train, logloss_te
```

In...

```
def create_accuracy_plots(df_s1,df_s2,show_log_odd = True,num_bins = 10,labels = ['

    nrows =2
    ncols =1
    if show_log_odd:
        if df_s1.shape[0]>0:
            df_s1['grp'] = pd.qcut(df_s1['score'],q=num_bins,duplicates = 'drop')
            df_scores_grp1 = df_s1.groupby('grp')[['y','score']].agg([np.mean,np.si
            df_scores_grp1['logodd'] = [np.log(x/(1-x)) for x in df_scores_grp1[('y
            df_scores_grp1['logoddp'] = [np.log(x/(1-x)) for x in df_scores_grp1[('
        if df_s2.shape[0]>0:
            df_s2['grp'] = pd.qcut(df_s2['score'],q=num_bins,duplicates = 'drop')
            df_scores_grp2 = df_s2.groupby('grp')[['y','score']].agg([np.mean,np.si
            df_scores_grp2['logodd'] = [np.log(x/(1-x)) for x in df_scores_grp2[('y
            df_scores_grp2['logoddp'] = [np.log(x/(1-x)) for x in df_scores_grp2[('

    fig, ax = plt.subplots(nrows,ncols,figsize = (10*ncols,7*nrows))
    if df_s1.shape[0]>0:
        if df_s1.shape[0]>0:
            g_y  = df_scores_grp1[('y','mean')]
            g_x  = df_scores_grp1[('score','mean')]
            ax[0].scatter(g_x,g_y, marker='s', linewidth=1,color = 'lightgreen'
                       ,alpha = 0.85,s =100,label = '{}'.format(labels[0]))
            prob_error = np.sqrt(((g_y-g_x)**2).mean())
        if df_s2.shape[0]>0:
            g2_y  = df_scores_grp2[('y','mean')]
            g2_x  = df_scores_grp2[('score','mean')]
            ax[0].scatter(g2_x,g2_y, marker='s', linewidth=1,color = 'salmon'
                   ,alpha = 0.85,s =100,label = '{}'.format(labels[1]))

        ax[0].plot([0,1],[0,1],linestyle = '--',color = 'grey',linewidth =5,label =
        ax[0].set_title('{} vs {}'.format(labels[0],labels[1]),size = 16)
        ax[0].set_xlabel('Predicted Good Rate %',size = 16)
        ax[0].set_ylabel('Actual Good Rate %',size = 16)
        ax[0].set_xlim(xlim)
        ax[0].set_ylim(xlim)
        ax[0].legend(loc = 'upper left')

    if show_log_odd:
        ax[1].plot([-2,2],[-2,2],color = 'black',linewidth = 3,label = 'ideal',alph
        if df_s1.shape[0]>0:
            ax[1].plot(df_scores_grp1['logoddp'],df_scores_grp1['logodd'], marker='
        if df_s2.shape[0]>0:
            ax[1].plot(df_scores_grp2['logoddp'],df_scores_grp2['logodd'], marker='
        ax[1].set_title('{} vs {}'.format(labels[0],labels[1]),size = 24)
        ax[1].set_xlabel('Predicted log odd',size = 18)
        ax[1].set_ylabel('True log odd',size = 18)
        ax[1].legend(loc = 'upper left')

    plt.show()
    return fig,ax
```

```
[NbConvertApp] Converting notebook Models_MP.ipynb to html
[NbConvertApp] Writing 1068155 bytes to Models_MP.html
```

In [ ]: