In [174]:
```
#pip install lightgbm
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os

import lightgbm as lgb
from lightgbm import LGBMClassifier
import sklearn.metrics as metrics
import shap
import seaborn as sn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
```

In [2]:
```
print(os.getcwd())
os.chdir('D:/OneDrive/ASU/Humana_Case_Competition')
print(os.getcwd())
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz 2.44.1/bin'
```

```
C:\Users\Jinhang Jiang
D:\OneDrive\ASU\Humana_Case_Competition
```

In [3]:
```
humana = pd.read_csv('Train_Dummy.csv')
```

In [4]:
```
holdout = pd.read_csv('Test_Dummy.csv')
```

In [35]:
```
#humana.to_csv('Train_Dummy.csv',index=False)
```

In [137]:
```
#hum=humana.iloc[0:39999,:]
```

In [141]:
```
#hum_hold=humana.iloc[40000:,:]
```

In [265]:
```
label = humana['transportation_issues']
data = humana.drop(['person_id_syn','transportation_issues'], axis = 1)
data = data.fillna(data.mean())
```

In [259]:
```
#label = hum['transportation_issues']
#data = hum.drop(['person_id_syn','transportation_issues'], axis = 1)
#data = data.fillna(data.mean())
```

In [260]:
```
#labelhold = hum_hold['transportation_issues']
#datahold = hum_hold.drop(['person_id_syn','transportation_issues'], axis = 1)
#datahold = datahold.fillna(data.mean())
```

# Data Preparation

In [271]:
```
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size = 0.2, random_s
#d_train = xgboost.DMatrix(X_train, label=y_train)
#d_test = xgboost.DMatrix(X_test, label=y_test)


# Feature Scaling

#sc = StandardScaler()
#X_train = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)
```

In [272]:
```
## transform data
d_train = lgb.Dataset(X_train, label=y_train)
d_test = lgb.Dataset(X_test, label=y_test)
```

In [273]:
```
# set params
params = {'objective': 'binary',
          'metric': 'auc',
          'num_class': 1,
          'is_unbalance': True,
          'boosting_type': 'dart',
          'learning_rate': 0.11,
          'max_depth': 12,
          'num_leaves': 12,
          'feature_fraction':  0.13,
          'lambda_l1': 17,
          'lambda_l2': 890,
          'max_bin':1017,
          'subsample': 0.38,
          'num_iterations':681,
          'min_data_in_leaf':1400,
          'tree_learner':'data'
         }
```

In [274]:
```
model = lgb.train(params,
                  d_train,
#                  num_boost_round = 1000,
                  valid_sets=[d_train, d_test],
                  early_stopping_rounds=30 )
```

```
[628]   training's auc: 0.791566        valid_1's auc: 0.751153
[629]   training's auc: 0.791546        valid_1's auc: 0.751161
[630]   training's auc: 0.791517        valid_1's auc: 0.751168
[631]   training's auc: 0.791658        valid_1's auc: 0.751225
[632]   training's auc: 0.791828        valid_1's auc: 0.751276
[633]   training's auc: 0.791799        valid_1's auc: 0.751274
[634]   training's auc: 0.791768        valid_1's auc: 0.751276
[635]   training's auc: 0.79185 valid_1's auc: 0.75136
[636]   training's auc: 0.792022        valid_1's auc: 0.751397
[637]   training's auc: 0.792048        valid_1's auc: 0.751387
[638]   training's auc: 0.792007        valid_1's auc: 0.751388
[639]   training's auc: 0.791977        valid_1's auc: 0.751391
[640]   training's auc: 0.792098        valid_1's auc: 0.751332
[641]   training's auc: 0.792069        valid_1's auc: 0.751335
[642]   training's auc: 0.792065        valid_1's auc: 0.75133

[643]   training's auc: 0.792173        valid_1's auc: 0.751304
[644]   training's auc: 0.792129        valid_1's auc: 0.751309
[645]   training's auc: 0.792243        valid_1's auc: 0.751412
[646]   training's auc: 0.792396        valid_1's auc: 0.751378
```

In [264]:
```
# make predictions for test data
y_pred = model.predict(data)
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy = accuracy_score(label, predictions)

#laucpr
print("Predict test set... ")
#test_prediction = DecisionTree.predict(X_test)
score = average_precision_score(label,  y_pred)

#auc_roc
fpr, tpr, threshold = metrics.roc_curve(label, y_pred)
roc_auc = metrics.auc(fpr, tpr)


print("Accuracy: %.2f%%" % (accuracy * 100.0),
      'area under the precision-recall curve test set: {:.6f}'.format(score),
     "roc:",roc_auc,)
```

```
Predict test set...
Accuracy: 85.26% area under the precision-recall curve test set: 0.147429 roc: 0.5
```

In [43]:
```
# cross validation
%time cv_results = lgb.cv(params,d_train,num_boost_round=3000,seed=42,nfold=5,metrics={'a

cv_results
```

```
Found `num_iterations` in params. Will use it instead of argument
```

In [23]:
```
LGBC = LGBMClassifier(**params)
```

In [24]:
```
LGBC.fit(X_train,y_train)
```

```
[LightGBM] [Warning] feature_fraction is set=0.8, colsample_bytree=1.0 will be ignored.
Current value: feature_fraction=0.8
[LightGBM] [Warning] min_data_in_leaf is set=1400, min_child_samples=20 will be ignore
d. Current value: min_data_in_leaf=1400
[LightGBM] [Warning] lambda_l1 is set=17, reg_alpha=0.0 will be ignored. Current value:
lambda_l1=17
[LightGBM] [Warning] lambda_l2 is set=890, reg_lambda=0.0 will be ignored. Current valu
e: lambda_l2=890
```

Out[24]:
```
LGBMClassifier(feature_fraction=0.8, is_unbalance=True, lambda_l1=17,
               lambda_l2=890, learning_rate=0.11, max_bin=1017, max_depth=12,
               metric='auc', min_data_in_leaf=1400, num_class=1,
               num_iterations=681, num_leaves=12, objective='binary',
               subsample=0.38)
```

In [110]:
```
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)

#laucpr
print("Predict test set... ")
#test_prediction = DecisionTree.predict(X_test)
score = average_precision_score(y_test,  y_pred)

#auc_roc
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)


print("Accuracy: %.2f%%" % (accuracy * 100.0),
      'area under the precision-recall curve test set: {:.6f}'.format(score),
     "roc:",roc_auc,)
```

```
Predict test set...
Accuracy: 71.02% area under the precision-recall curve test set: 0.374013 roc: 0.743248
8838487551
```

# Accuracy, AUCPR, AUC

In [10]:
```
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 70.08%
```

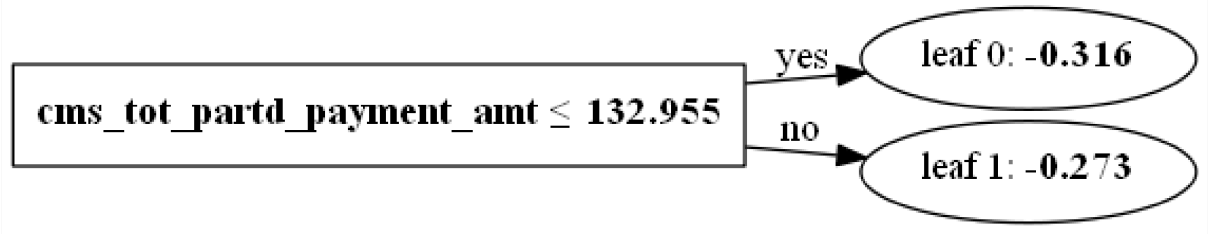
In [11]:
```
from sklearn.metrics import average_precision_score
print("Predict test set... ")
test_prediction = model.predict(X_test)
score = average_precision_score(y_test, test_prediction)
print('area under the precision-recall curve test set: {:.6f}'.format(score))
```

```
Predict test set...
area under the precision-recall curve test set: 0.352012
```

In [12]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, test_prediction)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)
```

```
0.7508939859206735
```

In [312]:
```
lgb.plot_tree(model)
fig = plt.gcf()
fig.set_dpi(300)
```



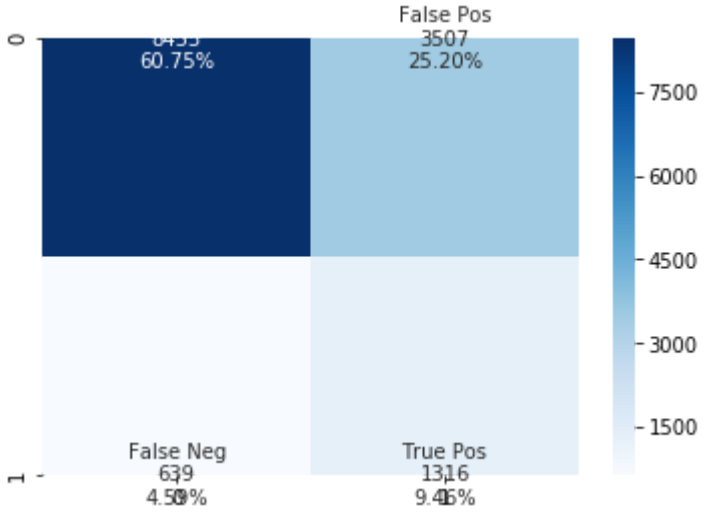

In [13]:
```
cm=confusion_matrix(y_test,np.round(y_pred,0))
cm
```

Out[13]:
```
array([[8441, 3519],
       [ 645, 1310]], dtype=int64)
```

In [13]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ['{0:0.0f}'.format(value) for value in
                cm.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm,annot=labels, fmt='', cmap='Blues')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2357de70808>
```



In [177]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
import xgboost
import shap
import sklearn.metrics as metrics
d_train = xgboost.DMatrix(X_train, label=y_train)
d_test = xgboost.DMatrix(X_test, label=y_test)
params = {
    # Parameters that we are going to tune.
    'max_depth':4,
    'min_child_weight': 1,
    'eta':0.05,
    'subsample': 0.9,
    'colsample_bytree': 0.4,
    'objective':'binary:logistic',
    "eval_metric": ["auc","logloss"],
    'gamma':5,
    "base_score": np.mean(y_train),
    'scale_pos_weight':1,
    'tree_method': "hist",
    'lambda': 80,
    'alpha': 0,
    'grow_policy': 'lossguide',
    'max_bin':256,
    'num_parallel_tree':1
}
```

In [178]:
```
model_xgboost = xgboost.train(params, d_train, 5000, evals = [(d_test, "test")], verbose_e
```

```
[0]     test-auc:0.69186        test-logloss:0.41215
Multiple eval metrics have been passed: 'test-logloss' will be used for early stopping.

Will train until test-logloss hasn't improved in 30 rounds.
[100]   test-auc:0.73824        test-logloss:0.36593
[200]   test-auc:0.73983        test-logloss:0.36472
[300]   test-auc:0.74038        test-logloss:0.36461
Stopping. Best iteration:
[272]   test-auc:0.74050        test-logloss:0.36446
```

In [179]:
```
y_pred_xgboost = model_xgboost.predict( xgboost.DMatrix(datahold))
# make predictions for test data
predictions = [round(value) for value in y_pred_xgboost]
# evaluate predictions
accuracy = accuracy_score(labelhold, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0),roc_auc_score(labelhold, predictions))
```

```
Accuracy: 85.84% 0.5410177111580067
```

In [50]:
```
cm_xgb=confusion_matrix(y_test,np.round(y_pred_xgboost,0))
cm_xgb
```

Out[50]:
```
array([[11814,   146],
       [ 1753,   202]], dtype=int64)
```

In [51]:
```
DATA = pd.DataFrame({"Target":y_test,"LighGBM":y_pred,"XGBoost":y_pred_xgboost})
```

In [52]:
```
DATA.head()
```

Out[52]:

| | Target | LighGBM | XGBoost |
|---|---|---|---|
| 50067 | 0.0 | 0.286386 | 0.056180 |
| 9816 | 0.0 | 0.218938 | 0.053977 |
| 18865 | 1.0 | 0.891468 | 0.761994 |
| 21561 | 1.0 | 0.794615 | 0.517718 |
| 7384 | 0.0 | 0.715681 | 0.286103 |

In [55]:
```
DATA["Score_Weighted"]=0.5*DATA["LighGBM"]+0.5*DATA["XGBoost"]
DATA
```

Out[55]:

| | Target | LighGBM | XGBoost | Score_Weighted |
|---|---|---|---|---|
| 50067 | 0.0 | 0.286386 | 0.056180 | 0.171283 |
| 9816 | 0.0 | 0.218938 | 0.053977 | 0.136457 |
| 18865 | 1.0 | 0.891468 | 0.761994 | 0.826731 |
| 21561 | 1.0 | 0.794615 | 0.517718 | 0.656167 |
| 7384 | 0.0 | 0.715681 | 0.286103 | 0.500892 |
| ... | ... | ... | ... | ... |
| 48648 | 0.0 | 0.587190 | 0.156943 | 0.372067 |
| 971 | 1.0 | 0.723894 | 0.331636 | 0.527765 |
| 52561 | 0.0 | 0.836873 | 0.385070 | 0.610971 |
| 18806 | 0.0 | 0.569762 | 0.179207 | 0.374484 |
| 25950 | 0.0 | 0.813424 | 0.483892 | 0.648658 |

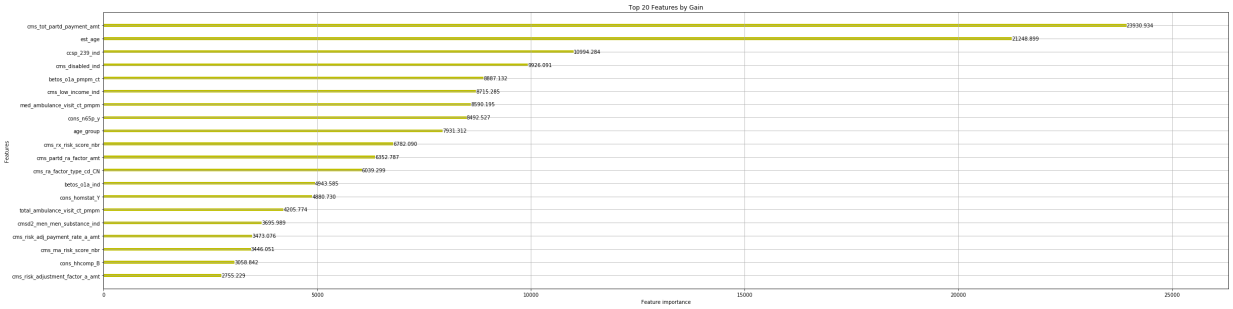
13915 rows × 4 columns

In [56]:
```
fpr, tpr, threshold = metrics.roc_curve(y_test, DATA['Score_Weighted'])
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)
```

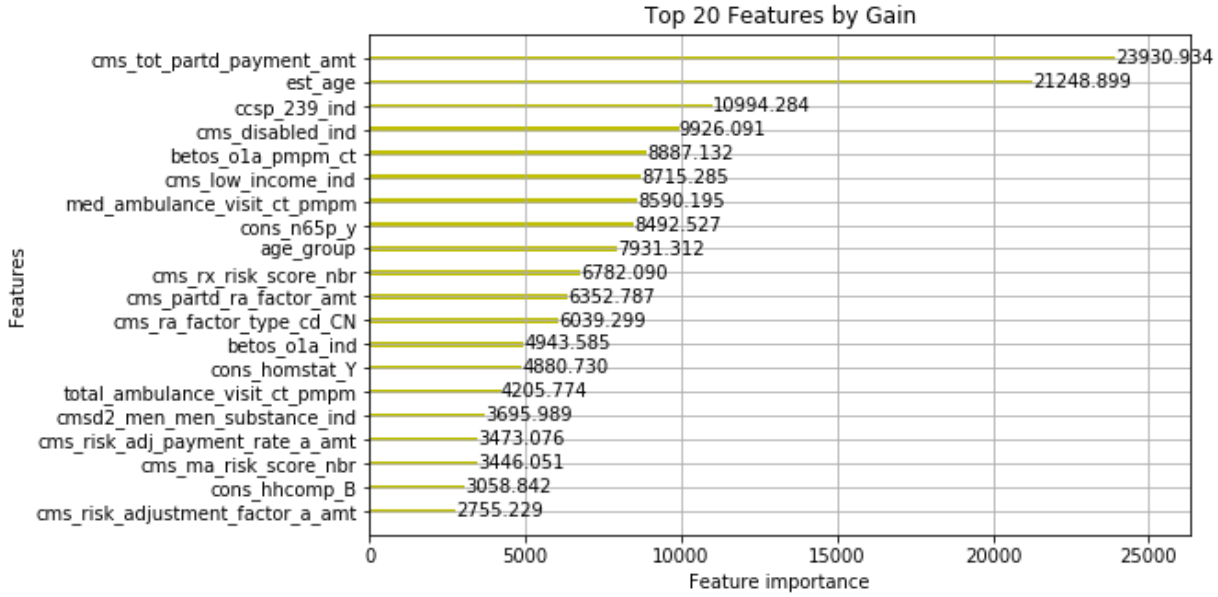
```
0.7500965280688399
```

# Variable Slection Plots

In [307]:
```
lgb.plot_importance(model,color='y',importance_type="gain",max_num_features=20, title='Top
plt.rcParams['figure.figsize'] = (18,10)
plt.show()
```




In [311]:
```
lgb.plot_importance(model, importance_type="gain",max_num_features=20, color = 'y')
plt.title('Top 20 Features by Gain')
plt.rcParams['figure.figsize'] = (12,5)
plt.show()
```




In [ ]:
```
```

# Predict and Save Holdout

In [277]:
```
win = holdout.drop(['person_id_syn','transportation_issues'], axis = 1)
```

In [278]:
```
finalprediction = model.predict(win)
```

In [280]:
```
len(finalprediction)
```

Out[280]: 17681

In [284]:
```
DATA = pd.DataFrame({"ID": ori_data['person_id_syn'], "Score":finalprediction})
DATA['RANK'] = DATA['Score'].rank(ascending=False).astype(int)
```

In [292]:
```
DATA
```

Out[292]:

| | ID | Score | RANK |
|---|---|---|---|
| 0 | 000M289dOSbe8dTL75c71YAI | 0.826966 | 306 |
| 1 | 000b16MOSTLY7A637698c5I3 | 0.243208 | 14355 |
| 2 | 0011MOdcfS9188T8aLYA3dIa | 0.298949 | 12023 |
| 3 | 001MO8SaT6dL8ae755cYA3dI | 0.245858 | 14255 |
| 4 | 001MOS3a40Tc5L1534YAeI40 | 0.746108 | 1006 |
| ... | ... | ... | ... |
| 17676 | ffc0aMO78c3ST3LY9f9bfA5I | 0.250187 | 14084 |
| 17677 | ffd22M84OSdT0LYb07A8f9I5 | 0.284506 | 12624 |
| 17678 | ffe9M2bae7OST85LYA85I650 | 0.615431 | 3092 |
| 17679 | ffeM3Ofc6e47S3TLd41e1YAI | 0.232535 | 14823 |
| 17680 | ffecMOS6770T1Lce1eY3AIa8 | 0.738643 | 1101 |

17681 rows × 3 columns

In [293]:
```
DATA.to_csv('CaseCompetition_Jinhang_Jiang.csv', index=False)
```

In [ ]:
```

```