This is to record my process of analyzing the dataset of speed dating. :)

The main target of my analysis is to find out the likelyhood a person can get a second date, which I transformed into the number of calls they got after the speed dating. Notice that on the one hand, the rules of speed dating is actually complicated with ten waves and each person can join more than one wave with different partners so I took 'iid' as key value and drop the information about the partner they are dating(this analysis focused on the features of the person himself/herself). On the other hand, the target that I was trying to predict is null in some rows so I dropped those rows to get the dataset for training. Those are the reasons why my dataset is not as large as it was originally.

Now let's get started!

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
dataset=pd.read_csv("D:\PyCharm Community Edition 2020.1.3\datasets\Speed Dati
ng Data.csv",encoding="unicode_escape")
```

In [2]:
```
dataset
```

Out[2]:

	iid	id	gender	idg	condtn	wave	round	position	positin1	order	...	attr3_3	sinc3
0	1	1.0	0	1	1	1	10	7	NaN	4	...	5.0	[illegible]
1	1	1.0	0	1	1	1	10	7	NaN	3	...	5.0	[illegible]
2	1	1.0	0	1	1	1	10	7	NaN	10	...	5.0	[illegible]
3	1	1.0	0	1	1	1	10	7	NaN	5	...	5.0	[illegible]
4	1	1.0	0	1	1	1	10	7	NaN	7	...	5.0	[illegible]
...	...	...	...	...	...	...	...	...	...	...	...	...	
8373	552	22.0	1	44	2	21	22	14	10.0	5	...	8.0	[illegible]
8374	552	22.0	1	44	2	21	22	13	10.0	4	...	8.0	[illegible]
8375	552	22.0	1	44	2	21	22	19	10.0	10	...	8.0	[illegible]
8376	552	22.0	1	44	2	21	22	3	10.0	16	...	8.0	[illegible]
8377	552	NaN	1	44	2	21	22	2	10.0	15	...	8.0	[illegible]

8378 rows × 195 columns

In [4]:
```python
pd.set_option('display.max_columns', None)
dataset.head()
```

Out[4]:

	iid	id	gender	idg	condtn	wave	round	position	positin1	order	partner	pid	match	i
0	1	1.0	0	1	1	1	10	7	NaN	4	1	11.0	0	
1	1	1.0	0	1	1	1	10	7	NaN	3	2	12.0	0	
2	1	1.0	0	1	1	1	10	7	NaN	10	3	13.0	1	
3	1	1.0	0	1	1	1	10	7	NaN	5	4	14.0	1	
4	1	1.0	0	1	1	1	10	7	NaN	7	5	15.0	1	

In [5]:
```python
##### data1 is for problem1: why someone could get calls from others?
data1=dataset[['iid','age','field','undergra','mn_sat','tuition','race','imprace','imprelig','from','zipcode','income','goal','date','go_out',
               'career','sports','tvsports','exercise','dining','museums','art','hiking','gaming','clubbing','reading','tv','theater',
               'movies','concerts','music','shopping','yoga','exphappy','expnum','them_cal']]
data1=data1.drop_duplicates()
ori=data1[data1['them_cal'].notnull()]
```

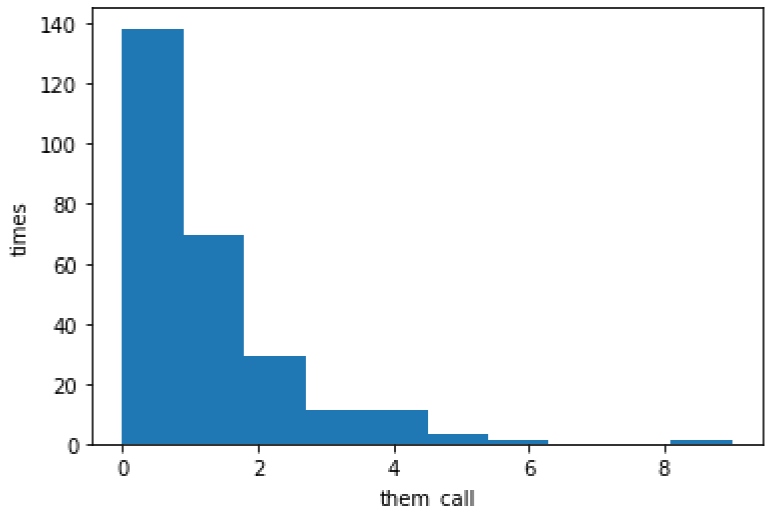
In [6]:
```python
data1.head()
```

Out[6]:

	iid	age	field	undergra	mn_sat	tuition	race	imprace	imprelig	from	zipcode
0	1	21.0	Law	NaN	NaN	NaN	4.0	2.0	4.0	Chicago	60,52
10	2	24.0	law	NaN	NaN	NaN	2.0	2.0	5.0	Alabama	35,22
20	3	25.0	Economics	NaN	NaN	NaN	2.0	8.0	4.0	Connecticut	6,26
30	4	23.0	Law	NaN	NaN	NaN	2.0	1.0	1.0	Texas	77,09
40	5	21.0	Law	NaN	NaN	NaN	2.0	8.0	1.0	Bowdoin College	94,02

In [9]:
```
#ori[['Embarked','Survived']].groupby(ori['them_cal']).sum().plot.bar()
import matplotlib.pyplot as plt
plt.hist(ori['them_cal'])
plt.xlabel('them_call')
plt.ylabel('times')
```

Out[9]: `Text(0, 0.5, 'times')`



Notice that most people can only get one or two calls from partners and the number of those who get more than 3 calls are not large enough for later works. So I put those people into the same class to increase the samples in that subset.

In [9]:
```
cn=0
for times in ori['them_cal'].values:
    if times>2:
        cn=cn+1
for iid in range(ori.shape[0]):
    if ori['them_cal'].iloc[iid]>2:
        ori['them_cal'].iloc[iid]=3
```

The next step is to encode the catogorical features. I tried target encoding this time. You can also try other ways of encoding such as label encoding/count encoding/CatBoost encoding, or the most common way--one-hot encoding(which will increase the number of columns so I did not choose for this time).

In [10]:

```python
#encode the categorical features using Target encoding
import category_encoders as ce
cat_features=['field','undergra','tuition','from','zipcode','career','income',
'mn_sat']
target_enc = ce.TargetEncoder(cols=cat_features)
target_enc.fit(ori[cat_features], ori['them_cal'])

# Transform the features, rename the columns with _target suffix, and join to
 dataframe
trainTest_TE = ori.join(target_enc.transform(ori[cat_features]).add_suffix('_t
arget'))
#valid_TE = x_test.join(target_enc.transform(x_test[cat_features]).add_suffix
('_target'))
trainTest=trainTest_TE.drop(columns=['field','undergra','tuition','from','zipc
ode','career','income','mn_sat'])
#valid=valid_TE.drop(columns=['field','undergra','tuition','from','zipcode','c
areer','income','mn_sat'])
```

In [7]:

```python
ori.head()
```

Out[7]:

	iid	age	field	undergra	mn_sat	tuition	race	imprace	imprelig	from	zipcode
0	1	21.0	Law	NaN	NaN	NaN	4.0	2.0	4.0	Chicago	60,521
10	2	24.0	law	NaN	NaN	NaN	2.0	2.0	5.0	Alabama	35,223
30	4	23.0	Law	NaN	NaN	NaN	2.0	1.0	1.0	Texas	77,096
40	5	21.0	Law	NaN	NaN	NaN	2.0	8.0	1.0	Bowdoin College	94,022
100	11	27.0	Finance	NaN	NaN	NaN	2.0	7.0	3.0	Argentina	0

In [ ]:

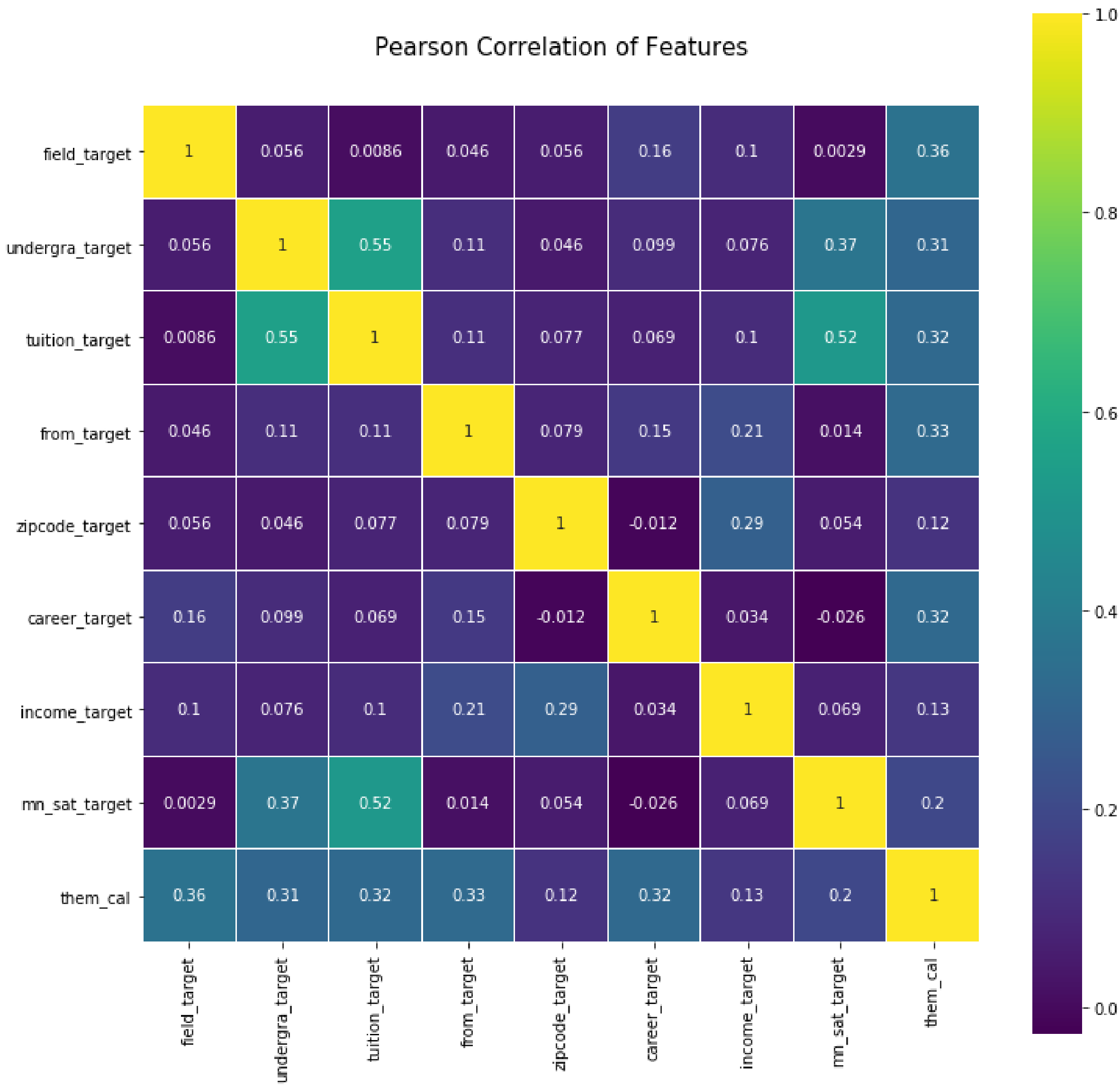
```python
##impute missing value(try scikit-learn method this time)
### missing values are mostly in columns 'expnum','field_target','undergra_tar
get','tuition_target',^(all the encoding-cause columns)
#from sklearn.experimental import enable_iterative_imputer
#from sklearn.impute import IterativeImputer
#imp = IterativeImputer(max_iter=10, random_state=0)
#imp.fit(train)
#IterativeImputer(random_state=0)
#imp.transform(valid)
#print(train.info())
```

Now starting to fill in the missing values in dataset. I tried to predict the value of missing ones according to the relationship between them.

In [12]:
```python
try_set=trainTest_TE[['field_target','undergra_target','tuition_target','from_target','zipcode_target','career_target','income_target','mn_sat_target']].join(ori['them_cal'])
```

In [13]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import MultipleLocator,FormatStrFormatter
colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(try_set.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=True)
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22fcdaa6988>
```



Since missing value in feature 'expnum'(which means the number of calls the person expect) is too much, I dropped this column.

In [14]:
```python
trainTest=trainTest.drop(columns=['expnum'])
```

Missing values in features of tuition_target,career_target,income_target,from_target,field_target,mn_sat_target and zipcode_target all need to be imputed. Here I used mice to calculate the missing value according to the other features. And according to the correlation between target-'them_cal' and other features, tuition_target and from_target seemed rather important. I used random forest model to predict those two features this time.

In [16]:
```
from impyute.imputation.cs import mice
trainTest_without=trainTest.drop(columns=['tuition_target','from_target'])
# start the MICE training
trainTest_be=mice(trainTest_without.values)
trainTest1=pd.DataFrame(trainTest_be,columns=trainTest_without.columns)
trainTestf=trainTest1.join(trainTest['tuition_target','from_target'])
```

In [18]:
```
trainTest2=trainTestf[['tuition_target','career_target','income_target','field
_target']]
tuition_notMissing_col=trainTest2[trainTest2['tuition_target'].notnull()].drop
(columns=['tuition_target'])
tuition_notMissing_target=trainTest2[trainTest2['tuition_target'].notnull()][
'tuition_target']
tuition_Missing_col=trainTest2[trainTest2['tuition_target'].isnull()].drop(col
umns=['tuition_target'])
```

In [19]:
```
trainTest3=trainTestf[['from_target','career_target','income_target','field_ta
rget']]
from_notMissing_col=trainTest3[trainTest3['from_target'].notnull()].drop(colum
ns=['from_target'])
from_notMissing_target=trainTest3[trainTest3['from_target'].notnull()]['from_t
arget']
from_Missing_col=trainTest3[trainTest3['from_target'].isnull()].drop(columns=[
'from_target'])
```

In [20]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

def impute_missing(notMissing_col,notMissing_target,missingSet_col):
    rfr=DecisionTreeRegressor(random_state=None,n_estimators=500,n_jobs=-1)
    rfr.fit(notMissing_col,notMissing_target)
    missingSet_target=rfr.predict(missingSet_col)
    rfr.score(notMissing_col,notMissing_result)
    return missingSet_target
trainTest.loc[trainTest['tuition_target'].isnull(),'tuition_target']=impute_mi
ssing(tuition_notMissing_col,tuition_notMissing_target,tuition_Missing_col)
trainTest.loc[trainTest['from_target'].isnull(),'from_target']=impute_missing(
from_notMissing_col,from_notMissing_target,from_Missing_col)
```

In [21]:
```
train_targeta= trainTest['them_cal']
train_cola=trainTest.drop(columns=['them_cal'])
```

In [49]:
```
from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set`
x_train, x_test, y_train, y_test = train_test_split(train_cola, train_targeta,
test_size=0.33, random_state=42)
```

In [51]:
```
for columns in x_test.columns:
    if x_test[columns].isna().sum()<5:
        x_test[columns].fillna(x_test[columns].mean(),inplace=True)
```

To avoid the problem of overfitting, I decreased the number of dimensions involved by PCA. I choosed the first three pricipal components, whose explained variance are larger than 0.1.

In [286]:
```
from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(n_components = 3 ,svd_solver = 'auto')
pca.fit(x_train[['sports','tvsports','exercise','dining','museums','art','hiki
ng','gaming','clubbing','reading','tv','theater',
               'movies','concerts','music','shopping','yoga']])
train1 = pca.transform(x_train[['sports','tvsports','exercise','dining','museu
ms','art','hiking','gaming','clubbing','reading','tv','theater',
               'movies','concerts','music','shopping','yoga']])
test1 = pca.transform(x_test[['sports','tvsports','exercise','dining','museum
s','art','hiking','gaming','clubbing','reading','tv','theater',
               'movies','concerts','music','shopping','yoga']])
```

In [33]:
```
pca.explained_variance_ratio_
```

Out[33]:
```
array([0.20364251, 0.1452092 , 0.1088017 ])
```

In [288]:
```
#train2=pd.DataFrame()
train1=pd.DataFrame(train1,columns=['pca1','pca2','pca3'])
test1=pd.DataFrame(test1,columns=['pca1','pca2','pca3'])
```

In [133]:
```
for idn in range(id_a.shape[0]-1):
    train1['iid'].iloc[idn]=id_a.iloc[idn]
train1.head()
```

Out[133]:

	pca1	pca2	pca3	iid
0	1.347344	-2.824201	-4.756137	169
1	-2.021160	-0.976545	-1.661397	468
2	-1.087454	-0.343775	-2.814884	529
3	-5.955742	1.868420	2.962307	297
4	-6.947236	6.532354	2.100525	61

In [138]:
```
x_train1=x_train.drop(columns=['sports','tvsports','exercise','dining','museum
s','art','hiking','gaming','clubbing','reading','tv','theater',
              'movies','concerts','music','shopping','yoga'])
x_train_f=pd.concat([x_train1,train1],axis=1)
```

In [334]:
```
x_train_f=x_train_f.dropna().drop(columns=['iid'])
```

Finish the feature scaling part.

```
In [341]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(x_train_f)
# Apply transform to both the training set and the test set.
train= scaler.transform(x_train_f)
test = scaler.transform(x_test_f)
x_train=pd.DataFrame(train,columns=[ 'age', 'race', 'imprace', 'imprelig', 'go
al', 'date', 'go_out',
       'exphappy', 'field_target', 'undergra_target', 'tuition_target',
       'from_target', 'zipcode_target', 'career_target', 'income_target',
       'mn_sat_target', 'pca1', 'pca2', 'pca3'])
x_test=pd.DataFrame(test,columns=[ 'age', 'race', 'imprace', 'imprelig', 'goa
l', 'date', 'go_out',
       'exphappy', 'field_target', 'undergra_target', 'tuition_target',
       'from_target', 'zipcode_target', 'career_target', 'income_target',
       'mn_sat_target', 'pca1', 'pca2', 'pca3'])
```

Selecting better parameters using gridSearchCV.

Notice that when using svm model, choose the "one versus rest' mode to " to predict target with more than 2 values in it.

In [ ]:

```
##gridsearchCV (for searching best parameter)
from sklearn.model_selection import GridSearchCV
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score
#parameters to choose from
param_grid = {'C': [0.0001,0.001,0.1, 1, 10, 100, 1000],
              'gamma': [0.00001,0.001, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','poly','sigmod']}

grid = GridSearchCV(SVC(), param_grid)
grid.fit(x_train_f,y_train)
print(grid.best_score_,grid.best_params_)
dict_param={}
def get_better_param(model,model_to_set,param_grid,traincol,trainTarget):
    grid_search = GridSearchCV(model_to_set,param_grid=param_grid,scoring='f1_
weighted')
    #X_train,X_test,y_train,y_test = train_test_split(traincol,trainTarget,ran
dom_state=10)
    grid_search.fit(traincol,trainTarget)
    print("Best parameters:{} for {}".format(grid_search.best_params_,model_to
_set))
    dict_param[model]=grid_search.best_params
    print("Best score on train set:{:.2f}".format(grid_search.best_score_))
    return dict_param
model_to_set=OneVsRestClassifier(SVC())
dict_param_SVM=get_better_param(svm,model_to_set,param_grid_SVM,train,y_train)
#print(estimator.get_params().keys())
#linear = svm.SVC(kernel='linear', C=1, decision_function_shape='ovo').fit(X_t
rain, y_train)
```

In [342]:

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score
model_to_set = OneVsRestClassifier(SVC())
parameters = {
    "estimator__C": [0.03,0.3,1,3,10,100,1000],
    "estimator__kernel": ["rbf","sigmod","poly"],
    "estimator__degree":[1, 2, 3, 4]}

svc = GridSearchCV(model_to_set, param_grid=parameters)

svc.fit(x_train_f, y_train[:175])

print(svc.best_score_)
print(svc.best_params_)
```

```
0.48571428571428577
{'estimator__C': 0.03, 'estimator__degree': 1, 'estimator__kernel': 'poly'}
```

In [343]:
```
from sklearn.ensemble import RandomForestClassifier

param_grid={'max_depth':[3,5,6,7,8,9,10,11,12,13,14,15,16,17]}
tree_model = RandomForestClassifier()
rf=GridSearchCV(tree_model,param_grid)
rf.fit(x_train_f, y_train[:175])

print(rf.best_params_)
```

```
{'max_depth': 13}
```

In [344]:
```
from sklearn.ensemble import AdaBoostClassifier
param_grid={'n_estimators':[10,30,100,300],'learning_rate':[0.01,0.1,0.3,1,3,1
0]}
adb=AdaBoostClassifier()
ad=GridSearchCV(adb,param_grid)
ad.fit(x_train_f, y_train[:175])
print(ad.best_params_)
```

```
{'learning_rate': 0.1, 'n_estimators': 10}
```

On the last step I used stacking to ensemble. I used Random Forest model, Adaboost and SVM in the first level. Then XGBoost in the second level.

In [347]:
```
from sklearn.model_selection import KFold

# Some useful parameters which will come in handy later on
ntrain = x_train_f.shape[0]
ntest = x_test_f.shape[0]
SEED = 0 # for reproducibility
NFOLDS = 7 # set folds for out-of-fold prediction
kf = KFold(n_splits = NFOLDS, shuffle=False)

def get_kfold_predict(clf, x_train, y_train, x_test):
    oof_train = np.zeros((ntrain,))
    oof_test = np.zeros((ntest,))
    oof_test_skf = np.empty((NFOLDS, ntest))

    for i, (train_index, test_index) in enumerate(kf.split(x_train)):
        x_tr = x_train.iloc[train_index]
        y_tr = y_train.iloc[train_index]
        x_te = x_train.iloc[test_index]

        clf.fit(x_tr, y_tr)

        oof_train[test_index] = clf.predict(x_te)
        oof_test_skf[i, :] = clf.predict(x_test)

    oof_test[:] = oof_test_skf.mean(axis=0)
    return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)
```

In [351]:

```
rf = RandomForestClassifier(n_estimators=500, warm_start=True, max_features='s
qrt',max_depth=13,
                            min_samples_split=3, min_samples_leaf=2, n_jobs=-1
, verbose=0)

ada = AdaBoostClassifier(n_estimators=10, learning_rate=0.01)

svm=SVC(C=0.03,gamma= 1, kernel='poly')
#gb = GradientBoostingClassifier(n_estimators=500, learning_rate=0.008, min_sa
mples_split=3, min_samples_leaf=2, max_depth=5, verbose=0)

rf_oof_train, rf_oof_test = get_kfold_predict(rf, x_train_f, y_train[:175], x_
test_f) # Random Forest
ada_oof_train, ada_oof_test = get_kfold_predict(ada, x_train_f, y_train[:175],
x_test_f) # AdaBoost
svm_oof_train, svm_oof_test = get_kfold_predict(svm, x_train_f, y_train[:175],
x_test_f) # Gradient Boost

x_train = np.concatenate((rf_oof_train, ada_oof_train, svm_oof_train), axis=1)
x_test = np.concatenate((rf_oof_test, ada_oof_test, svm_oof_test), axis=1)

from xgboost import XGBClassifier

gbm = XGBClassifier( n_estimators= 2000, max_depth= 4, min_child_weight= 2, ga
mma=0.9, subsample=0.8,
                     colsample_bytree=0.8, nthread= -1, scale_pos_weight=1).fi
t(x_train, y_train[:175])
predictions = gbm.predict(x_test)
prediction_eva=f1_score(y_test[:86], predictions, average='weighted')
print(prediction_eva)
#StackingSubmission = pd.DataFrame({'PassengerId': PassengerId, 'Survived': pr
edictions})
#StackingSubmission.to_csv('StackingSubmission.csv',index=False,sep=',')
```

```
[00:05:23] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.
2.0\src\learner.cc:516:
Parameters: { scale_pos_weight } might not be used.

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


0.4713798449612403
```