In [1]:
```
import pandas as pd
import numpy as np
import re
import time
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import seaborn as sb
from sklearn.metrics import recall_score,accuracy_score
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

from sklearn.cross_validation import StratifiedKFold
```

```
D:\software\Anaconda\lib\site-packages\sklearn\cross_validation.py:41: Deprecat
ionWarning: This module was deprecated in version 0.18 in favor of the model_se
lection module into which all the refactored classes and functions are moved. A
lso note that the interface of the new CV iterators are different from that of
this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [3]:
```
start = time.clock()
```

In [4]:
```
raw_data = pd.read_csv('data/training_dataset.csv',index_col=0)
```

In [5]:
```
raw_data.shape
```

Out[5]: (64006, 24)

In [6]:
```
def col_name_trans(arg1):
    temp = arg1
    temp = temp.lower()
    temp = temp.replace(' ','_')
    temp = re.sub(' +', '_', temp )
    temp = temp.replace('-','_')
    temp = temp.replace('&','_')
    temp = temp.replace('%','pct')
    temp = temp.replace('/','_')
    #if arg1 != "TOB-2/-2a__Denominator":
    temp = re.sub('_+', '_', temp )
    if temp[0].isdigit():
        temp = "c_"+ temp
    return temp
```

In [7]:
```
raw_data.rename(columns=lambda x: col_name_trans(x), inplace=True)
```

In [8]:
```
raw_data2 = raw_data.copy()
raw_data.head()
```

Out[8]:

Id	education_level	age	age_range	employment_status	gender	children	weekly_earnings	year
1	High School	51	50-59	Unemployed	Female	0	0	2005
2	Bachelor	42	40-49	Employed	Female	2	1480	2005
3	Master	47	40-49	Employed	Male	0	904	2005
4	Some College	21	20-29	Employed	Female	0	320	2005
5	High School	49	40-49	Not in labor force	Female	0	0	2005

5 rows × 24 columns

In [9]:
```
data_prep = raw_data.copy()
```

In [10]:
```
data_prep.dtypes
```

Out[10]:
```
education_level           object
age                        int64
age_range                 object
employment_status         object
gender                    object
children                   int64
weekly_earnings            int64
year                       int64
weekly_hours_worked        int64
sleeping                   int64
grooming                   int64
housework                  int64
food_drink_prep            int64
caring_for_children        int64
playing_with_children      int64
job_searching              int64
shopping                   int64
eating_and_drinking        int64
socializing_relaxing       int64
television                 int64
golfing                    int64
running                    int64
volunteering               int64
total                    float64
dtype: object
```

In [11]:
```
data_prep.age_range.unique()
```

Out[11]:
```
array(['50-59', '40-49', '20-29', '30-39', '80+', '60-69', '0-19', '70-79'], dt
ype=object)
```

In [12]:
```
data_prep.age_range.replace({'0-19':1,'20-29':2,'30-39':3,'40-49':4,
                                    '50-59':5,'60-69':6,'70-79':7,'80+':8}, inplace=
data_prep.age_range.unique()
```

Out[12]:
```
array([5, 4, 2, 3, 8, 6, 1, 7], dtype=int64)
```

In [13]:
```
data_prep.education_level.replace({'9th grade':1,'10th grade':2,'11th grade':3,'1
                                    'High School':5,'Some College':6,'Associate Deg
                                    'Master':9,'Doctoral Degree':10,'Prof. Degree':
data_prep.education_level.unique()
```

Out[13]:
```
array([ 5,  8,  9,  6,  3,  7,  1,  2, 10,  4], dtype=int64)
```

In [14]:
```
data_prep.gender.replace({'Female':0,'Male':1}, inplace=True)
data_prep.gender.unique()
```

Out[14]:
```
array([0, 1], dtype=int64)
```

In [15]:
```
##data_prep.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
##data_prep.employment_status.unique()
```

In [16]:
```
data_prep['is_weekly_earnings'] = [ 1 if x>0 else 0 for x in data_prep.weekly_ear
```

In [17]:
```
data_prep.head()
```

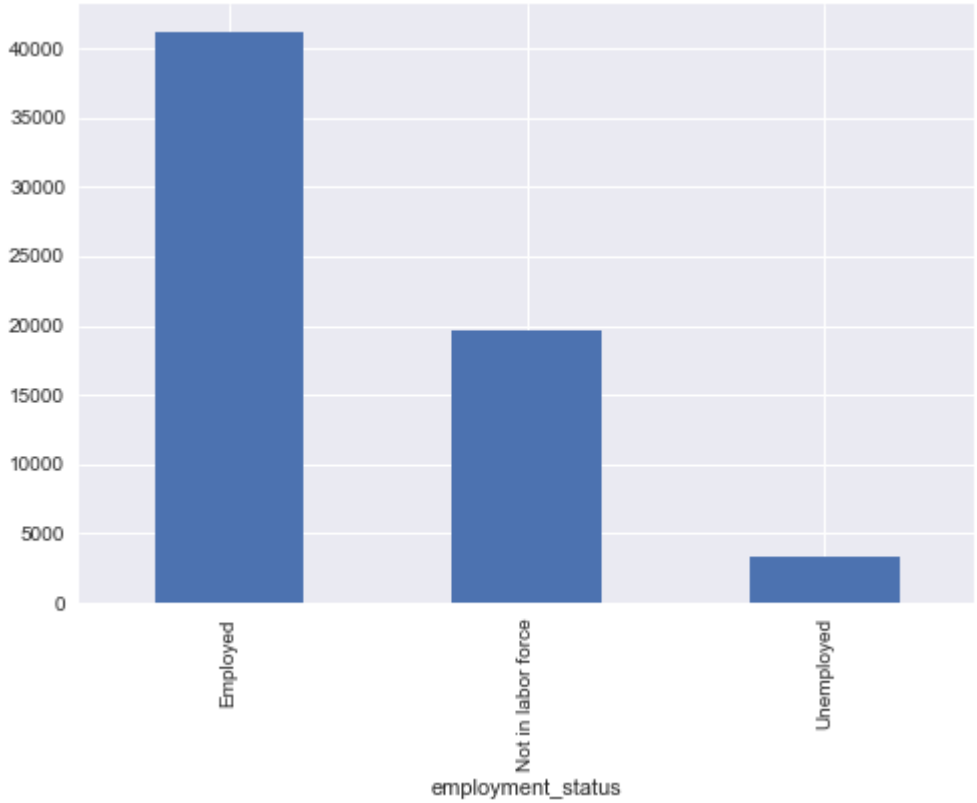
Out[17]:

Id	education_level	age	age_range	employment_status	gender	children	weekly_earnings	year
1	5	51	5	Unemployed	0	0	0	2005
2	8	42	4	Employed	0	2	1480	2005
3	9	47	4	Employed	1	0	904	2005
4	6	21	2	Employed	0	0	320	2005
5	5	49	4	Not in labor force	0	0	0	2005

5 rows × 25 columns

In [18]:
```
##data_prep = data_prep.loc[data_prep['weekly_earnings'] == 0]
```

In [19]:
```
data_prep.groupby('employment_status')['employment_status'].count().plot(kind='ba
plt.show()
```



In [20]:
```
data_prep.columns
```

Out[20]:
```
Index(['education_level', 'age', 'age_range', 'employment_status', 'gender',
       'children', 'weekly_earnings', 'year', 'weekly_hours_worked',
       'sleeping', 'grooming', 'housework', 'food_drink_prep',
       'caring_for_children', 'playing_with_children', 'job_searching',
       'shopping', 'eating_and_drinking', 'socializing_relaxing', 'television',
       'golfing', 'running', 'volunteering', 'total', 'is_weekly_earnings'],
      dtype='object')
```

In [21]:
```
col = ['education_level', 'age_range','gender','children','year','weekly_hours_wo
       'sleeping', 'grooming', 'food_drink_prep',
       'caring_for_children', 'playing_with_children', 'job_searching',
        'eating_and_drinking', 'socializing_relaxing', 'television']
```

In [22]:
```
X = pd.DataFrame(data_prep,columns=col)
X.shape
```

Out[22]:
```
(64006, 16)
```

In [23]:
```
y = pd.DataFrame(data_prep,columns=['employment_status'])
y.shape
```

Out[23]:
```
(64006, 1)
```

In [24]:
```
def model_run_all(clf,X_train_scaled,y_train_new,X_test_scaled,y_test_new,num):
    print('1', end='')
    if num == 0:
        class_type = 'Unemployed'
        y_train_new.employment_status.replace({'Unemployed':1,'Employed':0,'Not i
        y_test_new.employment_status.replace({'Unemployed':1,'Employed':0,'Not in
    elif num == 1:
        class_type = 'Employed'
        y_train_new.employment_status.replace({'Unemployed':0,'Employed':1,'Not i
        y_test_new.employment_status.replace({'Unemployed':0,'Employed':1,'Not in
    else:
        class_type = 'Not in labor force'
        y_train_new.employment_status.replace({'Unemployed':0,'Employed':0,'Not i
        y_test_new.employment_status.replace({'Unemployed':0,'Employed':0,'Not in

    print('2', end='')
    sm = SMOTE(random_state=12, ratio = 0.80)
    X_train_res, y_train_res = sm.fit_sample(X_train_scaled, y_train_new)

    print('3', end='')
    clf.fit(X_train_res, y_train_res)
    y_predicted_train = clf.predict(X_train_scaled)
    y_predicted_test = clf.predict(X_test_scaled)

    print('4', end='\n')
    print('Model:',clf.__class__.__name__,'| Class:',class_type)
    print('Train data Recall: {:.2f}'
     .format(recall_score(y_train_new,y_predicted_train)))
    print('Test data Recall: {:.2f}'
     .format(recall_score(y_test_new,y_predicted_test)))

    y_train_proba = pd.DataFrame(clf.predict_proba(X_train_scaled),columns=['prob
    y_test_proba = pd.DataFrame(clf.predict_proba(X_test_scaled),columns=['prob_0

    confusion = confusion_matrix(y_test_new, y_predicted_test)
    #print(clf.__class__.__name__,' Test confusion matrix:\n', confusion)
    return clf,y_train_proba,y_test_proba
```

In [25]:
```
def model_run_with_cross(X_train,y_train,X_test,y_test):
    X_train2 = X_train.copy()
    X_test2 = X_test.copy()
    y_train2 = y_train.copy()
    y_test2 = y_test.copy()

    X_train = X_train.loc[X_train['weekly_earnings'] == 0]
    xtrain_index = X_train.index.values
    y_train = y_train.ix[xtrain_index]

    X_test = X_test.loc[X_test['weekly_earnings'] == 0]
    xtest_index = X_test.index.values
    y_test = y_test.ix[xtest_index]

    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    y_train_new = y_train.copy()
    y_test_new = y_test.copy()

    rnd_clf = RandomForestClassifier(random_state=42,max_depth= 3,max_features='s
    svm_clf = SVC(probability=True,random_state=42,kernel = 'rbf', C=6)
    xgb_clf = xgb.XGBClassifier(max_depth=5, n_estimators=300, learning_rate=0.04

    get_all_model = []
    models = [svm_clf,xgb_clf,rnd_clf]
    for i,j in zip(models,[0,1,2]):
        temp = []
        all_clf,yy_train,yy_test = model_run_all(i,X_train_scaled,y_train_new.cop
        temp.append(all_clf)
        temp.append(yy_train)
        temp.append(yy_test)
        temp.append(all_clf.__class__.__name__)
        get_all_model.append(temp)
        print('----------------------------------------\n')

    clf_svc = get_all_model[0][0]
    prob_train_svc = get_all_model[0][1]
    prob_test_svc = get_all_model[0][2]

    clf_xgb = get_all_model[1][0]
    prob_train_xgb = get_all_model[1][1]
    prob_test_xgb = get_all_model[1][2]

    clf_rfc = get_all_model[2][0]
    prob_train_rfc = get_all_model[2][1]
    prob_test_rfc = get_all_model[2][2]

    y_pred_tr_temp = pd.concat([get_all_model[0][1]['prob_1'],get_all_model[1][1]
                                ,axis=1,names=['s','d','r'])
    y_pred_tr_temp.columns.values[0] = "Unemployed"
    y_pred_tr_temp.columns.values[1] = "Employed"
    y_pred_tr_temp.columns.values[2] = "Not in labor force"
    y_train_pred = pd.DataFrame(dict(employment_status_pred=y_pred_tr_temp.idxmax
                                 id=xtrain_index)).reset_index(drop=True)
```

```
    y_train_pred.set_index('id',inplace = True)
    X_train2 = X_train2.merge(y_train_pred, how='left',left_index=True,right_inde
    X_train2.loc[X_train2['weekly_earnings'] != 0,'employment_status_pred'] ='Emp
    y_train2.sort_index(inplace = True)
    y_train2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
    X_train2.employment_status_pred.replace({'Unemployed':0,'Employed':1,'Not in
    print('Train data Recall Final: {:.2f}'
         .format(recall_score(y_train2.employment_status,X_train2.employment_stat
    confusion = confusion_matrix(y_train2.employment_status,X_train2.employment_s
    print('Train confusion matrix:\n', confusion)

    print('------------------------------------------------------------\n')
    y_pred_tt_temp = pd.concat([get_all_model[0][2]['prob_1'],get_all_model[1][2]
                                ,axis=1,names=['s','d','r'])
    y_pred_tt_temp.columns.values[0] = "Unemployed"
    y_pred_tt_temp.columns.values[1] = "Employed"
    y_pred_tt_temp.columns.values[2] = "Not in labor force"
    y_test_pred = pd.DataFrame(dict(employment_status_pred=y_pred_tt_temp.idxmax(
                                    id=xtest_index)).reset_index(drop=True)
    y_test_pred.set_index('id',inplace = True)
    X_test2 = X_test2.merge(y_test_pred, how='left',left_index=True,right_index=T
    X_test2.loc[X_test2['weekly_earnings'] != 0,'employment_status_pred'] ='Emplo
    y_test2.sort_index(inplace = True)
    y_test2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
    X_test2.employment_status_pred.replace({'Unemployed':0,'Employed':1,'Not in l
    print('Test data Recall Final: {:.2f}'
         .format(recall_score(y_test2.employment_status,X_test2.employment_status_
    confusion = confusion_matrix(y_test2.employment_status,X_test2.employment_sta
    print('Test confusion matrix:\n', confusion)
    print('------------------------------------------------------------\n')
```

In [26]:
```
skf = StratifiedKFold(y.employment_status.values,n_folds=10)
```

In [*]:
```
for train_index,test_index in skf:
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        model_run_with_cross(X_train,y_train,X_test,y_test)
        print('#####################################################')
```

```
1234
Model: RandomForestClassifier | Class: Not in labor force
Train data Recall: 0.98
Test data Recall: 0.98
--------------------------------------

Train data Recall Final: 0.82
Train confusion matrix:
 [[ 2057     2   891]
 [  220 36446   322]
 [ 3942     9 13716]]
------------------------------------------------------------

Test data Recall Final: 0.84
Test confusion matrix:
 [[ 240    0   88]
 [  18 4067   25]
 [ 415    0 1548]]
------------------------------------------------------------
```

In [*]:
```
temp_time = time.clock()
print(temp_time - start)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
temp_time = time.clock()
print(temp_time - start)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =.1,random_stat
```

In [ ]:
```
X_train2 = X_train.copy()
X_test2 = X_test.copy()
y_train2 = y_train.copy()
y_test2 = y_test.copy()
```

In [ ]:
```
X_train = X_train.loc[X_train['weekly_earnings'] == 0]
X_train.shape
```

In [ ]:
```
X_train.index.values
```

In [ ]:
```
xtrain_index = X_train.index.values
y_train = y_train.ix[xtrain_index]
y_train.shape
```

In [ ]:
```
X_test = X_test.loc[X_test['weekly_earnings'] == 0]
X_test.shape
```

In [ ]:
```
xtest_index = X_test.index.values
y_test = y_test.ix[xtest_index]
y_test.shape
```

In [ ]:
```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
y_train_new = y_train.copy()
y_test_new = y_test.copy()
```

In [ ]:
```
def model_run_all(clf,X_train_scaled,y_train_new,X_test_scaled,y_test_new,num):
    print('1', end='')
    if num == 0:
        class_type = 'Unemployed'
        y_train_new.employment_status.replace({'Unemployed':1,'Employed':0,'Not i
        y_test_new.employment_status.replace({'Unemployed':1,'Employed':0,'Not in
    elif num == 1:
        class_type = 'Employed'
        y_train_new.employment_status.replace({'Unemployed':0,'Employed':1,'Not i
        y_test_new.employment_status.replace({'Unemployed':0,'Employed':1,'Not in
    else:
        class_type = 'Not in labor force'
        y_train_new.employment_status.replace({'Unemployed':0,'Employed':0,'Not i
        y_test_new.employment_status.replace({'Unemployed':0,'Employed':0,'Not in

    print('2', end='')
    sm = SMOTE(random_state=12, ratio = 0.80)
    X_train_res, y_train_res = sm.fit_sample(X_train_scaled, y_train_new)

    print('3', end='')
    clf.fit(X_train_res, y_train_res)
    y_predicted_train = clf.predict(X_train_scaled)
    y_predicted_test = clf.predict(X_test_scaled)

    print('4', end='\n')
    print('Model:',clf.__class__.__name__,'| Class:',class_type)
    print('Train data Recall',clf.__class__.__name__,': {:.2f}'
     .format(recall_score(y_train_new,y_predicted_train)))
    print('Test data Recall',clf.__class__.__name__,': {:.2f}'
     .format(recall_score(y_test_new,y_predicted_test)))

    y_train_proba = pd.DataFrame(clf.predict_proba(X_train_scaled),columns=['prob_
    y_test_proba = pd.DataFrame(clf.predict_proba(X_test_scaled),columns=['prob_0

    confusion = confusion_matrix(y_test_new, y_predicted_test)
    print(clf.__class__.__name__,' Test confusion matrix:\n', confusion)
    return clf,y_train_proba,y_test_proba
```

In [ ]:
```
log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42,max_depth= 3,max_features='sqrt'
svm_clf = SVC(probability=True,random_state=42,kernel = 'rbf', C=6)
xgb_clf = xgb.XGBClassifier(max_depth=5, n_estimators=300, learning_rate=0.04) #m
```

In [ ]:
```
#ada_clf,yy_train,yy_test = model_run_all(rnd_clf,X_train_scaled,y_train_new.copy
```

In [ ]:
```
get_all_model = []
models = [svm_clf,xgb_clf,rnd_clf]
for i,j in zip(models,[0,1,2]):
    temp = []
    all_clf,yy_train,yy_test = model_run_all(i,X_train_scaled,y_train_new.copy(),
    temp.append(all_clf)
    temp.append(yy_train)
    temp.append(yy_test)
    temp.append(all_clf.__class__.__name__)
    get_all_model.append(temp)
    print('--------------------------------------\n')
```

In [ ]:
```
    # choose the model for each of the class
# change the parameters for each class
# choose the cutoff probability for each class
#combine the values
#Unemployed: SVC
#Employed: XGBClassifier
#Not in labor force: RandomForestClassifier
```

In [ ]:
```
get_all_model
```

In [ ]:
```
clf_svc = get_all_model[0][0]
prob_train_svc = get_all_model[0][1]
prob_test_svc = get_all_model[0][2]

clf_xgb = get_all_model[1][0]
prob_train_xgb = get_all_model[1][1]
prob_test_xgb = get_all_model[1][2]

clf_rfc = get_all_model[2][0]
prob_train_rfc = get_all_model[2][1]
prob_test_rfc = get_all_model[2][2]
```

In [ ]:
```
y_pred_tr_temp = pd.concat([get_all_model[0][1]['prob_1'],get_all_model[1][1]['pr
                            ,axis=1,names=['s','d','r'])
y_pred_tr_temp.columns.values[0] = "Unemployed"
y_pred_tr_temp.columns.values[1] = "Employed"
y_pred_tr_temp.columns.values[2] = "Not in labor force"

y_pred_tr_temp.head()
```

In [ ]:
```
X_train.index.values
```

In [ ]:
```
y_train_pred = pd.DataFrame(dict(employment_status_pred=y_pred_tr_temp.idxmax(axi
                                 id=xtrain_index)).reset_index(drop=True)
y_train_pred.set_index('id',inplace = True)
```

In [ ]:
```
y_train_pred.head()
```

In [ ]:
```
X_train2.shape
```

In [ ]:
```
X_train2 = X_train2.merge(y_train_pred, how='left',left_index=True,right_index=Tr
```

In [ ]:
```
X_train2.employment_status_pred.unique()
```

In [ ]:
```
X_train2.head()
```

In [ ]:
```
X_train2.loc[X_train2['weekly_earnings'] != 0,'employment_status_pred'] ='Employe
X_train2.shape
```

In [ ]:
```
X_train2.employment_status_pred.unique()
```

In [ ]:
```
X_train2.head()
```

In [ ]:
```
y_train2.sort_index(inplace = True)
y_train2.head()
```

In [ ]:
```
y_train2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor for
X_train2.employment_status_pred.replace({'Unemployed':0,'Employed':1,'Not in labo
```

In [ ]:
```
print('Train data Recall Final: {:.2f}'
     .format(recall_score(y_train2.employment_status,X_train2.employment_status_p

confusion = confusion_matrix(y_train2.employment_status,X_train2.employment_statu
print('Train confusion matrix:\n', confusion)
```

In [ ]:
```

```

In [ ]:
```
y_pred_tt_temp = pd.concat([get_all_model[0][2]['prob_1'],get_all_model[1][2]['pr
                            ,axis=1,names=['s','d','r'])
y_pred_tt_temp.columns.values[0] = "Unemployed"
y_pred_tt_temp.columns.values[1] = "Employed"
y_pred_tt_temp.columns.values[2] = "Not in labor force"
y_test_pred = pd.DataFrame(dict(employment_status_pred=y_pred_tt_temp.idxmax(axis
                                id=xtest_index)).reset_index(drop=True)
y_test_pred.set_index('id',inplace = True)
X_test2 = X_test2.merge(y_test_pred, how='left',left_index=True,right_index=True,
X_test2.loc[X_test2['weekly_earnings'] != 0,'employment_status_pred'] ='Employed'
y_test2.sort_index(inplace = True)
y_test2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor forc
X_test2.employment_status_pred.replace({'Unemployed':0,'Employed':1,'Not in labor
print('Test data Recall Final: {:.2f}'
     .format(recall_score(y_test2.employment_status,X_test2.employment_status_pre
confusion = confusion_matrix(y_test2.employment_status,X_test2.employment_status_
print('Train confusion matrix:\n', confusion)
```

In [ ]:
```
y_pred_tt_temp = pd.concat([get_all_model[0][2]['prob_1'],get_all_model[1][2]['pr
                            ,axis=1,names=['s','d','r'])
y_pred_tt_temp.columns.values[0] = "Unemployed"
y_pred_tt_temp.columns.values[1] = "Employed"
y_pred_tt_temp.columns.values[2] = "Not in labor force"

y_pred_tt_temp.head()
```

In [ ]:
```
y_test_pred = pd.DataFrame(dict(employment_status_pred=y_pred_tt_temp.idxmax(axis
                                id=xtest_index)).reset_index(drop=True)
y_test_pred.set_index('id',inplace = True)
```

In [ ]:
```
X_test2 = X_test2.merge(y_test_pred, how='left',left_index=True,right_index=True,
```

In [ ]:
```
X_test2.loc[X_test2['weekly_earnings'] != 0,'employment_status_pred'] ='Employed'
X_test2.shape
```

In [ ]:
```
X_test2.head()
```

In [ ]:
```
y_test2.sort_index(inplace = True)
y_test2.head()
```

In [ ]:
```
y_test2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor forc
X_test2.employment_status_pred.replace({'Unemployed':0,'Employed':1,'Not in labor
```

In [ ]:
```
print('Test data Recall Final: {:.2f}'
     .format(recall_score(y_test2.employment_status,X_test2.employment_status_pre

confusion = confusion_matrix(y_test2.employment_status,X_test2.employment_status_
print('Train confusion matrix:\n', confusion)
```

In [ ]:
```

```

In [ ]:
```
y_pred_final.employment_status_pred.unique()
```

In [ ]:
```
X_train2.shape
```

In [ ]:
```
y_train2 = y_train.copy()
y_train2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor for
y_train_pred.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
y_pred_tt_temp = pd.concat([get_all_model[0][2]['prob_1'],get_all_model[1][2]['pr
                            ,axis=1,names=['s','d','r'])
y_pred_tt_temp.columns.values[0] = "Unemployed"
y_pred_tt_temp.columns.values[1] = "Employed"
y_pred_tt_temp.columns.values[2] = "Not in labor force"

y_pred_tt_temp.head()
```

In [ ]:
```
y_train_pred = pd.DataFrame(y_pred_tr_temp.idxmax(axis=1),columns=['employment_st
y_train_pred.head()
```

In [ ]:
```
y_test_pred = pd.DataFrame(y_pred_tt_temp.idxmax(axis=1),columns=['employment_sta
y_test_pred.head()
```

In [ ]:
```
temp_time = time.clock()
print(temp_time - start)
```

In [ ]:
```
y_train2 = y_train.copy()
y_train2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor for
y_train_pred.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
```

In [ ]:
```
y_test2 = y_test.copy()
y_test2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor forc
y_test_pred.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
```

In [ ]:
```
y_train2.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor for
y_train_pred.employment_status.replace({'Unemployed':0,'Employed':1,'Not in labor
print('Train data Recall Final: {:.2f}'
     .format(recall_score(y_train2,y_train_pred,average='macro')))

confusion = confusion_matrix(y_train2, y_train_pred)
print('Test confusion matrix:\n', confusion)
```

In [ ]:
```
print('Test data Recall Final: {:.2f}'
     .format(recall_score(y_test2,y_test_pred,average='micro')))

confusion = confusion_matrix(y_test2, y_test_pred)
print('Test confusion matrix:\n', confusion)
```

In [ ]:
```
0.72+.88+.77
```

In [ ]:
```

```

In [ ]:
```
raw_data_test = pd.read_csv('data/test_dataset.csv',index_col=0)
```

In [ ]:
```
raw_data_test.shape
```

In [ ]:
```
raw_data_test.rename(columns=lambda x: col_name_trans(x), inplace=True)
```

In [ ]:
```
raw_data_test.head()
```

In [ ]:
```
data_prep_test = raw_data_test.copy()
raw_data_test2 = raw_data_test.copy()
```

In [ ]:
```
data_prep_test.age_range.unique()
```

In [ ]:
```
data_prep_test.age_range.replace({'0-19':1,'20-29':2,'30-39':3,'40-49':4,
                                  '50-59':5,'60-69':6,'70-79':7,'80+':8}, inplace=
data_prep_test.age_range.unique()
```

In [ ]:
```
data_prep_test.education_level.replace({'9th grade':1,'10th grade':2,'11th grade'
                                 'High School':5,'Some College':6,'Associate Deg
                                 'Master':9,'Doctoral Degree':10,'Prof. Degree':
data_prep_test.education_level.unique()
```

In [ ]:
```
data_prep_test.gender.replace({'Female':0,'Male':1}, inplace=True)
data_prep_test.gender.unique()
```

In [ ]:
```
data_prep_test['is_weekly_earnings'] = [ 0 if x>0 else 1 for x in data_prep_test.
```

In [ ]:
```
data_prep_test2 = data_prep_test.copy()
```

In [ ]:
```
data_prep_test2.shape
```

In [ ]:
```
data_prep_test = data_prep_test.loc[data_prep_test['weekly_earnings'] == 0]
```

In [ ]:
```
data_prep_test.shape
```

In [ ]:
```
X_final_test = pd.DataFrame(data_prep_test,columns=col)
X_final_test.shape
```

In [ ]:
```
X_final_test_scaled = scaler.transform(X_final_test)
```

In [ ]:
```
y_svc_proba = pd.DataFrame(clf_svc.predict_proba(X_final_test_scaled),columns=['p
y_xgb_proba = pd.DataFrame(clf_xgb.predict_proba(X_final_test_scaled),columns=['p
y_rfc_proba = pd.DataFrame(clf_rfc.predict_proba(X_final_test_scaled),columns=['p
```

In [ ]:
```
y_pred_temp = pd.concat([y_svc_proba['prob_1'],y_xgb_proba['prob_1'],y_rfc_proba[
                         ,axis=1,names=['s','d','r'])
y_pred_temp.columns.values[0] = "Unemployed"
y_pred_temp.columns.values[1] = "Employed"
y_pred_temp.columns.values[2] = "Not in labor force"
```

In [ ]:
```
y_pred_final = pd.DataFrame(dict(employment_status=y_pred_temp.idxmax(axis=1),
                                 id=X_final_test.index.values)).reset_index(drop=
y_pred_final.set_index('id',inplace = True)
```

In [ ]:
```
y_pred_final.employment_status.unique()
```

In [ ]:
```
raw_data_test2 = raw_data_test2.drop(['employment_status'], axis=1)
```

In [ ]:
```
raw_data_test2 = raw_data_test2.merge(y_pred_final, how='left',left_index=True,ri
```

In [ ]:
```
raw_data_test2.head()
raw_data_test2.employment_status.value_counts()
```

In [ ]:
```
raw_data_test2.loc[raw_data_test2['weekly_earnings'] != 0,'employment_status'] ='
raw_data_test2.tail()
```

In [ ]:
```
raw_data_test2.employment_status.unique()
```

In [ ]:
```
raw_data_test2.shape
```

In [ ]:
```
raw_data_test2.employment_status.unique()
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = ada_clf.estimators_[5]
```

In [ ]:
```
# Export the image to a dot file
export_graphviz(tree, out_file = 'tree.dot', feature_names = col, rounded = True,
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Write graph to a png file
graph.write_png('tree.png')
```

In [ ]:
```
log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42,max_depth= 3,max_features='sqrt'
svm_clf = SVC(probability=True,random_state=42,kernel = 'rbf', C=6)
xgb_clf = xgb.XGBClassifier(max_depth=5, n_estimators=300, learning_rate=0.04) #m
```

In [ ]:
```
grid_values = {'n_estimators':[100,200,300,400,500,700,800],'max_features':['sqrt
for i, eval_metric in enumerate(('recall','roc_auc')):
    grid_clf_custom = GridSearchCV(rnd_clf, param_grid=grid_values, scoring=eval_

    ada_clf,yy_train,yy_test = model_run_all(grid_clf_custom,X_train_scaled,y_tra
    #grid_clf_custom.fit(X_twovar_train, y_train)

    print('Grid best parameter (max. {0}): {1}'
          .format(eval_metric, ada_clf.best_params_))
    print('Grid best score ({0}): {1}'
          .format(eval_metric, ada_clf.best_score_))
```

In [ ]:
```
grid_values = {'kernel':['linear','rbf'],'C':[1,2,3,4,5,6]}
for i, eval_metric in enumerate(('recall','roc_auc')):
    grid_clf_custom = GridSearchCV(svm_clf, param_grid=grid_values, scoring=eval_

    ada_clf,yy_train,yy_test = model_run_all(grid_clf_custom,X_train_scaled,y_tra
    #grid_clf_custom.fit(X_twovar_train, y_train)

    print('Grid best parameter (max. {0}): {1}'
          .format(eval_metric, ada_clf.best_params_))
    print('Grid best score ({0}): {1}'
          .format(eval_metric, ada_clf.best_score_))
```

In [ ]:
```
grid_values = {'max_depth':[1,2,3,4,5], 'n_estimators':[100,200,300,400,500],'lea
for i, eval_metric in enumerate(('recall','roc_auc')):
    grid_clf_custom = GridSearchCV(xgb_clf, param_grid=grid_values, scoring=eval_

    ada_clf,yy_train,yy_test = model_run_all(grid_clf_custom,X_train_scaled,y_tra
    #grid_clf_custom.fit(X_twovar_train, y_train)

    print('Grid best parameter (max. {0}): {1}'
          .format(eval_metric, ada_clf.best_params_))
    print('Grid best score ({0}): {1}'
          .format(eval_metric, ada_clf.best_score_))
```

In [ ]:
```

```