Detecting Depression with Machine Learning

In the region of Lechi during the last five years, the rate of depression has drastically increased by a margin of 20%. The regional government, concerned about this alarming situation, has decided to implement an early detection program to help potential patients.
In this assignment, you will use machine learning to predict the potential cases of depression in the region, by using the data available in the registry office of Lechi.

In [56]:
```
# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats
from matplotlib import cm
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
```

1 - Data Preparation

The dataset contains a list of inhabitants of the region with their respective data. It contains 15164 rows per 13 columns. First let's import the *.csv* file using pandas and assign it to a DataFrame named *dataset*.

In [57]:
```
# Read CSV and print dataset shape
dataset = pd.read_csv('stress_train.csv')
print('stress_train.csv')
print(dataset.shape)
dataset.head()
```

```
stress_train.csv
(15164, 13)
```

Out[57]:

	age	workclass	education	education_years	marital_status	occupation	relationship	race	sex	hours_week	country	permit	stress
0	43	Private	HS-grad	9	Divorced	Transport-moving	Not-in-family	NaN	Male	16	United-States	B	0
1	31	Self-emp-not-inc	Some-college	10	Married-civ-spouse	Craft-repair	Husband	NaN	Male	40	United-States	D	0
2	47	Self-emp-not-inc	Some-college	10	Divorced	Exec-managerial	Not-in-family	NaN	Female	65	United-States	C	0
3	32	Private	HS-grad	9	Never-married	Sales	Own-child	NaN	Female	35	United-States	B	0
4	25	State-gov	Some-college	10	Divorced	Protective-serv	Other-relative	NaN	Male	45	United-States	A	0

2 - Data Exploration

Using *Pandas* method *.describe* we can have a summary of the statistics of the variables contained in *dataset*, including categorical variables.

In [58]:
```
# Statistics of X features matrix
dataset.describe(include = 'all')
```

Out[58]:

	age	workclass	education	education_years	marital_status	occupation	relationship	race	sex	hours_week	country	permit	s
count	15164.000000	14228	15164	15164.000000	15164	14224	15164	2328	15164	15164.000000	14901	15164	15164.00
unique	NaN	8	16	NaN	5	14	6	4	2	NaN	40	5	
top	NaN	Private	HS-grad	NaN	Married-civ-spouse	Adm-clerical	Husband	Black	Male	NaN	United-States	C	
freq	NaN	10715	4890	NaN	6080	2197	4666	1596	7607	NaN	13596	3086	
mean	38.211949	NaN	NaN	10.082036	NaN	NaN	NaN	NaN	NaN	39.496835	NaN	NaN	0.21
std	13.769714	NaN	NaN	2.506576	NaN	NaN	NaN	NaN	NaN	12.407518	NaN	NaN	0.40
min	17.000000	NaN	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	1.000000	NaN	NaN	0.00
25%	27.000000	NaN	NaN	9.000000	NaN	NaN	NaN	NaN	NaN	36.000000	NaN	NaN	0.00
50%	37.000000	NaN	NaN	10.000000	NaN	NaN	NaN	NaN	NaN	40.000000	NaN	NaN	0.00
75%	47.000000	NaN	NaN	12.000000	NaN	NaN	NaN	NaN	NaN	43.000000	NaN	NaN	0.00
max	90.000000	NaN	NaN	16.000000	NaN	NaN	NaN	NaN	NaN	99.000000	NaN	NaN	1.00

We see that there are four numerical variables, a binary *stress* (outcome variable).
The average of the population is 38 years, with 10 years of education and about 40 working hours per week. The incidence of *stress* (depression) is about 21%.

2.1 - Handling Missing Values

Let's look at the numerosity of each variable and the presence of missing values :

- The numerical variables *age*, _education*years*, _hours*week* and categorical variables *education*, _marital*status*, *relationship*, *sex* and *permit* are complete.
- Categorical variables *workclass*, *occupation*, *race* and *country* have missing values.
- In particular *race* has much more missing data compared to others.

In [59]:
```
# 14228/15164 = 93%
# 2077/15164 = 13%
```

Cleaning data approach :

In order to have a consistent dataset with complete information, without compromising the rapresentativity of the sample, we will approach to missing data problem in the following way :
race has just 13% of observations compared to the dataset size. We could try to infer the *race* from other variables but we will introduce multicollinearity among the data. Those synthetic points would be used later in the model to infer *stress*. In conclusion since *race* has so few observations, it will be removed from dataset.
Second step is to remove all the rows of the variables that have missing values.
Droping "race" and rows with missing data allow us to work with more than 90% of the initial dataset, assuring the consistency of information.

In [60]:
```
# Step 1 : Drop 'race'
dataset.drop('race', axis = 1, inplace = True)
#dataset.describe(include = 'all')
```

In [61]:
```
# Step 2: Drop missing rows containing missing values
dataset.dropna(axis = 0, how = 'any', inplace = True)
```

Let's give a look the dataset cleaned :

In [64]:
```
print('Dataset cleaned after missing values :')
dataset.shape
```

```
Dataset cleaned after missing values :
```

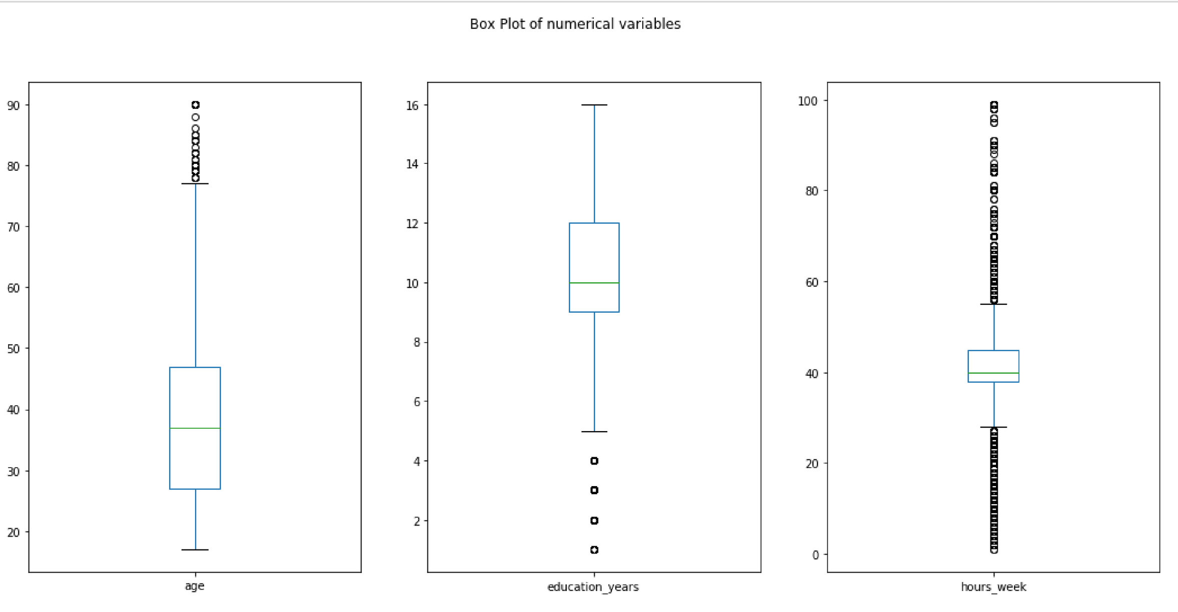
Out[64]:
```
(13977, 12)
```

This line to check the final numerosity being above 90% of the orginal data : 13977/15164 = 92% .

2.2 - Outliers

To have an idea of the distribution of the numerical variables and to reveal the presence of outliers, let's draw the boxplots with the pandas command *plot* :

In [66]:
```
# Explorative analysis : boxplots
dataset[['age','education_years','hours_week']].plot(kind='box', subplots=True, layout=(1,3), sharex=False, sharey=False, fig
size=(18,8), title='Box Plot of numerical variables')
plt.savefig('dataset_bp')
plt.show()
```



Box plots reveal the presence of outliers among the variables.
With the following statement, let's count how many lies over 3 standard deviations from the mean :

In [67]:
```
# Detect the values which are over 3 standard deviations from the mean.
num_variables = ['age','education_years','hours_week']
for var in num_variables:
    print(var, " outliers : ", len(dataset[np.abs(dataset[var]-dataset[var].mean()) > (3*dataset[var].std())]))
```

```
age  outliers :  56
education_years  outliers :  75
hours_week  outliers :  174
```

Then let's remove the outliers using the *'Z-score'*.
For each column of the dataset, first it computes the Z-score of each value, relative to the column mean and standard deviation. Then is taken the absolute of Z-score because the direction does not matter, only if it is below the threshold of 3 standard deviation. The method *.all(axis=1)* ensures that for each row, all column satisfy the constraint.

After removing the outliers, the dataset is pretty much the same size : from 13977 to 13675 observation.

In [68]:
```
dataset = dataset[(np.abs(stats.zscore(dataset[['age','education_years','hours_week']])) < 3).all(axis=1)]
print('Dataset cleaned after outliers :')
print(dataset.shape)
```

```
Dataset cleaned after outliers :
(13675, 12)
```

3 - Data Preparation

3.1 - Encoding Categorical variables

At this point we need to prepare the X feature matrix and encoding categorical variables.
One hot encoding will create dummy categorical variables and will add up to 83 features.

In [79]:
```
cat_columns = ['workclass','education','marital_status','occupation','relationship','sex','country','permit']
```

In [80]:
```
dataset_processed = pd.get_dummies(dataset, prefix_sep = '__',
                                   columns = cat_columns)
```

In [81]:
```
dataset_processed.shape
```

Out[81]:
```
(13675, 97)
```

In [82]:
```
# Statistics
dataset_processed.describe()
```

Out[82]:

	age	education_years	hours_week	stress	workclass__Federal-gov	workclass__Local-gov	workclass__Private	workclass__Self-emp-inc	workclass emp
count	13675.000000	13675.000000	13675.000000	13675.000000	13675.000000	13675.000000	13675.000000	13675.000000	13675.
mean	37.841463	10.162194	39.611408	0.219232	0.030713	0.073053	0.756782	0.030713	0.
std	12.939459	2.413037	10.944015	0.413741	0.172545	0.260233	0.429041	0.172545	0.
min	17.000000	3.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	27.000000	9.000000	38.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.
50%	37.000000	10.000000	40.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.
75%	47.000000	12.000000	44.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.
max	77.000000	16.000000	76.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.

8 rows × 97 columns

In [83]:
```
# X matrix and y vector definition :
X = dataset_processed.drop('stress', axis = 1)
y = dataset_processed['stress']
```

3.2 - Train/Test split

Let's create 4 splits for the original dataset : A Train setand a Test set split for X and y. The percentage of training / testing split is 75 to 25 :

In [85]:
```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [86]:
```
print('Training set X')
print(np.shape(X_train))
print('and y')
print(np.shape(y_train))
print('Test set X')
print(np.shape(X_test))
print('and y')
print(np.shape(y_test))
```

```
Training set X
(10256, 96)
and y
(10256,)
Test set X
(3419, 96)
and y
(3419,)
```

3.3 Model Selection / Grid Search

For this assignment we will use a **Random Forest classifier**.
Random Forest builds a series of parallel decision trees, each one trained on a subset of the original dataset. The split criteria is based on a random set of variables. Finally the results of the trees are pooled in order to get better performance.
Random Forest classifiers are robust to multicollinearity, so we can use the entire set of variables without reducing dimensionality.

Let's run a Gridsearch with 4 folds cross validation to find the hyperparameters for the model, optimized for the best F1 score :

In [87]:
```
# RANDOM FOREST 'rf'

classifier = RandomForestClassifier(random_state = 42)

parameters = {# resampling with or withut repcement
              'bootstrap': [True, False],
              # Maximum number of levels in tree
              'max_depth': [5, 10, 50, None],
              # Number of features to consider at every split
              'max_features': ['auto', 'sqrt'],
              # Minimum number of samples required at each leaf node
              'min_samples_leaf': [1, 2, 4],
              # Minimum number of samples required to split a node
              'min_samples_split': [2, 5, 10],
              # Number of trees in random forest
              'n_estimators': [10, 50, 100]}

gs_rf = GridSearchCV(classifier, parameters, cv = 4, scoring = 'f1', n_jobs = 20, verbose = 1, refit = True)

gs_rf = gs_rf.fit(X_train,y_train)
```

```
Fitting 4 folds for each of 432 candidates, totalling 1728 fits
```

```
[Parallel(n_jobs=20)]: Done  10 tasks      | elapsed:   12.5s
[Parallel(n_jobs=20)]: Done 160 tasks      | elapsed:   27.2s
[Parallel(n_jobs=20)]: Done 410 tasks      | elapsed:   39.6s
[Parallel(n_jobs=20)]: Done 760 tasks      | elapsed:  1.1min
[Parallel(n_jobs=20)]: Done 1210 tasks      | elapsed:  1.5min
[Parallel(n_jobs=20)]: Done 1728 out of 1728 | elapsed:  2.4min finished
```

In [88]:
```
#SUMMARIZE the results of your GRIDSEARCH
print('***GRIDSEARCH RESULTS***')

print("Best score: %f using %s" % (gs_rf.best_score_, gs_rf.best_params_))
means = gs_rf.cv_results_['mean_test_score']
stds = gs_rf.cv_results_['std_test_score']
params = gs_rf.cv_results_['params']

#for mean, stdev, param in zip(means, stds, params):
#    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
***GRIDSEARCH RESULTS***
Best score: 0.631948 using {'bootstrap': True, 'max_depth': 50, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_sampl
es_split': 5, 'n_estimators': 50}
```

Let's save the model and evaluate its performance.

In [90]:
```
#SAVE BEST MODEL in the variable best_model
best_model_rf = gs_rf.best_estimator_
best_model_rf
```

Out[90]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=50, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=2, min_samples_split=5,
            min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
            oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Training Performance

The model gets a F1 score of 0.71 and 0.89 accuracy on training set.

In [91]:
```
# Training with Grid Search best model
y_hat = best_model_rf.predict(X_train)
cm = confusion_matrix(y_train, y_hat)
print('Confusion Matrix')
print(cm)
tn = cm[0,0]
fn = cm[0,1]
fp = cm[1,0]
tp = cm[1,1]
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = 2*((precision*recall)/(precision+recall))
acc = np.sum(cm.diagonal()/np.sum(cm))
print('Best Model Training')
print('F1 =',f1,'\nAccuracy =',acc)
```

```
Confusion Matrix
[[7720  284]
 [ 744 1508]]
Best Model Training
F1 = 0.7457962413452027
Accuracy = 0.8997659906396256
```

In [92]:
```
# Model Validation
y_hat_val = best_model_rf.predict(X_test)
cm = confusion_matrix(y_test, y_hat_val)
print('Confusion Matrix')
print(cm)
tn = cm[0,0]
fn = cm[0,1]
fp = cm[1,0]
tp = cm[1,1]
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = 2*((precision*recall)/(precision+recall))
acc = np.sum(cm.diagonal()/np.sum(cm))
print('Best Model Validation')
print('F1 =',f1,'\nAccuracy =',acc)
```

```
Confusion Matrix
[[2501  172]
 [ 345  401]]
Best Model Validation
F1 = 0.6080363912054587
Accuracy = 0.8487861947937994
```

Testing Performance

The model gets a F1 score of 0.6 and 0.845 accuracy on test set.

4 - Import Validation Set

In [ ]: